Abstract

This report details the microarchitecture design and analysis of ASIC version of a RISC-V Processor. RISC-V is a Reduced Instruction Set Computing (RISC) architecture, which means that it uses a small set of simple instructions that can be executed quickly. The RISC-V ISA is designed with a modular structure, allowing designers to choose the features they need for their specific application. This modularity makes it easy to customize the processor to suit specific requirements, such as power consumption, performance, or security.

In this project SCR1 is implemented The RISC-V SCR1 is a configurable 32-bit RISC-V processor core developed by Syntacore, a Russian company specializing in processor IP cores. The SCR1 core is designed to be highly configurable, allowing it to be optimized for a wide range of applications, from low-power IoT devices to high-performance computing systems.

The microprocessor implements the RV32I instruction sets and has a 3- stage pipelining architecture of the RISC-V specifications. It shows integration of UART and I2C peripherals into the Digital Core of RISC-V processor. The whole process entails logical and physical Synthesis, Placement, CTS and Routing of the design performed using 45nm, which relies on the Cadence EDA framework. Along with this, the analysis of various design factors affecting the performance of the final chip such as power, area and timing is also considered. This design has a whole area of 6998809 μm^2 and consumes 24.64 mW power.

Table of Contents

Table of Contents

List of Figures

List of Figures

List of Figures

CHAPTER 1

INTRODUCTION

1.1 Introduction

Integrated circuits (ICs) revolutionized electronics, and today they are present in most technology applications and in a great part the industrial products. For example, microprocessors, almost every electronic device available today, are powered by a general-purpose microprocessor: computers, smartphones, digital televisions, smart speakers, etc. According to the Statista Research Department, the worldwide integrated circuit market has reached 361.23 billion U.S. dollars in revenue, in 2020. Furthermore, the estimated market growth in 2021 is by over 20 percent to 436.37 billion U.S. dollars. Application-Specific Integrated Circuits (ASICs) are ICs manufactured for two main purposes:(1) To meet user's specifications for the demands of a particular system; or (2) For reuse, so that several other macro systems can be designed considering the already finalized ASIC as a component. A traditional design methodology for ASICs is the standard cell one, which consists of mapping a complex circuit into pre-designed logic cells. The cell library contains the description of the layout of each logic cell.

One of the key advantages of integrated circuits is their small size and low power consumption, which makes them ideal for use in portable electronic devices. Additionally, the ability to mass-produce integrated circuits using automated manufacturing techniques has made them both inexpensive and widely available. There are many types of integrated circuits, like digital logic circuits, analog circuits, memory circuits, and microprocessors. The specific type of integrated circuit used depends on the requirements of the application and the desired functionality of the device. Integrated circuits (ICs) are classified into two main categories: digital and analog. Digital ICs are used to process binary signals, which are either 0 or 1, and perform logic operations such as AND, OR, and NOT. They are widely used in computers, communication systems, and other digital devices.

Integrated circuits are typically manufactured using a process called photolithography, which involves etching microscopic patterns onto a silicon wafer using light-sensitive materials and chemicals. This process allows IC manufacturers to create complex electronic circuits that are incredibly small and precise. The integrated circuits have revolutionized the electronics industry and have enabled the development of many of the modern devices we use today, from smartphones and laptops to medical devices and automotive systems. [10]

1.2 ASIC Design Flow.

An application-specific integrated circuit (ASIC) is an integrated circuit(IC) customized for a particular use, rather than intended for general-purpose use. In today's world, ASICs offer many advantages over off-the-shelf devices.

1. Smaller die size leads to board size reduction.
2. Reduced power consumption, less heat dissipation.
3. Lower costs under mass production.
4. Improved performance.
5. Better radiation tolerance.
6. Improved testability.
7. Enhanced reliability.
8. Proprietary design implementation.

1.2.1 Standard-Cell–Based ASIC

A cell-based ASIC uses predefined logic cells like AND gates, OR gates, multiplexers, and flip-flops known as standard cells. The flexible blocks in a CBIC are built of rows of standard cells. Placement of the standard cells and the interconnect is defined by an ASIC designer in a CBIC. The advantage of CBICs is that they can be designed in less time with small amount of money compared to full-custom ASICs, and also the most important thing is it reduce the risk by using a predesigned, pretested, and precharacterized standard-cell library which can be optimized individually. At the same time, the disadvantages are the time or expense of designing or buying the standard-cell library and the time needed to fabricate all layers of the ASIC for each new design. Figure-1.1 shows a CBIC.

Standard-cell-based ASIC design offers several advantages over other types of digital circuit design, including high performance, low power consumption, and flexibility in design. Each standard cell in the library is constructed using full-custom design methods, but you can use these predesigned and pre-characterized circuits without having to do any full custom design yourself. This design style gives you the same performance and a flexibility advantage of a full-custom ASIC but reduces design time and reduces risk. [1]

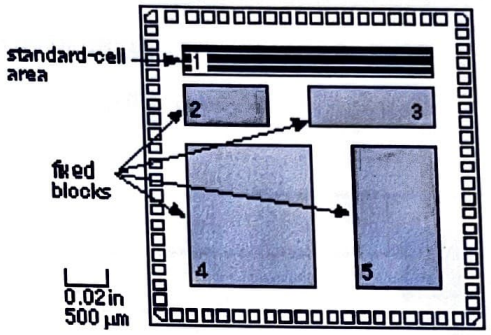

Figure: 1.1-Block diagram of Cell based ASIC (CBIC) [1]

1.2.2 Need for Low Power ASIC

For early digital circuits, high speed and minimum area were the main design constraints.Most of the EDA tools were designed specifically to meet these criteria. Power consumption was never highly visible. Nowadays, the area reduction of digital circuits isno longer a big issue as with the latest sub-micron techniques, many millions of transistors can be fit in a single IC. Smaller chip size eventually leads to high demand for portable and handheld devices. More and more applications are battery powered, and low power IC's are the key to extend the usage time in between battery recharge, and in turn increase battery life and reliability of the product. Also in submicron technologies, there is a limitation on the proper functioning of circuits due to heat generated by power dissipation. Market forces are demanding low power for not only longer battery life but also reliability, portability, performance, cost and time to market. This is very true in the field of personal computing devices, wireless communications systems, home entertainment systems, which are becoming popular now-a-days. Implantable medical devices, such as pace maker, deep brain system for Parkinson's disease, and spinal cord stimulator for pain management, particularly need to dissipate less power for longer battery life and improved component reliability and safety.

As process technology reduces into 90nm and below, performance and density are taken to new levels, yet power loss in both switching and leakage makes designing with these devices a major challenge. Leakage power reduction is essential in sustaining the scalingof the CMOS process. Leakage power is now becoming proportional to dynamic or switching power loss as shown in Figure 1.2. While lowering of the threshold voltage leads to significant increase in sub-

threshold leakage current, the increase in gate tunneling leakage current is caused by thinner gate oxides. While scaling improvestransistor density, functionality, and higher performance on chip, it also results in power dissipation increase. Therefore, it has become necessary to use new techniques tomanage energy at the system level. [2]

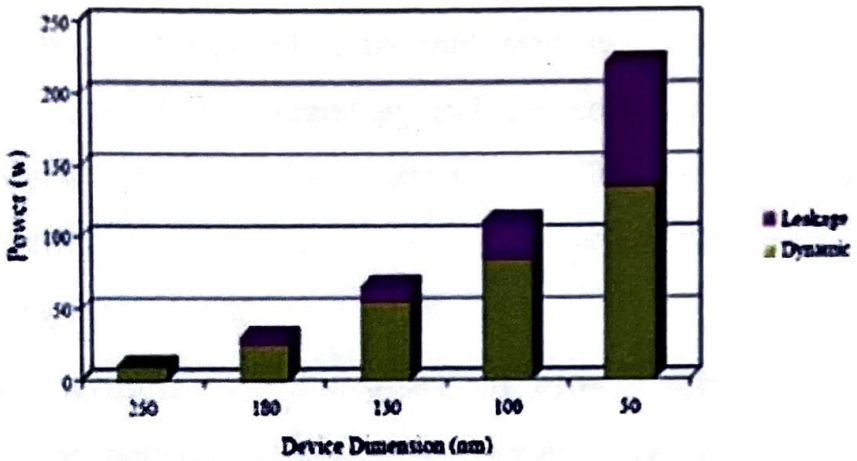

Figure: 1.2- Graph Dynamic and Leakage Power Comparison [2]

Bottom line, low power budget has become one of the most important design parametersfor VLSI (Very Large Scalė Integration) systems. [2]

1.2.3 ASIC Design Flow:

ASIC (Application-Specific Integrated Circuit) design flow is the process of designing and manufacturing a custom integrated circuit for a specific application, such as a microcontroller, digital signal processor, or communication system. Figure 1.3 shows the ASIC design flow. The ASIC design flow typically involves several stages, including:

1. **Design Specification:** This stage involves defining the requirements and specifications for the ASIC design. This includes identifying the target application, performance requirements, power consumption, and other key parameters.
2. **RTL Design:** The RTL design stage involves creating the ASIC design using a hardware description language (HDL) such as Verilog or VHDL. This stage includes creating the architecture and interfaces for the ASIC, as well as defining the logic and functionality.
3. **Functional Verification:** The functional verification stage involves testing the RTL design to ensure that it meets the required functionality and performance specifications. This is typically done using simulation tools and test benches.

4. **Synthesis:** The synthesis stage involves translating the RTL design into a gate-level netlist that can be physically implemented on an ASIC. This involves optimizing the design for area, power, and performance using synthesis tools such as Synopsys Design Compiler or Cadence Genus.
5. **Place and Route:** The place and route stage involves physically placing the gates on the target ASIC technology and connecting them together. This involves optimizing the physical layout and wiring of the design for the target technology and performance requirements using tools such as Cadence Innovus or Synopsys IC Compiler.
6. **Timing Analysis**: The timing analysis stage involves ensuring that the design meets the required frequency and performance specifications. This involves analyzing the timing paths in the design and optimizing them for maximum performance and minimum delay using tools such as Synopsys Prime Time or Cadence Tempus.
7. **Physical Verification:** The physical verification stage involves ensuring that the design meets the manufacturing requirements and constraints of the target technology. This includes checking for issues such as DRC violations, LVS errors, and other layout-related issues.
8. **Tapeout:** The tapeout stage involves preparing the final design files and submitting them to the foundry for manufacturing. This includes creating the GDSII files, which contain the physical layout of the design, as well as documentation and other supporting materials.
9. **Post-Silicon Validation:** The post-silicon validation stage involves testing the manufactured ASIC to ensure that it meets the required functionality and performance specifications. This involves running various test cases and verifying that the output matches the expected results.

The ASIC design flow is a complex process that requires specialized knowledge and expertise in areas such as digital design, analog design, verification, and manufacturing. ASIC design companies typically employ a team of engineers with expertise in these areas to develop custom ASICs for their clients. [2]

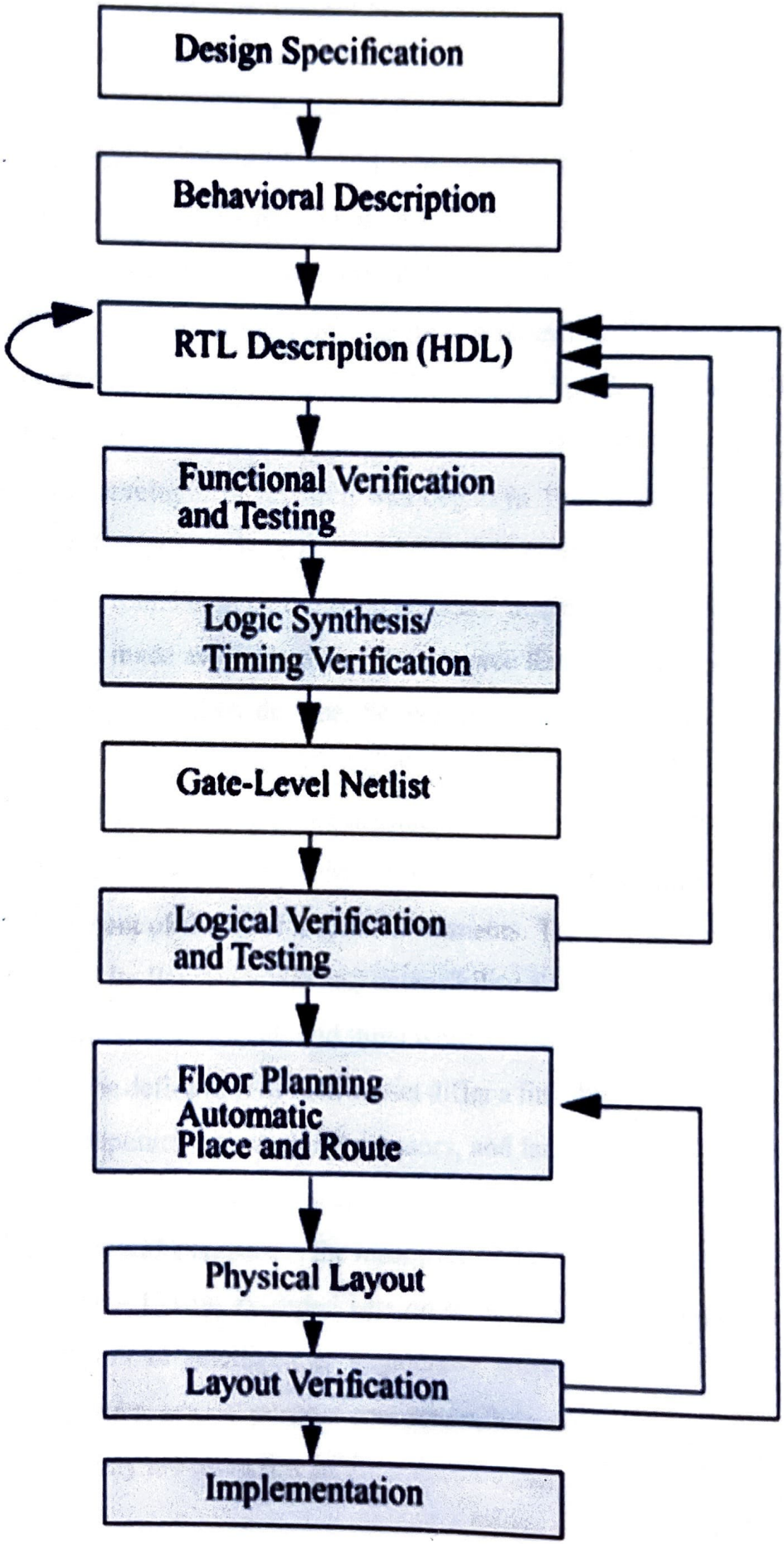

Figure 1.3-ASIC design flow [3]

1.3 RISC-V(Reduced Instruction Set Computer) Processor

The word RISC refers to the "reduced instruction set computer" in the RISC V processor, which only executes a small number of computer instructions, while "V" refers to the fifth generation. It is an ISA (instruction set architecture) for hardware that is open source and is based on the well-known RISC idea. Unlike previous ISA designs, this one is offered under an open-source license. As a result, several manufacturing firms have announced and offered RISC-V hardware together with open-source operating systems. Professor David Patterson of the University of California, Berkeley created the RISC sometime in the 1980s. In two volumes titled "Computer Organization and Design" and "Computer Architecture at Stanford University," professors John Hennessy and David contributed their work. As a result, they were given the ACM A.M. Turing Prize in 2017.

The RISC fifth-generation development research was begun in 1980 and continued until it was ultimately designated as RISC-V, which is pronounced risk five. Reduced Instruction Set Computer (RISC) concepts provide the foundation of the open standard instruction set architecture used by RISC-V. The RISC-V ISA is made available under open-source licenses that don't charge a fee, in contrast to the majority of previous ISA designs. Several businesses are promoting or supplying RISC-V hardware.

The RISC-V ISA stands out for its load store architecture, bit patterns that make it easier for CPU multiplexers to function, IEEE 754 floating-point, and architectural neutrality. To speed up sign extension, it fixes the placement of the most important elements. The instruction set is intended to be used in a variety of ways. Its flexible width and extensibility allow for the constant addition of new encoding bits. It supports several subsets and three word-widths, including 32, 64, and 128 bits . For the three word-widths, the definitions of each subset differ a little bit. Small, embedded devices, desktop computers, supercomputers with vector processors, and large-scale 19- inch rack-mounted parallel computers are all supported by the subsets.

Because a shortage of memory address space is the most irrecoverable fault in instruction set design, the instruction set space for the 128-bit extended version of the ISA was set aside due to 60 years of industry experience. Artificial intelligence, augmented reality, automotive, cloud servers, computer devices and controllers, general purpose processors, Internet of Things, machine learning, network edge, and virtual reality are just a few of the industries that employ RISC V.[5]

1.4 Background

The setting of voluntary standards in engineering dates back more than a century. The release of RISC-V to the open community in 2015 – for both standardization and ongoing improvement through open collaboration – marked the first time the hardware community embraced open-source standards and collaboration at this level. Ensuring long term access to, and development of, the RISC-V Instruction Set Architecture is a strategic choice. RISC-V International is wholly committed to design freedom, choice, and flexibility, and supports open architecture extensions to the RISC-V ISA. We do not support work on alternative versions of RISC-V.

Our process is modelled on decades of success demonstrated by the open-source community. RISC-V International's intellectual property is developed and contributed collectively by members, and governed by the terms of open-source licenses. Once IP is provided globally in this way, it is permanently open and remains available for all. Global success of any architecture is driven by the shared collective interest of a community. Open source and global standards have a long history of success because they have a license framework that ensures anyone, anywhere can have ongoing access to them. Prof. Krste Asanović and graduate students Yunsup Lee and Andrew Waterman started the RISC-V instruction set in May 2010 as part of the Parallel Computing Laboratory (Par Lab) at UC Berkeley, of which Prof. David Patterson was Director. The Chisel hardware construction language that was used to design many RISC-V processors was also developed in the Par Lab. The Par Lab was a five-year project to advance parallel computing funded by Intel and Microsoft for $10M over 5 years, from 2008 to 2013 1. It also received funding from several other companies and the State of California. While the project overall did not have Federal funding, Yunsup Lee and Andrew Waterman received some funding from the DARPA POEM photonics project, which funded some of the processor implementation development (but not the RISC-V ISA). The funds were 6.1 basic research via MIT as prime contract with the International Computer Science Institute as the subcontract. All the projects in the Par Lab were open source using the Berkeley Software Distribution (BSD) license, including RISC-V and Chisel.

The ISA specification itself (i.e., the encoding of the instruction set) was effectively put into the public domain when the ISA tech reports were published, though the actual technical report text was later put under a Creative Commons license to allow it to be improved by external contributors including the RISC-V Foundation.

The US developers conceived the revolutionary instruction set architecture (ISA) known as RISC-V in 2010. Grounded in reduced instruction set computer (RISC) principles, it's a common, open-source, and completely free ISA that can be used to develop software and hardware. These attributes are just part of what makes the architecture unique and attractive to developers and manufacturers.

While it's far from new, momentum around RISC-V is continuing to accelerate, with various services, technologies, and products that leverage this architecture emerging in the past year alone. According to Deloitte, use of RISC-V's open-chip processors was expected to double this year and is set to double again in 2023.

The creation of RISC-V brought with it a novel approach to silicon design. Unlike the expensive and decades-old legacy ISAs that are not designed to handle the latest compute workloads, this new approach was created with flexibility, modularity, and scalability in mind – without cost. This increased accessibility drives more competition, leading to a new generation of innovation that will benefit everyone.

And it's not just the big players that stand to benefit – smaller developers and manufacturers can use RISC-V to design and build hardware without the usual costs associated with licensing proprietary ISAs or paying royalties. As a result, companies of all sizes can innovate, improve, and create using the architecture, allowing everyone to have a fair crack of the whip when it comes to putting the best products on the market. This spells great things for both the industry's advancement and the benefits consumers will get from the new and innovative devices on the market.

With applications including artificial intelligence, IoT, automotive, and cloud servers, to name a few, the possibilities for the architecture and what it can enable are limitless. But outside of its benefits to the future /of these industries, the open-source nature of RISC-V also brings more specific advantages to the manufacturers that adopt and enter this ecosystem and the community of engineers that use it.

Artificial intelligence, the Internet of Things, and machine learning are ever more commonplace in everyday devices – and the ability to create custom processors capable of handling the performance and power requirements of these newer and mightier workloads is essential.

According to research firm Semico, the demand for these technologies is driving up the usage of this innovative architecture. The company predicts that the number of chips that include at least some RISC-V technology will grow 73.6 percent annually through 2027.

For the engineers and developers, RISC-V provides greater flexibility to customize processors through an ISA that natively supports extensions. This allows designs to uniquely align their solution with the device requirements and stay within the RISC-V definition—no matter what application, performance, or power requirements the chipset needs to meet.

At the same time, when new RISC-V-compatible architecture is developed, it allows engineers to access the technical advantages of platforms entering the RISC-V community while still using tools and processes they're familiar with. Entering this community brings its own benefits by adding to a growing set of shared tools and development resources within the community.

Manufacturers and developers are continuing to embrace RISC-V and recognize its numerous benefits. As this community of like-minded people and organizations grows, we can expect a tidal wave of applications built using this flexible and scalable architecture in the near future. The excitement around this architecture is well-founded.

RISC-V also encourages academic usage. The simplicity of the integer subset permits basic student exercises, and is a simple enough ISA to enable software to control research machines. The variable-length ISA provides room for instruction set extensions for both student exercises and research, and the separated privileged instruction set permits research in operating system support without redesigning compilers. RISC-V's open intellectual property paradigm allows derivative designs to be published, reused, and modified. [5]

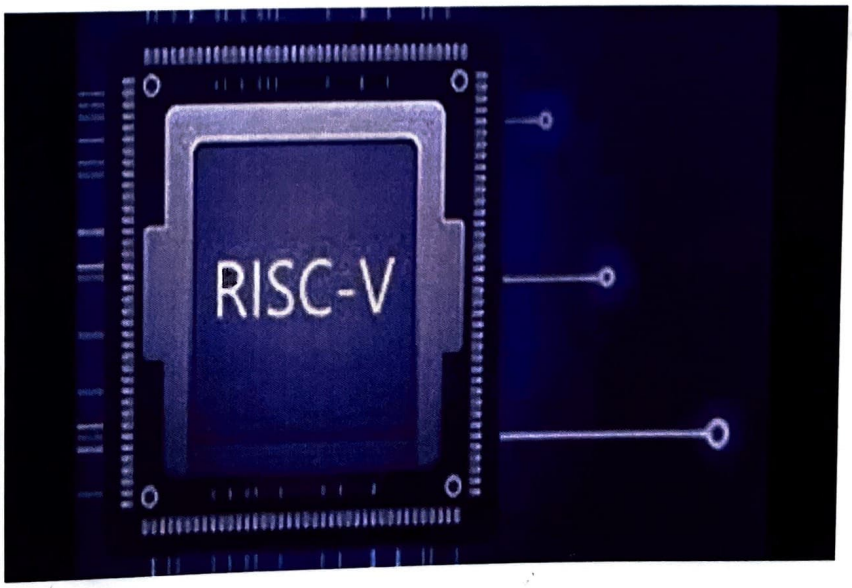

Figure 1.4- RISC-V processor prototype[4]

1.5 Motivation

Microprocessors and Microcontrollers have been designed around two philosophies: Complex Instruction Set Computer (CISC) and Reduced Instruction Set Computer (RISC). The CISC concept is an approach to the Instruction Set Architecture (ISA) design that emphasizes doing more with each instruction using a wide variety of addressing modes, variable number of operands in various locations in its Instruction Set.

As a result, the instructions are of widely varying lengths and execution times thus demanding a very complex Control Unit, which occupies a large real estate on chip. On the other hand, the RISC Processor have reduced number of Instructions, fixed instruction length, more general-purpose registers, load-store architecture and simplified addressing modes which makes individual instructions execute faster, achieve a net gain in performance and an overall simpler design with less silicon consumption as compared to CISC.

The above features make the RISC design ideally suited to participate in a powerful trend in the embedded Processor market - the "system-on-a-chip". The RISC-V processor is a proven ISA and follows established RISC design principles which has single-cycle instructions and also uses a load-store architecture. It features a simple, stable, software-centric design (small, fixed base with modular fixed standard extensions) and is modular, layered and extensible, allowing for software and hardware freedom on architecture hence it is flexible and scalable (i.e., suitable for microcontrollers to personal computers to supercomputers).

The RISC-V processor has 32-bit and 64-bit variants and extensions to support floating point instructions. It is supported by various language compilers (e.g., GNU Compiler Collection and Linux operating system) and offers a range of hardware support from microcontrollers to systems on module, systems on chip and field programmable gate arrays which accelerates the design-to-market timeline through collaboration and open source IP reuse.

RISC-V is a flexible and versatile architecture that can be used in a wide range of applications. Its open-source nature and customizable design make it an attractive option for companies and researchers looking to innovate and differentiate their products.

This project presents the Digital Core of the RISC-V processor. The Digital core is integrated with peripherals such as UART and I2C. This project also aims to prioritize the cost, customization and low power in implementation of RISC architecture on a system on chip.

1.6 Objectives

The RISC V processor s used in embedded systems, artificial intelligence & machine learning and has a wide range of applications which make it a major technology in demand of the industry. These processors are used in high-performance-based embedded system applications. This processor is important as it permits smaller device manufacturers to design hardware without paying. This processor simply allows the researchers and developers to design as well as research with a freely available ISA or instruction set architecture.

Like many other ISAs, the specification defines different levels of instruction sets. This includes 32 and 64-bit variants, and extensions to support floating point instructions. This allows versions to be developed that are suited to a range of applications from small embedded microcontrollers to desktop personal computers and supercomputers with vector processors. Hence due to the rising demand of this processor, we aim to build a functioning RISC-V processor.

The main objective of the project is to implement the digital block of RISC-V processor. The project aims at:

1. Familiarize with the ASIC flow that includes RTL, verification, synthesis (SDC, genus), placement and routing (innovus), timing (tempus) analysis, and PV checks.
2. To implement digital block of the RISC processor.
3. To integrate the blocks to form a RISC SoC.

The prototype is intended to be converted to a product that is designed with the combination of other digital blocks and finally combined with the analog blocks to build a full-fledged RISC V processor which can be called as the final product.

CHAPTER 3

DESIGN IMPLEMENTATION AND METHODOLOGY

3.1 Overview

Implementing a digital block for a RISC-V SoC (system on chip) involves several steps, including designing the block architecture, selecting the appropriate tools and technologies, implementing the design using a hardware description language (HDL), and verifying the functionality of the block. Here is a high-level overview of the process:

- Design the block architecture: This involves selecting the required RISC-V processor cores, peripherals, memory interfaces, and other components that will be integrated into the digital block.
- Select the tools and technologies: There are several tools and technologies available for designing and implementing digital blocks, including hardware description languages (HDLs) like Verilog and VHDL, Electronic Design Automation (EDA) tools, and FPGA or ASIC technologies.
- Implement the design using an HDL: Once the block architecture is finalized, the next step is to implement the design using an HDL. This involves writing the code that describes the behavior of each component in the block.
- Verify the functionality of the block: After the HDL code is written, it is important to verify the functionality of the digital block. This can be done using simulation tools and by running test cases on the block.
- Integrate the digital block into the SoC: Once the digital block has been designed, implemented, and verified, it can be integrated into the RISC-V SoC along with other blocks that make up the system.

It is worth noting that the process of designing and implementing a digital block for a RISC-V SoC can be complex and time-consuming. It may require expertise in hardware design, digital signal processing, and computer architecture. Therefore, it is important to carefully plan and execute each step in the process to ensure a successful outcome.

3.2 Aramb-180

Figure 4.1 shows the block diagram of ARAMB-180 which includes different blocks such as a

Digital Core, Analog Core and Memory. This project mainly focuses on the Digital core part of the RISC-V processor. The Digital core is integrated with UART and I2C.

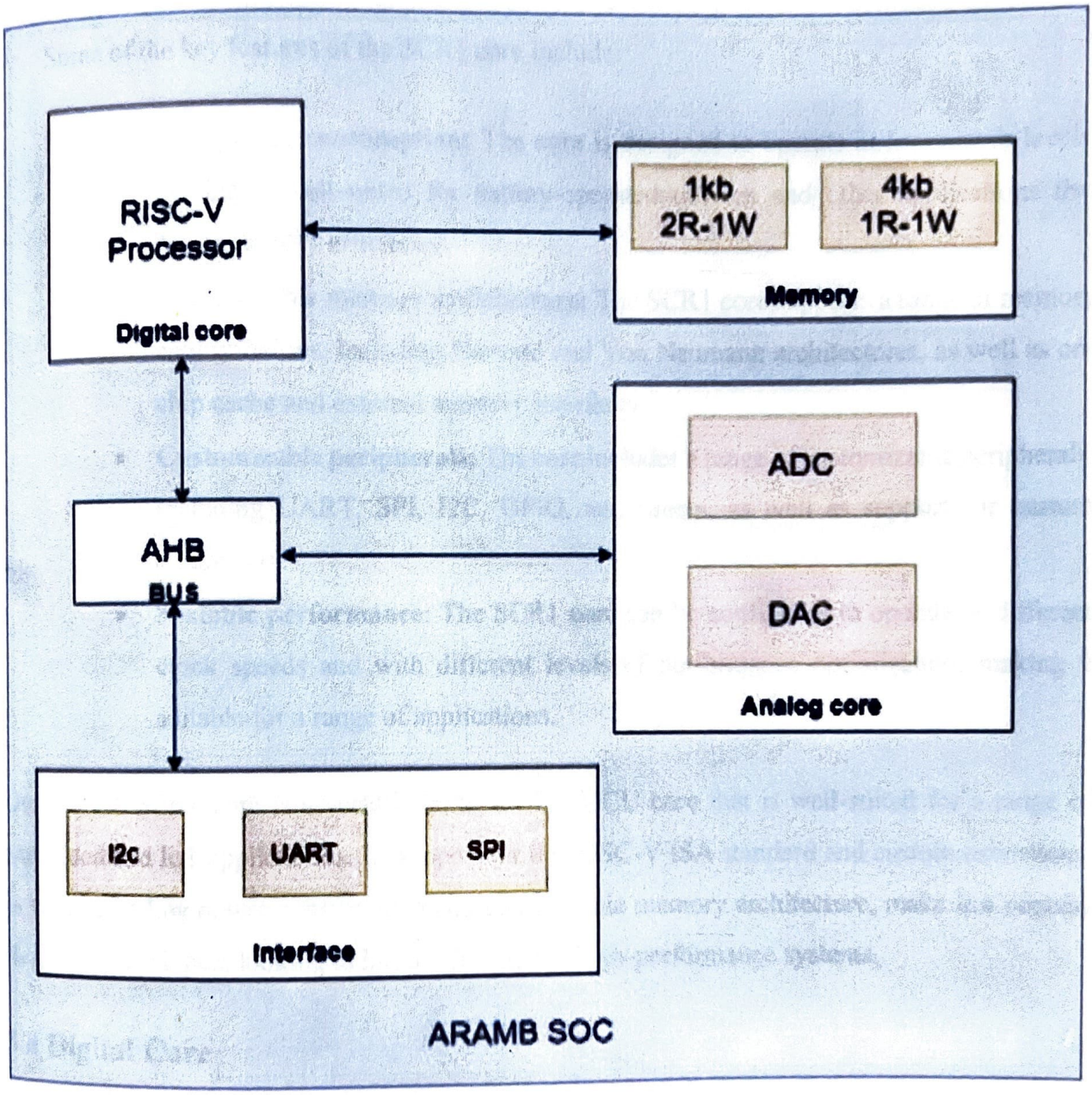

Figure 3.1-Block Diagram of ARAMB 180

3.3 RISC-V SYNTACORE SCR1

SCR1 is a 32-bit RISC-V microcontroller (MCU) core developed by Syntacore, a Russian semiconductor IP company. RISC-V is an open-source instruction set architecture (ISA) that has gained popularity in recent years due to its flexibility, scalability, and ability to support custom extensions. The SCR1 core is designed for use in embedded systems, IoT devices, and other

applications that require low power consumption and high performance. It is a fully-featured MCU core that supports the RISC-V ISA standard, including the RV32I base instruction set, as well as hardware multiply/divide, atomic operations, and a range of custom extensions.

Some of the key features of the SCR1 core include:

- **Low power consumption:** The core is designed to operate at low power levels, making it well-suited for battery-operated devices and other applications that require energy efficiency.
- **Configurable memory architecture:** The SCR1 core supports a range of memory configurations, including Harvard and Von Neumann architectures, as well as on-chip cache and external memory interfaces.
- **Customizable peripherals:** The core includes a range of customizable peripherals, including UART, SPI, I2C, GPIO, and timers, as well as support for custom extensions.
- **Scalable performance**: The SCR1 core can be configured to operate at different clock speeds and with different levels of performance optimization, making it suitable for a range of applications.

Overall, the SCR1 core is a versatile and flexible MCU core that is well-suited for a range of embedded and IoT applications. Its support for the RISC-V ISA standard and custom extensions, as well as its low power consumption and configurable memory architecture, make it a popular choice for developers looking to build efficient and high-performance systems.

3.4 Digital Core

The core is load-store architecture, where only load and store instructions access memory and arithmetic instructions only operate on integer registers. The core provides a 32-bit user address space that is byte-addressed and little-endian. The execution environment will define what portions of the address space are legal to access. Digital cores is designed to be easily integrated into a wider range of systems and applications, enabling faster time-to-market and reducing development costs.

Figure 4.2 shows the block diagram of the Digital core integrated with UART and I2c.Figure 4.3 shows the block diagram of the Digital core.

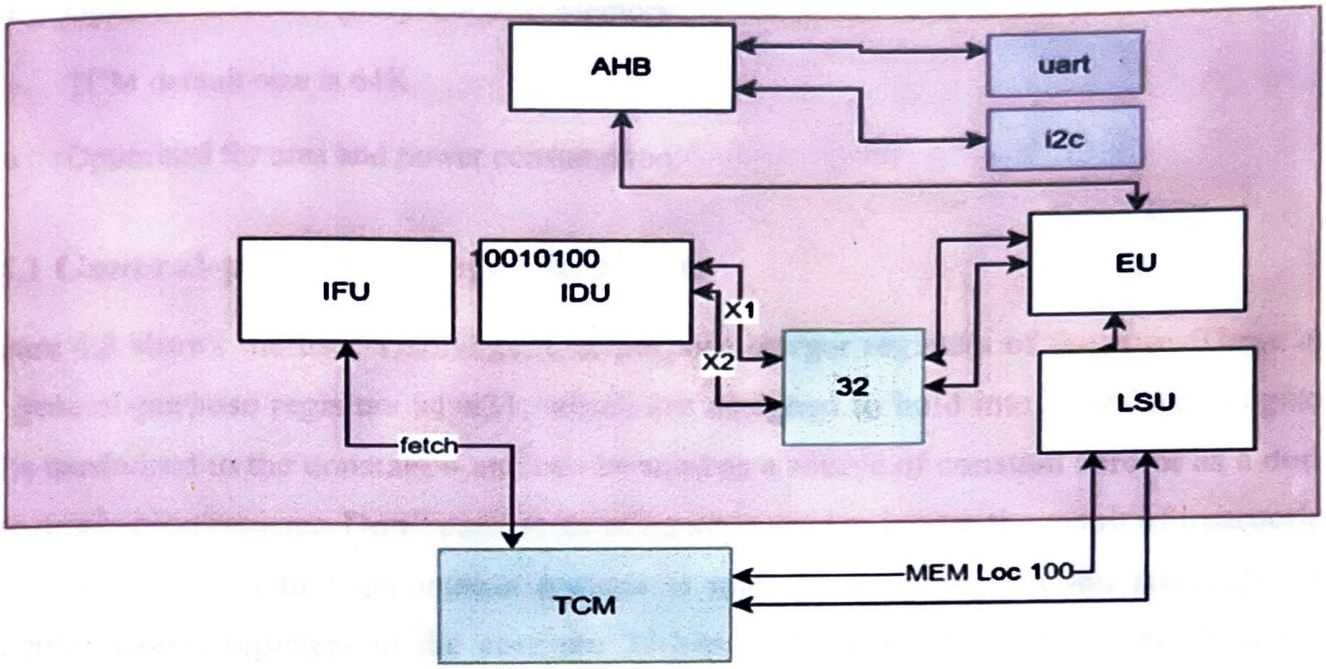

Figure 3.2 Block Diagram of Digital Core

Digital core consists of:

- Instruction Fetch Unit (IFU).
- Instruction Decode Unit (IDU).
- Execution Unit with integer ALU (EXU).
- Load-Store Unit (LSU).
- Control-Status Register (CSR).
- Memory-mapped Timer.
- Tightly-Coupled Memory (TCM).

Key features:

- Harvard architecture (separate instruction and data buses)
- Machine privilege level only
- 32- 32-bit general purpose integer registers
- Instruction set is RV32I standard extensions
- 47 Integer (32-bit) instructions
- stage pipeline implementation
- 32-bit AXI4/AHB-Lite external memory interface

- Optional on-chip Tightly-Coupled Memory
- TCM default size is 64K.
- Optimized for area and power consumption.

3.4.1 General-purpose Integer Registers

Figure 4.3 shows the user-visible general-purpose integer registers of the core. There are 31 general-purpose registers x1-x31, which are designed to hold integer values. Register x0 is hardwired to the constant 0 and can be used as a source of constant zero or as a don't care destination register. Don't care destination x0 is used to ignore the result of instruction execution provided that destination register is mandatory for instruction structure. All general-purpose registers in the core are 32-bits wide. The core implements 32-bit pc register, which is used as program counter, meaning that it holds the address of the current instruction. [15]

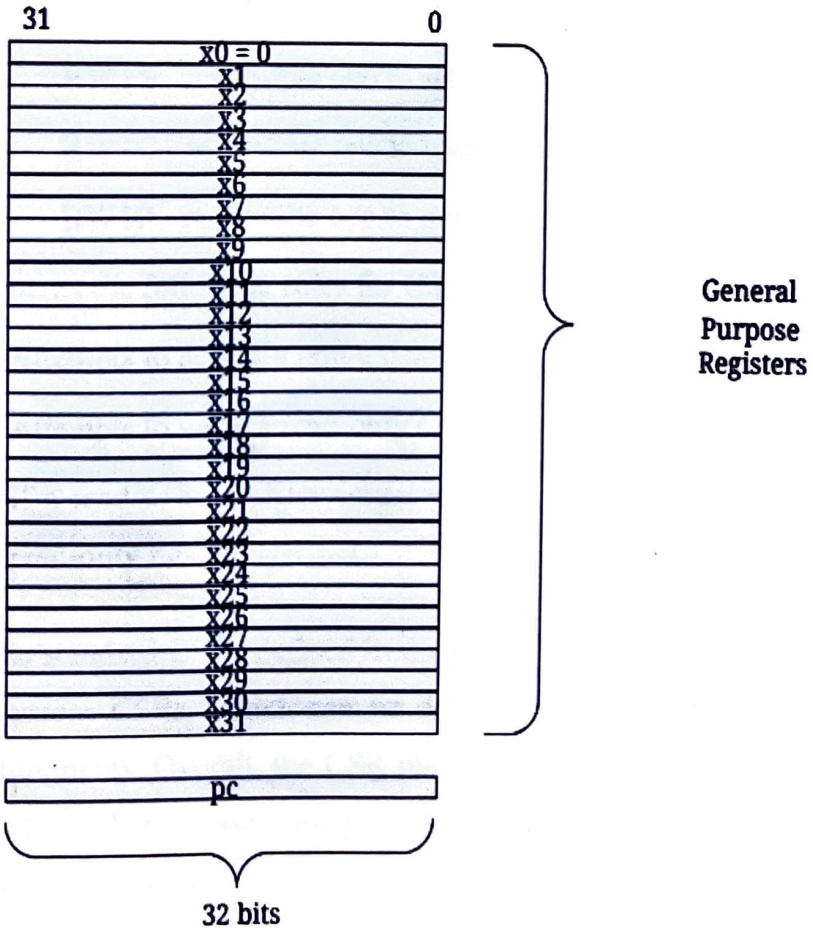

Figure 3.3: General-purpose integer registers[15]

3.4.2 Control and Status Registers

According to the RISC-V specification, the core uses 12-bit encoding space to address up to 4096 control/status registers (CSR) in the instructions which atomically read and modify CSRs. The core implements subset of CSRs according to the mapping shown in the next paragraphs.

The core follows RISC-V convention, where the upper 4 bits of the CSR address are used to encode the read and write accessibility of the CSRs according to the privilege level. The top two bits [11:10] indicate whether the register is read/write (00, 01, or 10) or read-only (11). The next two bits [9:8] indicate the lowest privilege level that can access the CSR.

The following definitions are used to designate bit or bit field properties throughout the individual CSR descriptions:

- RO - read only (write attempt results in illegal instruction exception)
- QRO - quiet read only (write attempt is ignored)
- RZ - read as zero
- RW - read/write
- RW1S - read/write one to set
- RW1C - read/write one to clear
- RW1P - read/write one to pulse

The core implements the following rules for CSR access:

1. Attempts to access a non-existent CSR raise an illegal instruction exception;
2. Attempts to write a read-only CSR also raise illegal instruction exception;
3. If a read/write register contains some bits that are read-only, then writes to the read-only bits are ignored.

In addition to the standard CSRs defined in the RISC-V ISA specification, the CSR architecture also supports custom CSRs, which can be defined by the system designed to meet specific application requirements. Overall, the CSR plays a critical role in managing and controlling the behaviour of the RISC-V processor, and provides a powerful tool for system designers to optimize and customize the behaviour of their systems. [15]

2. Data access width and alignment

The core supports following memory access widths: 32-bit words for instruction and data memory, 16-bit half words for data memory only, 8-bit bytes for data memory only. The core considers data memory as a contiguous collection of bytes numbered in ascending order in the range 0x00000000-0xFFFFFFFF (32-bit address). The core considers instruction memory as a contiguous collection of 32-bit words for base 32-bit instruction set (RV32I) or as a contiguous collection of 16-bit half words for compact instruction set (RV32C). Instructions in memory must be aligned to 4-byte boundary or 2-byte boundary correspondingly. Byte numbering in memory starts from 0. In case of compact instruction set the last instruction address is 0xFFFFFFFE. In case of non-compact instruction set the last instruction address is 0xFFFFFFFC. Instruction fetch from memory is physically done as 32-bit words aligned to 4-byte boundary ignoring any unnecessary portion of the word during instruction decode.

3. Subroutine call and stack behavior

The core supports subroutine call and stack handling with implemented base and compact instruction sets. Register x1 (ABI name is "ra") is recommended to be used as a return address by RISC-V calling convention, and register x2 (ABI name is "sp") is recommended to be used as a stack pointer. However, any subset of general purpose registers x1..x31 can be used for these purposes if needed.

4. Memory access ordering

The core uses strong memory access ordering, meaning that the sequence and the number of memory accesses are guaranteed to correspond one-to-one to underlying sequence of instructions executed. Given that, FENCE instruction is executed as NOP, FENCE.I instruction flushes the instruction fetch queue.

5. System memory map

A system memory map is a diagram or chart that shows the layout and organization of memory in a computer or embedded system. It provides a visual representation of the address space and the various memory regions used by the system.

The core implements Harvard architecture characterized by independent access to instruction memory and data memory through dedicated external memory interfaces. Figure 4.5 shows the illustrative view of the system memory map for the core.

The core provides dual-port tightly-coupled memory (TCM) which can be used for both instructions and data. TCM is characterized by short memory response to support time critical code and/or data of the application. TCM is mapped to system memory map with fixed base address 0x00480000. Detailed description of TCM is given in Tightly-Coupled Memory section of this specification.[15]

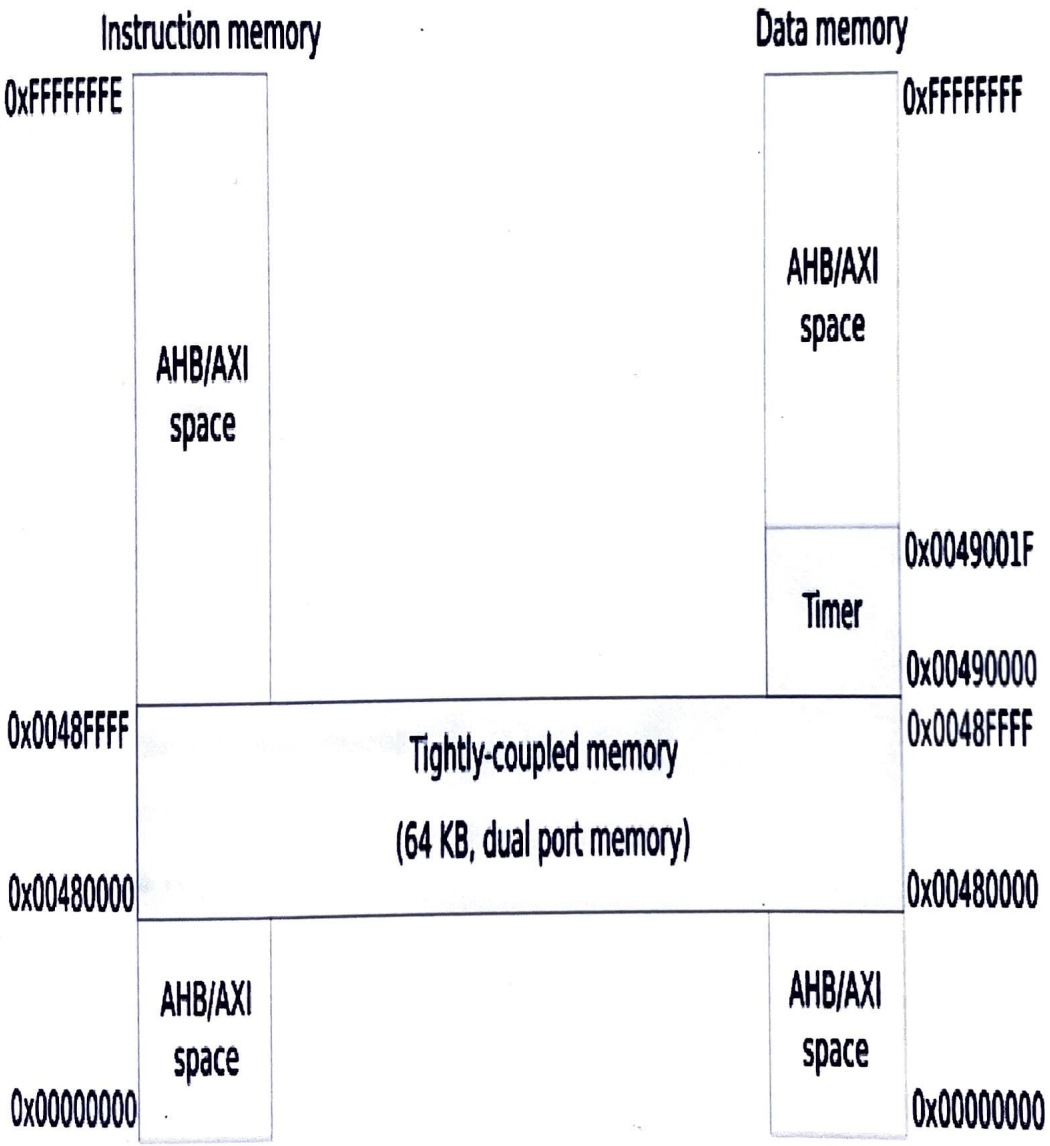

Figure 3.5 System memory map[15]

6. Tightly-Coupled Memory

Tightly-Coupled Memory (TCM) is random access memory (RAM) with guaranteed single-cycle response time. TCM is designed for both instruction and data sections of the code which require

maximum throughput. TCM is implemented as dual-port memory with independent access from Instruction and Data memory interfaces (I/F).

Instruction memory I/F does always read TCM as 32-bit words (read only access). Data memory I/F supports 8/16/32 bits wide access to TCM (read/write access). TCM size is up to 64 k Bytes. TCM base address is 0x00480000.[15]

3.4.4 Instruction execution phases

SCR1 has simple in-order pipeline. Functional phases of instruction execution are listed below.

Stage 1:

- Instruction request
- Instruction receive

Stage 2:

- Instruction decode

Stage 3:

- Operand fetch
- Arithmetical and logical operations
- Load/store operations
- Instruction flow control
- Writeback

1. **Instruction request**

   In this phase CPU requests instruction words from Instruction Memory using the address contained in the IMEM_ADDR register. The phase can take arbitrary number of cycles depending on the memory latency. Fetching an instruction word from TCM always takes one clock cycle. In the SCR1 this phase is implemented in Instruction Fetch Unit (IFU).

2. **Instruction receive**

   In this phase the instruction words received from the Instruction Memory are placed into the instruction fetch queue, or, in case of queue bypass (SCR1_NO_DEC_STAGE parameter is defined), are passed directly to the instruction decode unit. Instruction receive unit is responsible for assembling the instruction from parts in case when more than one memory access is needed to fetch that instruction. In SCR1 this phase is implemented in Instruction

Fetch Unit (IFU).

3. **Instruction decode**

   Instruction is decoded and provided to the execution unit as a set of control signals and immediate operand. All decoded signals are placed into the queue, or, in case of a queue bypass (SCR1_NO_EXE_STAGE parameter is defined), are passed directly to the execution logic. In SCR1 this phase is implemented in Instruction Decode Unit (IDU).

4. **Operand fetch**

   The required operands are fetched from the registers (GPRs or CSRs) or from the immediate field of the instruction buffer. This phase is required only for instructions which have operands. In SCR1 operand fetch is always a part of the execution stage and is implemented in Execution Unit (EXU), which requests data from MPRF or CSRF.

5. **Arithmetical and logical operations**

   This covers arithmetical and logical operations with integer values,. This phase is required only for instructions which need the results of arithmetical and logical operations. In SCR1 this phase is always a part of execution stage and implemented in Arithmetic Logic Unit (ALU).

6. **Load/store operations**

   In this phase all operations with Data Memory are executed. This phase is required only for instructions which perform load/store operations. The phase can take arbitrary number of cycles depending on memory latency (no timeout is implemented). Loads and stores data from/to TCM always take two clock cycles. In SCR1 this phase is always a part of the execution stage and implemented in Load-Store Unit (LSU).

7. **Instruction flow control**

   During the normal program flow the next instruction address (PC) is PC+4 for regular instructions or PC+2 for RVC instructions. Below is the list of instructions and events that can alter normal program flow or cause the transition to another state:

   - Jump instruction
   - Taken branch instruction
   - Instruction fence
   - Wait for interrupt instruction

- Exception
- Interrupt
- MRET instruction
- Debug mode redirection event

Detection of such cases is function of the instruction flow control phase as well as calculation of all required control signals, the next PC value and the next status of CPU. MRET instruction and change of MSTATUS or MIE CSRs always takes two clock cycles. In SCR1 this phase is always part of execution stage and implemented in Execution Unit (EXU).

8. **Writeback**

Writeback is a moment, when GPRs, CSRs and PC registers are updated with new values, calculated in previous phases. After this point, the instruction is considered completed. In SCR1 this phase update of GPRs and CSRs is implemented in MPRF, CSRF and update of PC is implemented in the EXU.

3.4.5 Pipeline configurations

Digital core is implemented with 3-stage pipelining. Three-stage pipelining is a technique used to improve performance by breaking down the execution of instructions into three stages: fetch, decode, and execute. Each stage is implemented in a separate pipeline stage, allowing the processor to overlap the execution of multiple instructions and improve throughput. A simple 3 –stage pipelining is shown in the Figure 4.6

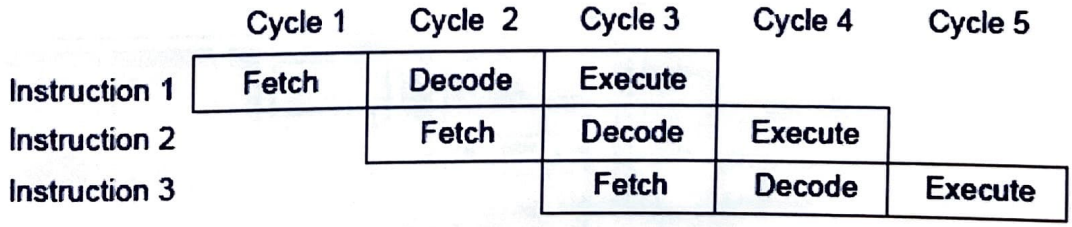

Figure 3.6 Simple 3-stage pipelining

3-stage pipeline configuration includes the following stages:

1. **Instruction request stage:**

In this stage, the processor retrieves the instruction from memory and loads it into an instruction register. The program counter (PC) is also updated to point to the next instruction.

2. Instruction receive and decode stage

In this stage, the processor decodes the instruction and determines the operation to be performed. This involves identifying the instruction type, the operands involved, and any addressing modes that may be used. Once the instruction has been decoded, the processor can prepare to execute it. This may involve setting up the necessary data paths, loading data into registers, or fetching additional instructions from memory. The goal of the instruction receives and decode stage is to prepare the processor to execute.

3. Execution stage

In this stage, the processor performs the operation specified by the instruction. This may involve arithmetic or logical operations, memory accesses, or control flow operations such as branching.

By breaking down the execution of instructions into these three stages, the processor can fetch, decode, and execute multiple instructions simultaneously, improving throughput and overall performance. However, pipelining can also introduce new challenges, such as pipeline hazards (e.g. when an instruction depends on the result of a previous instruction that has not yet completed), which must be managed through techniques such as forwarding or stalling the pipeline.

Overall, three-stage pipelining is a common technique used in modern processor designs to improve performance and efficiency. Figure 4.7 shows 3-stage Pipelining of the digital core.

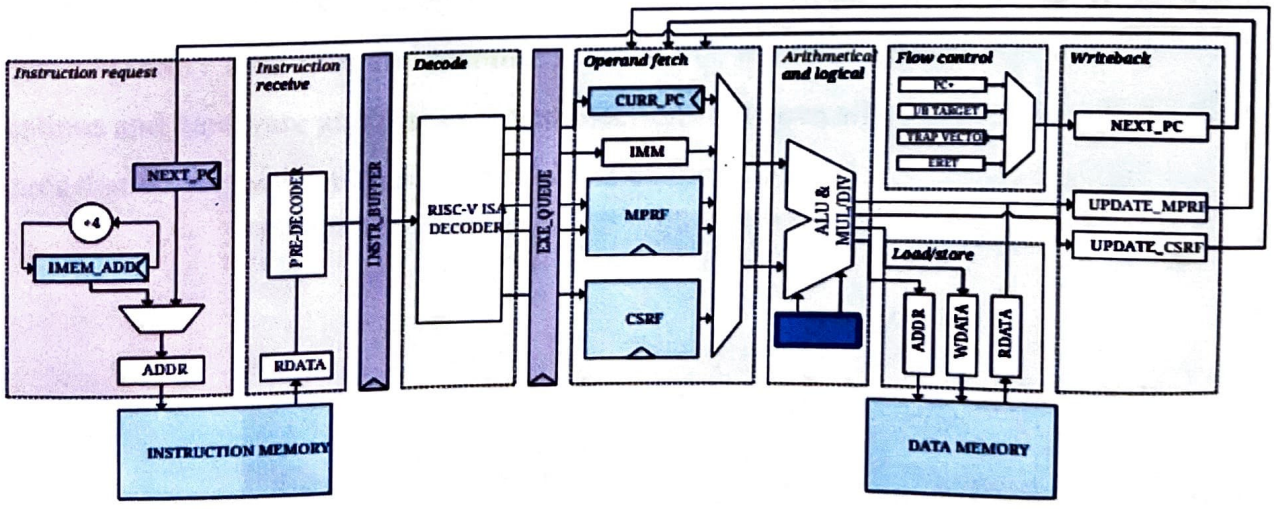

Figure 3.7- 3-stage Pipelining[15]

3.5 Hardware Requirements

Hardware required for programming the Digital core consists of a personal computer or desktop. However, it is required for the system to comprise of certain specifications which include: 16GB RAM or more. Figure 4.8 shows the image of Desktop.

Figure 3.8 –Desktop with minimum 16GB RAM

3.6 Software Requirements

The software used for the implementation of the project are the CADENCE Tools. Various parts of the projects use different tools as mentioned below.

3.6.1 Linux

Linux® is an open source operating system (OS). An operating system is the software that directly manages a system's hardware and resources, like CPU, memory, and storage. The OS sits between applications and hardware and makes the connections between all of your software and the physical resources that do the work. Figure 4.9 shows an example of desktop of Linux terminal.

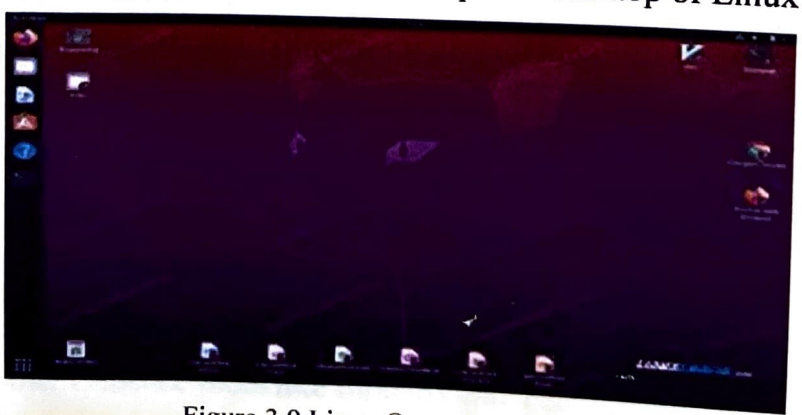

Figure 3.9 Linux Operating

3.6.2 Cadence

Cadence Design Systems is a leading electronic design automation (EDA) software and engineering services company. Cadence offers a broad portfolio of tools and services that enable the design and verification of complex integrated circuits and systems-on-chip (SoCs). Figure 4.10 shows the logo of Cadence.

Figure 3.10 Cadence logo

3.6.3 Genus

Synthesis Solution incorporates a multi-level massively parallel architecture that delivers up to 5X faster synthesis turnaround times and scales linearly beyond 10M instances. In addition, the tool's new physically aware context-generation capability can reduce iterations between unit- and chip-level synthesis by 2X or more. This powerful combination enables up to 10X improvement in RTL design productivity.Genus uses a combination of advanced algorithms and machine learning techniques to improve the synthesis process, resulting in faster time-to-market and better power and performance. It is capable of handling large designs with millions of instances and supports advanced features like mixed-signal synthesis, automatic timing constraints generation, and hierarchical design management.

Some of the key features of Genus include:

1. High-quality synthesis for complex, high-performance designs.
2. Automatic logic restructuring and optimization.
3. Built-in design constraints and validation.
4. Support for multiple design languages including Verilog, VHDL, and SystemVerilog.
5. Advanced clock tree synthesis and optimization.
6. Hierarchical design management and support for incremental synthesis.
7. Integration with other Cadence tools like Innovus and Tempus for seamless implementation.

Overall, Genus is a powerful synthesis tool that helps designers optimize their digital circuits for power, performance, and area. It enables faster design closure and improved design quality, making it an essential tool for digital circuit design.

3.6.4 Innovus

The Cadence Innovus Implementation System is optimized for the most challenging designs, as well as the latest FinFET 16nm, 14nm, 7nm, 5nm, and 3nm process nodes, helping get an earlier design start with a faster ramp-up. With unique new capabilities in placement, optimization, routing, and clocking, the Innovus system features an architecture that accounts for upstream and downstream steps and effects in the design flow. This architecture minimizes design iterations and provides the runtime boost needed to get to market faster. Using the Innovus system, it's possible to be equipped to build integrated, differentiated systems with less risk. The Innovus system features a variety of key capabilities. Its massively parallel architecture can handle large designs and take advantage of multi-threading on multi-core workstations, as well as distributed processing over networks of computers.

Innovus is a powerful implementation tool that helps designers optimize their digital circuits for power, performance, and area. It enables faster design closure and improved design quality, making it an essential tool for digital circuit implementation

CHAPTER 4

PHYSICAL DESIGN VERIFICATION AND RESULTS

4.1. Design Flow

Figure 5.1 shows the steps involved in the implementation of the Digital Core using the SUTRA methodology.

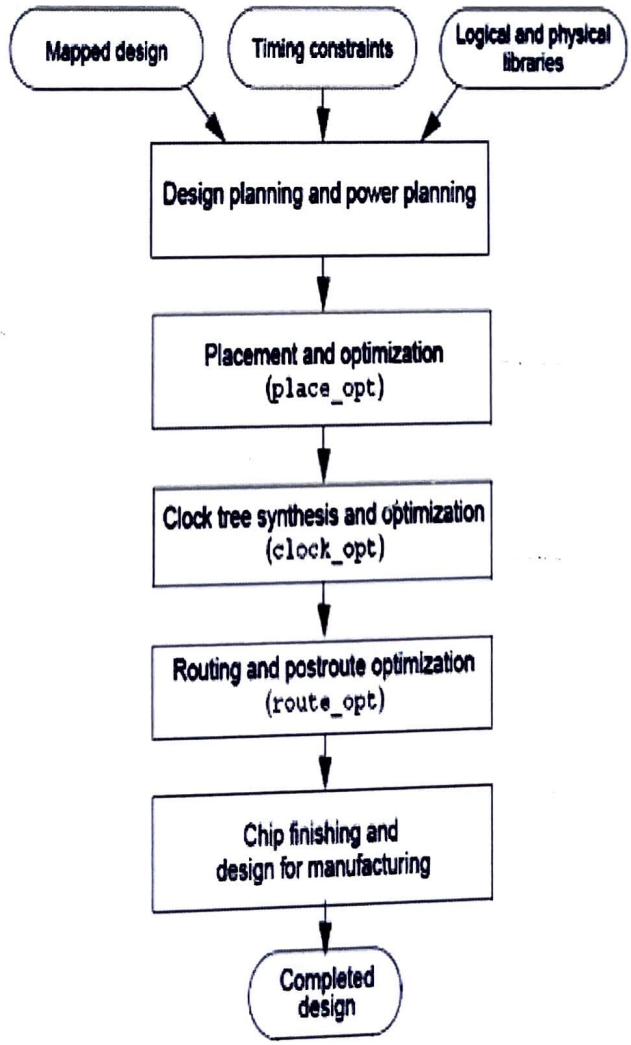

Figure 4.1 VLSI Physical Design Flow [17]

4.2 Results

This section presents the results obtained after the steps mentioned in chapter 5.1 is performed. Along with the images, power consumption, area, etc. is listed.

The project was performed in 2 stages:

1. ASIC flow of Digital core with 8 SRAM blocks.
2. ASIC flow of Digital core integrated with UART and I2C.

4.2.1 ASIC flow of Digital core with 8 SRAM blocks

1. Synthesis.

In Cadence, synthesis is the process of converting the RTL (Register Transfer Level) description of a digital circuit into a gate-level netlist. During synthesis, Cadence generates several reports that provide information about the design and the synthesis process. Here are some of the reports that are typically generated after synthesis in Cadence:

- **Synthesis log**: The synthesis log is a report that provides information about the synthesis process, such as the input files, the synthesis options used, and any errors or warnings that occurred during synthesis.
- **Design summary**: The design summary report provides an overview of the design, including the number of cells, the number of nets, the area of the design, and the number of timing violations.
- **Timing report:** The timing report provides information about the timing of the design, including the critical path, the setup and hold times, and any timing violations that occurred during synthesis.

- **Power report**: The power report provides information about the power consumption of the design, including the dynamic power, the static power, and the total power.
- **Area report**: The area report provides information about the area of the design, including the total area, the cell area, and the net area.
- **Gate-level netlist:** The gate-level netlist is the output of the synthesis process. It is a description of the digital circuit in terms of gates and their interconnections.

The reports generated after synthesis provide designers with valuable information about the design and the synthesis process. The reports were generated using the command >*make synth* Figure 5.8 shows the physical reports of after synthesis. The total number of logical instances is 4521.The area of the Digital block is 65594547 μm².Figure 5.9 shows the power reports of the synthesis. The total clock power is 24.57 mW. The leakage and switching power respectively 0.27 mW and 1.93 mW.

Physical

Snapshot	Density (%)	Logical instances	Logical area (um^2)	Total instances	Total area (um^2)	Blocked area
route	0.25	4,521	6,559,547	4,52	6,559,547	0

Instances Detail

Snapshot	Standard Cell							Total (SC)	Other				MSV				Total
		register	icg	latch	buffer	inverter	combo	total	macro	physical	io	blackbox	power_switch	isolation	level_shifter	always_on	total
route		210	15	0	448	1,100	2,740	4,513	8	0	0	0	0	0	0	0	4,521

Area Detail

Snapshot	Standard Cell							Total (SC)	Other			
		register (um^2)	icg (um^2)	latch (um^2)	buffer (um^2)	inverter (um^2)	combo (um^2)	total (um^2)	macro (um^2)	physical (um^2)	io (um^2)	blackbox (um^2)
route		1,704	113	0	1,269	1,735	6,541	11,361	6,548,186	0	0	0

Multibit Detail

Snapshot	Multibit		
	avg_width	count	coverage (%)
route	1.00	0	0.00

Figure 4.8 Synthesis physical reports of Stage 1

Power

Snapshot	Whole Design				Total Clock Power (mW)
	Total (mW)	Leakage (mW)	Internal (mW)	Switching (mW)	
route	24.57	0.27	22.37	1.93	24.57

Leakage Details

Snapshot	Leakage Power Breakdown					
	clock_combinational (mW)	clock_sequential (mW)	combinational (mW)	io (mW)	macro (mW)	sequential (mW)
route	0.00	0.00	0.00	0.00	0.27	0.00

Internal Details

Snapshot	Internal Power Breakdown					
	clock_combinational (mW)	clock_sequential (mW)	combinational (mW)	io (mW)	macro (mW)	sequential (mW)
route	0.14	0.02	0.43	0.00	21.64	0.14

Switching Details

Snapshot	Switching Power Breakdown					
	clock_combinational (mW)	clock_sequential (mW)	combinational (mW)	io (mW)	macro (mW)	sequential (mW)
route	0.12	0.01	1.54	0.00	0.23	0.02

Figure 4.9 Synthesis power reports of stage 1

2. Floorplanning

The >*make fp* command is used to generate the floorplanning of the design. Floorplannig is done in Innovus . Innovus GUI(Graphical User Interface) terminal is opened by the command > *make iterminal* given by the automated flow SUTRA a methodology designed in BNMIT

The command>*read_dbpnr/FloorplanCompleted.inn* gives the floorplan of the design. Figure 5.10 shows the floorplan of the core.

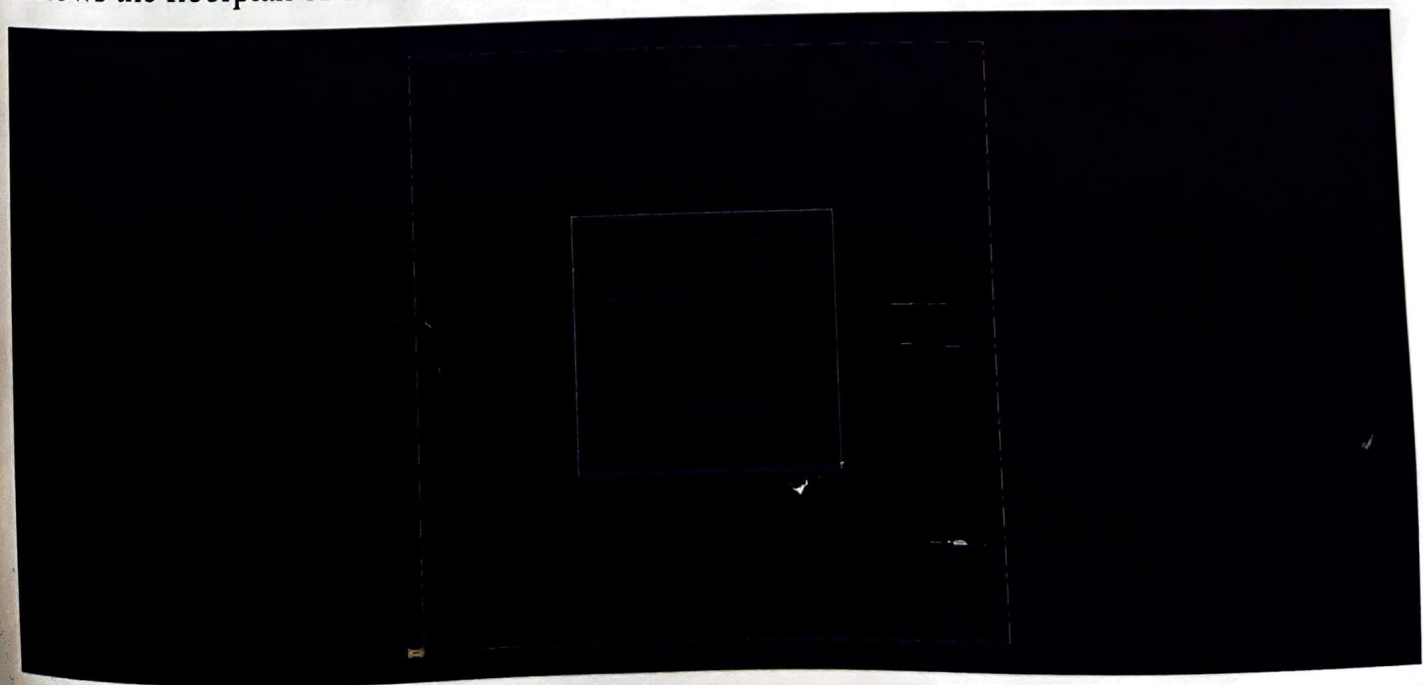

Figure 4.10 Floorplan of stage 1

3. Power mesh

Power mesh is created for power supply. The command >*make pmesh* gives the pmesh of the design. The command >*read_db pnr/powermesh.inn* gives the floorplan of the design.in innovus terminal. Figure 5.11 shows the power mesh of the design.

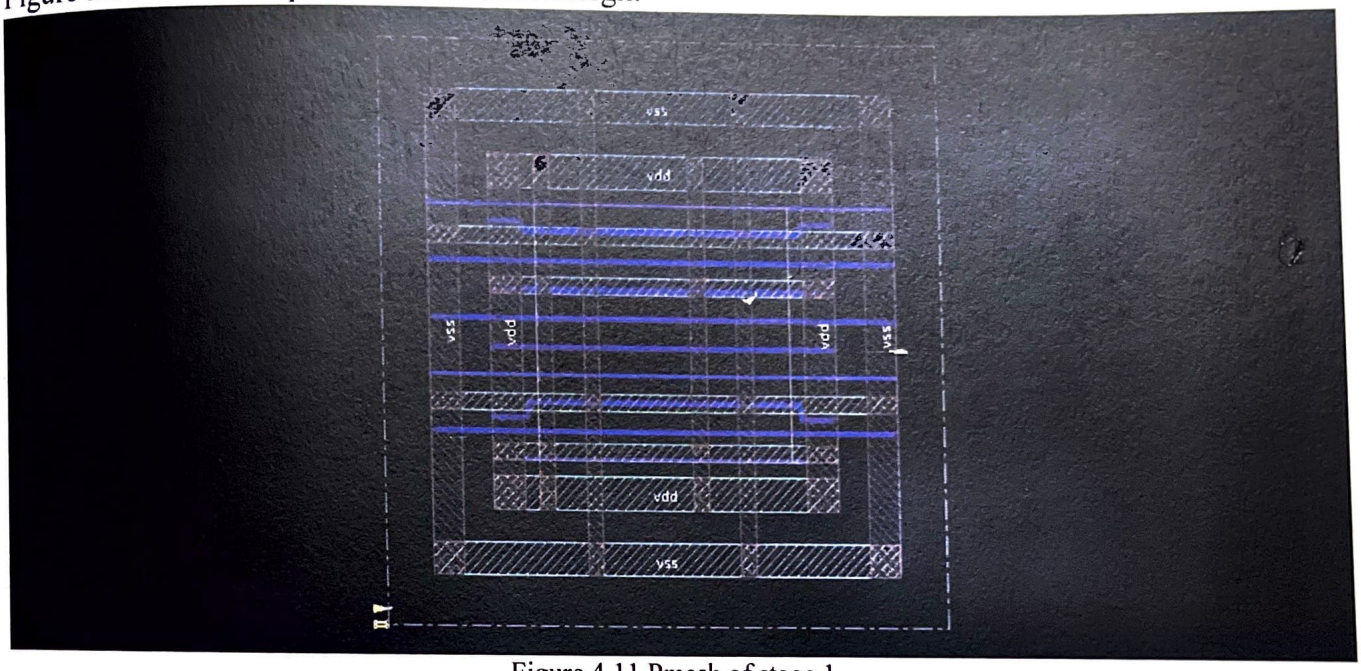

Figure 4.11 Pmesh of stage 1

4. Placement.

The command >*make place* does the placement of the design. The command >*read_db pnr/place.inn* gives the floorplan of the design.in innovus terminal. Figure 5.12 shows the placement of the design.

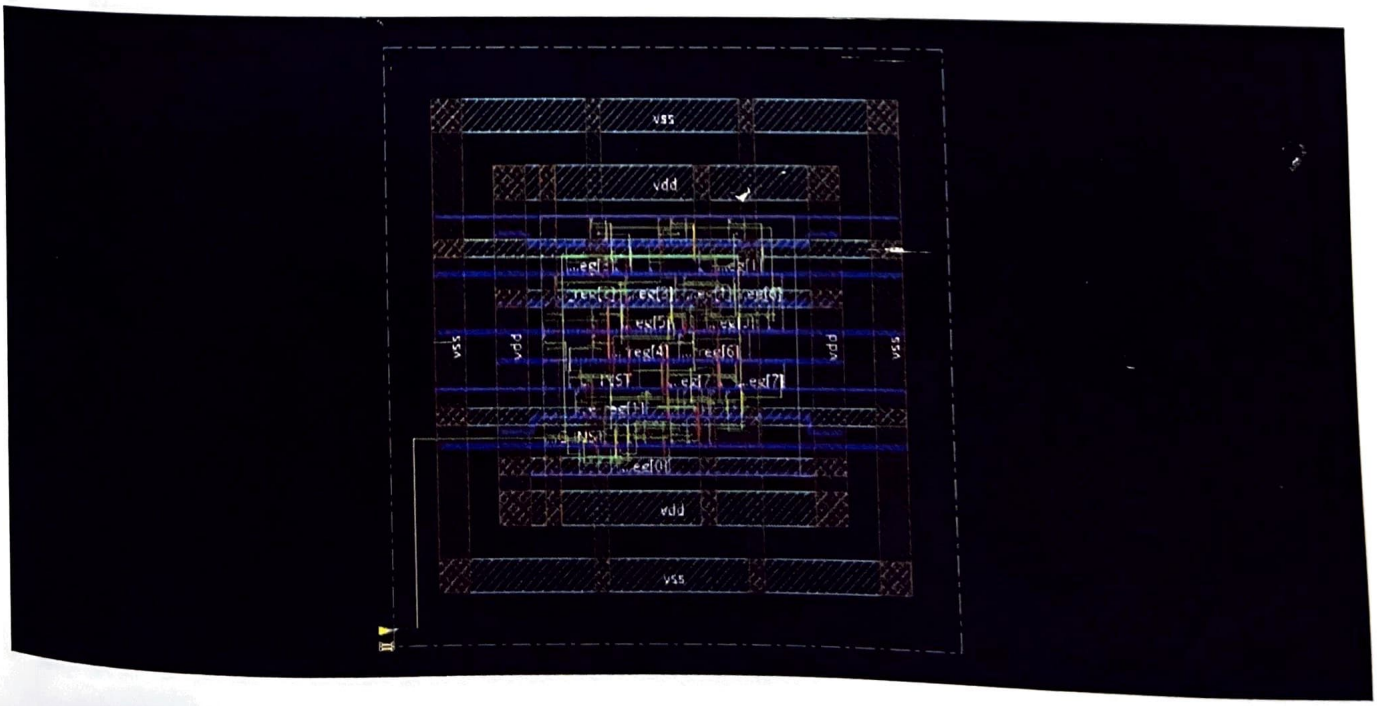

Figure 4.12 Placement of stage 1

5. SRAM blocks

Figure 5.13 shows the SRAM Blocks which is to be placed in the Digital core such that the core utilization is > 60%.

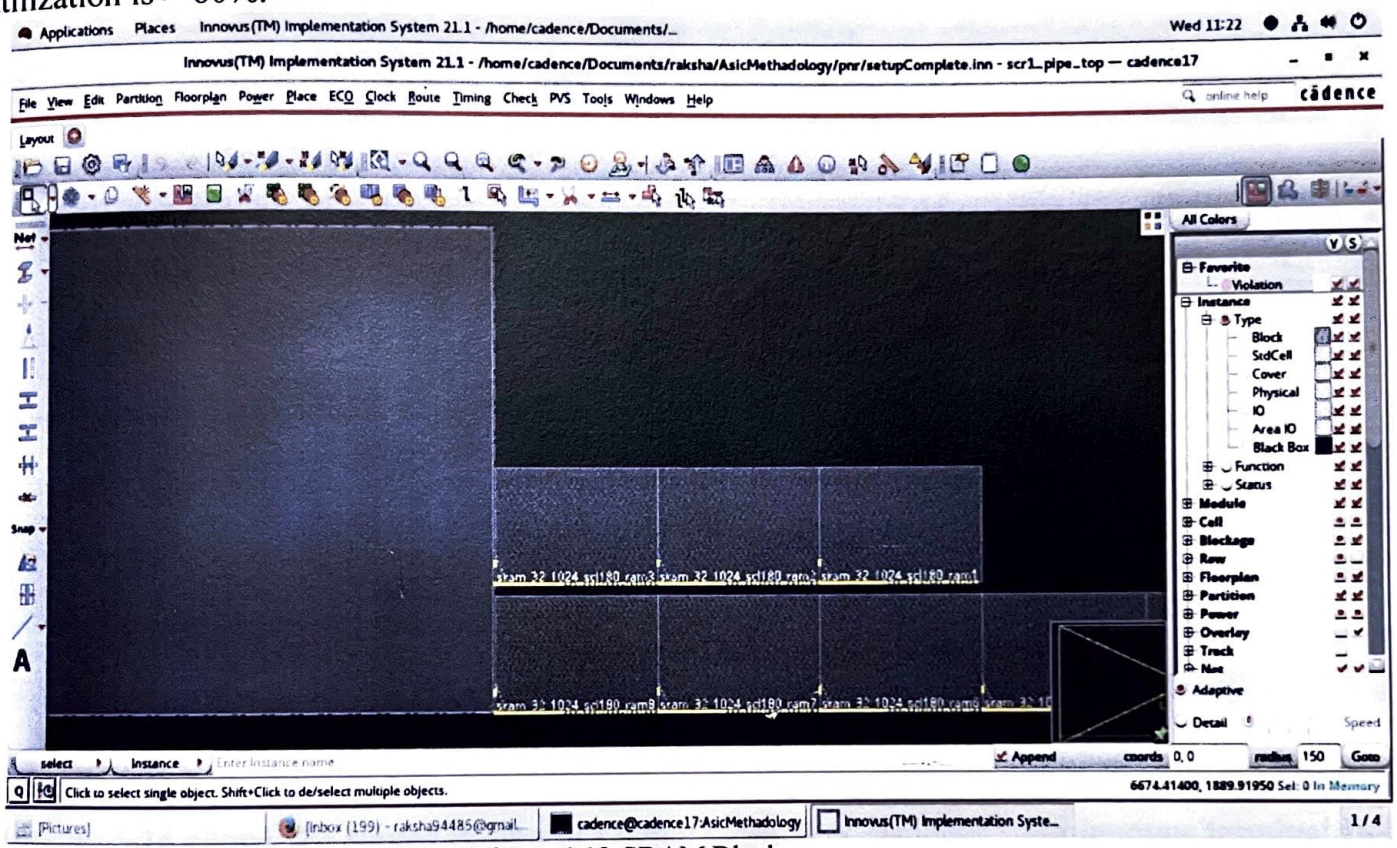

Figure 4.13-SRAM Block

6. SRAM blocks after placement

Placement of SRAM blocks is done in Innovus GUI terminal.Which can be invoked by the command >*make iterminal.* Figure 5.14 shows the placement of the SRAM block which is done manually such that, the core utilization is less than 60%.

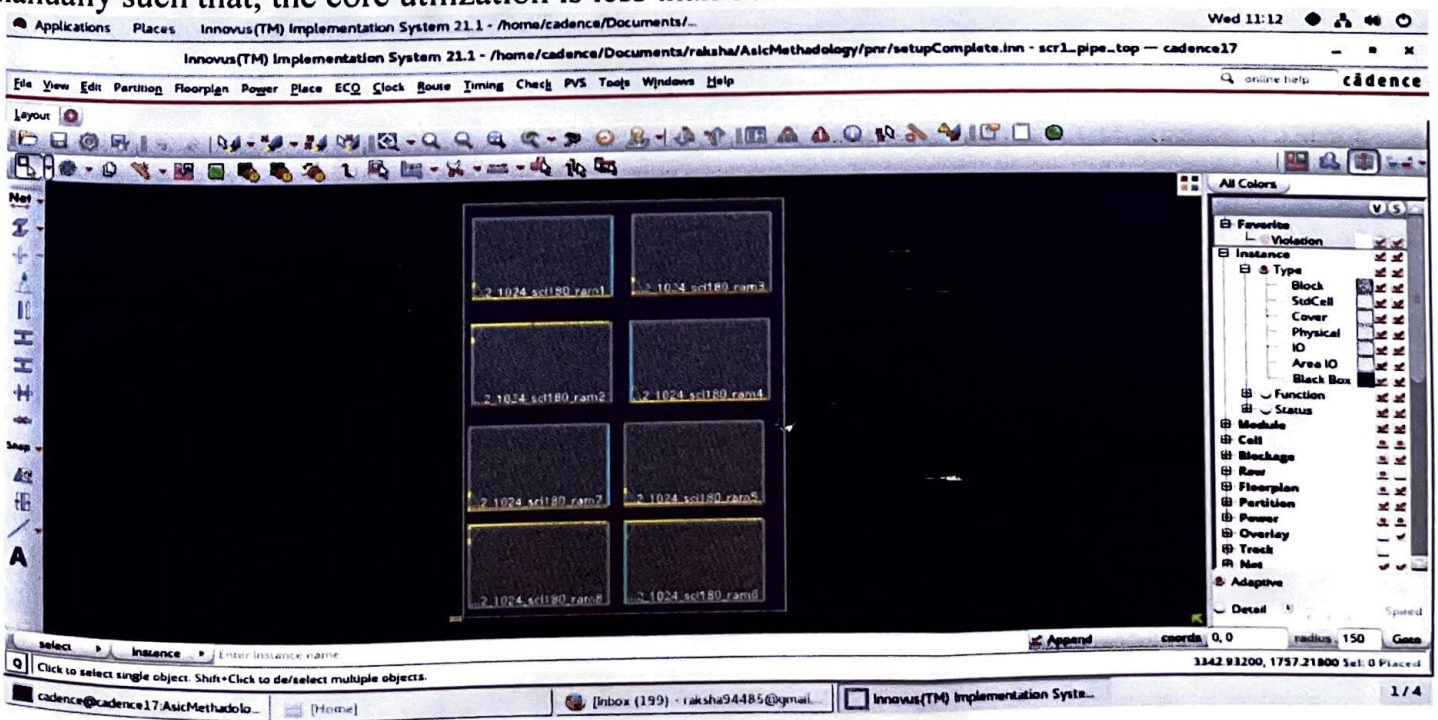

Figure 4.14- SRAM blocks after placement

7. Power mesh creation

Figure 5.15 shows the power mesh created around the SRAM blocks. The power mesh on the SRAM blocks is chipped off manually.

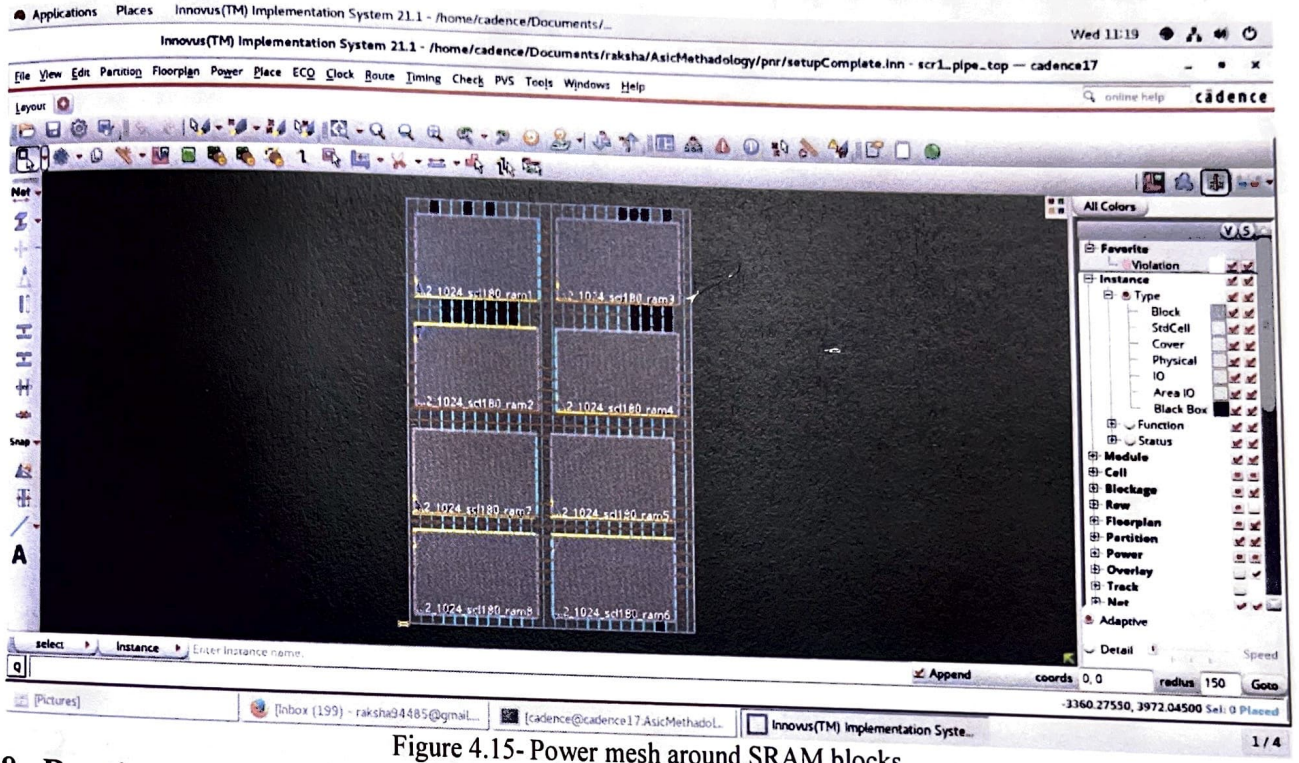

Figure 4.15- Power mesh around SRAM blocks.

8. Routing

Figure 5.16 shows the routed design of Digital Core .It is obtained from innovus terminal from the command >read_db pnr/routeCompleted.inn. Figure 5.17 shows Routed Digital Core in html page.

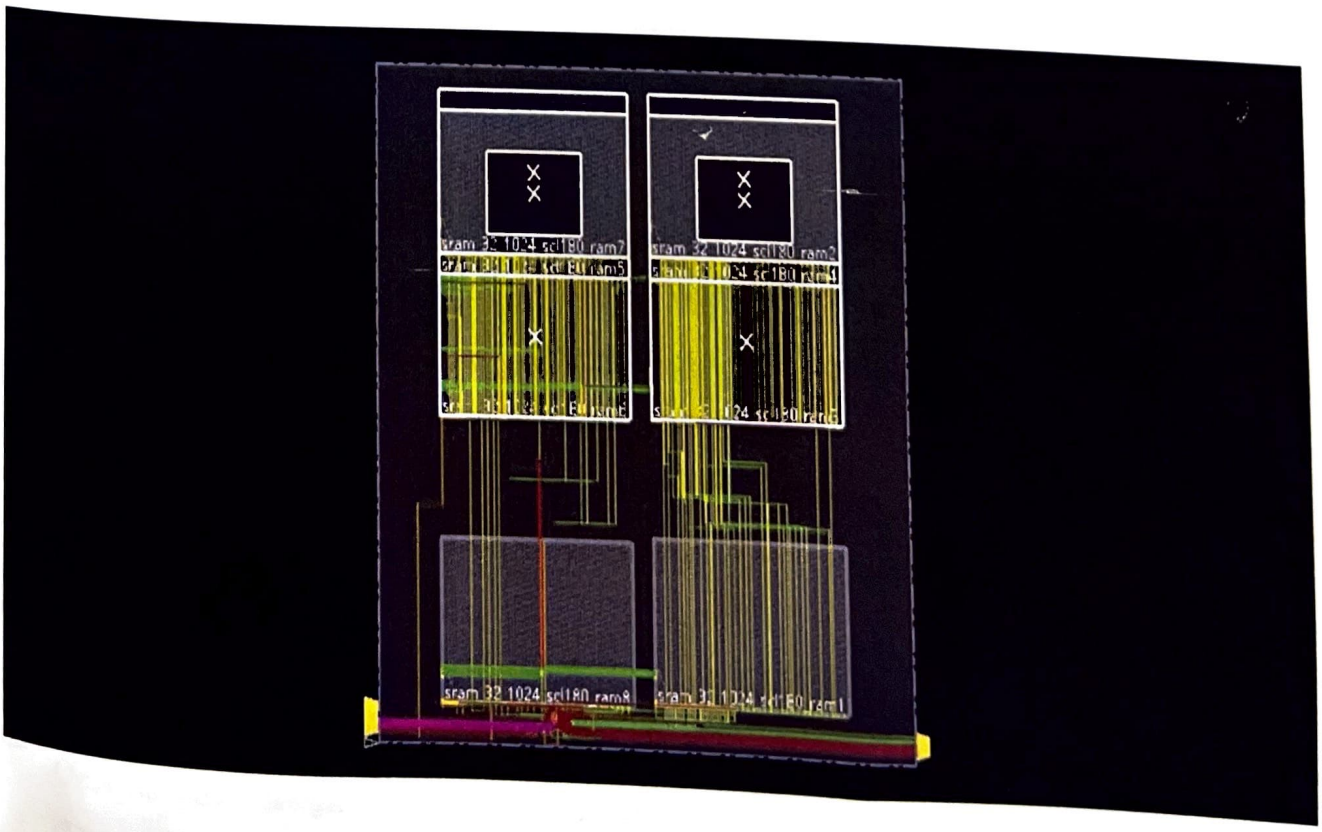

Figure 4.16- Routed Digital Core

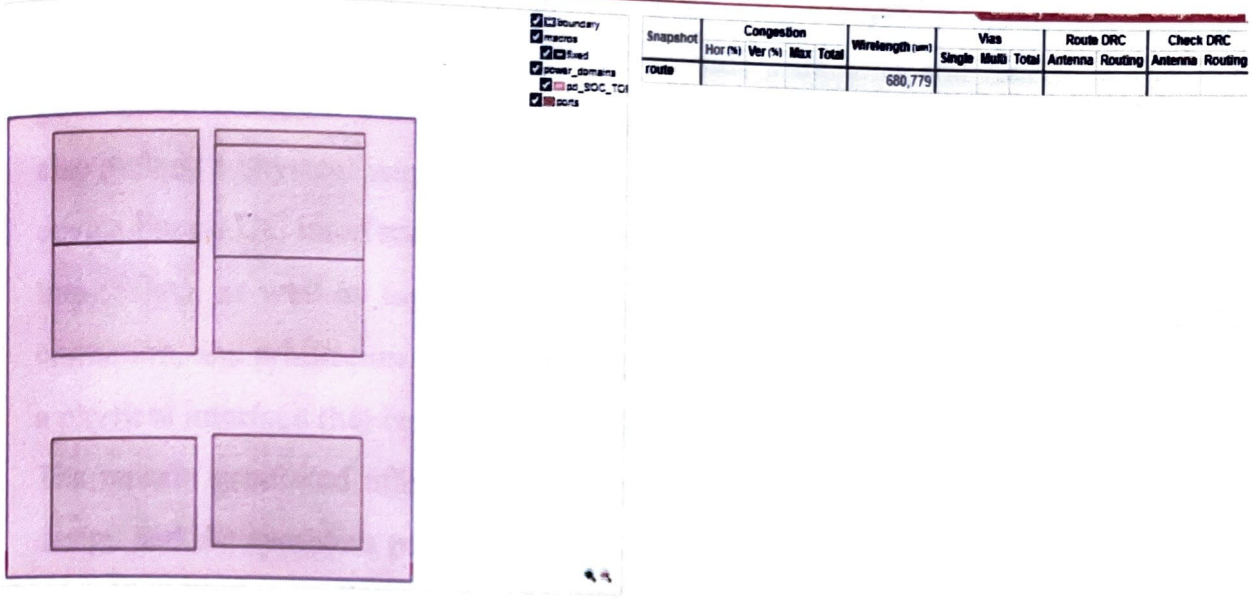

Figure 4.17- Routed Digital Core in html

This completes the Stage 1 of the project.

4.2.2. ASIC flow of Digital core integrated with UART and I2C.

. Lib and .lef files of UART and I2C was taken from the other team and was integrated with the core. All the steps of Chapter 5.2.1 were performed and the results were obtained.

1. Synthesis

The synthesis of the SCR1 microcontroller with UART and I2C interfaces involves converting the RTL (Register Transfer Level) description of the digital circuit into a gate-level netlist that includes the UART and I2C interfaces.

The synthesis process involves converting the RTL design into a gate-level netlist that includes the UART and I2C interfaces. This process involves several steps, including technology mapping, optimization, and timing analysis. During this process, the synthesis tool generates reports that provide information about the design and the synthesis process.

The synthesis of the SCR1 microcontroller with UART and I2C interfaces involves a series of steps that involve the use of specialized synthesis tools and techniques. The synthesis process is critical to the success of the design, as it ensures that the RTL design is converted into an optimized and efficient gate-level netlist that meets the requirements of the application.

The reports of the synthesis were generated using the command >*make synth.*

Once the HDL design is complete, it can be synthesized into a digital circuit using a FPGA (Field Programmable Gate Array) or ASIC (Application-Specific Integrated Circuit) design

tool. The synthesized circuit can then be programmed onto the FPGA or manufactured as an ASIC to create a custom UART or I2C interface for your specific application.

For a UART interface, the circuit would include a transmit and receive buffer, a baud rate generator, and logic to handle data framing, error detection, and flow control. The circuit would also include a physical interface that connects to the UART pins of the microcontroller or other device.For an I2C interface, the circuit would include a serial clock line (SCL) and a serial data line (SDA), as well as logic to handle data transmission and reception, error detection and correction, and arbitration between multiple devices on the bus. The circuit would also include a physical interface that connects to the I2C pins of the microcontroller or other device.

The reports generated after synthesis provide designers with valuable information about the design and the synthesis process. The reports were generated using the command >*make synth*

Figure 5.18 shows the physical reports of after synthesis. The total number of logical instances is 25349.The area of the Digital block is 6998809 μm².Figure 5.19 shows the power reports of the synthesis. The total clock power is 24.57 mW. The leakage and switching power respectively 0.19 mW and 1.60 mW.

Stylus Metrics | runStylus Metrics | run

Physical

Snapshot	Density (%)	Logical instances	Logical area (um^2)	Tota	instances	Total area (um^2)	Blocked area
route	3.22	25,349	6,998,809		25,349	6,998,809	0

Instances Detail

Snapshot	Standard Cell						Total (SC)	Other				MSV				Total
	register	icg	latch	buffer	inverter	combo	total	macro	physical	io	blackbox	power_switch	isolation	level_shifter	always_on	total
route	4,210	15	0	5,148	5,157	10,809	25,339	10	73,801	0	0	0	0	0	0	25,349

Area Detail

Snapshot	Standard Cell						Total (SC)	Other			
	register (um^2)	icg (um^2)	latch (um^2)	buffer (um^2)	inverter (um^2)	combo (um^2)	total (um^2)	macro (um^2)	physical (um^2)	io (um^2)	blackbox (um^2)
route	29,068	111	0	16,491	16,260	39,070	101,000	6,897,810	75,720	0	0

Multibit Detail

Snapshot	Multibit		
	avg_width	count	coverage (%)
route	1.00	0	0.00

Figure 4.18- Physical design II

Snapshot	Whole Design				Total Clock Power (mW)
	Total (mW)	Leakage (mW)	Internal (mW)	Switching (mW)	
route	24.64	0.27	22.76	1.60	24.64

Leakage Details

Snapshot	Leakage Power Breakdown					
	clock_combinational (mW)	clock_sequential (mW)	combinational (mW)	io (mW)	macro (mW)	sequential (mW)
route	0.00	0.00	0.01	0.00	0.27	0.00

Internal Details

Switching Details

Snapshot	Switching Power Breakdown					
	clock_combinational (mW)	clock_sequential (mW)	combinational (mW)	io (mW)	macro (mW)	sequential (mW)
route	0.19	0.01	1.36	0.00	0.03	0.01

Figure 4.19-Power II

2. Floorplanning

Figure 5.20 shows the floorplan after the integration of UART and I2C. The floorplan was obtained using the command >*make fp.* The command > *make iterminal* is used to invoke innovus. The command >*read_db pnr/FloorplanCompleted.inn* is used to obtain the floorplan.

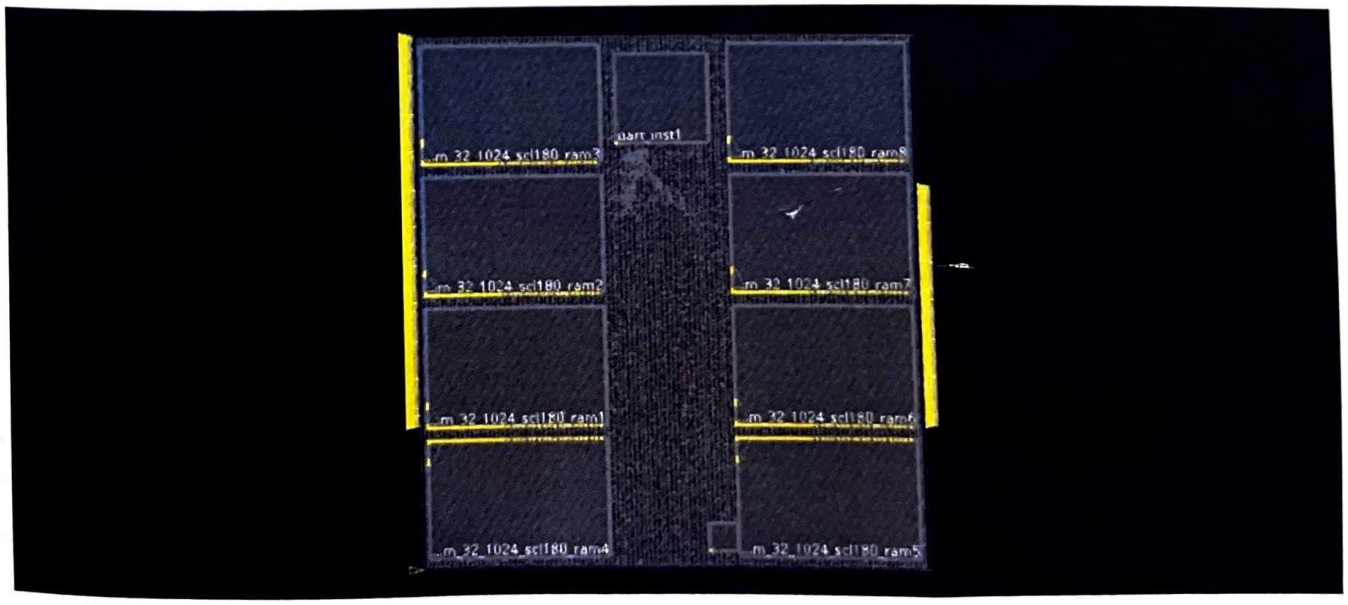

Figure 4.20 Floorplan after integration

3. Power Mesh Creation

Power mesh is created for power supply. The command >*make pmesh* gives the pmesh of the design. The command >*read_db pnr/powermesh.inn* gives the placement of the design.in innovus terminal. Figure 5.21 shows the power mesh of the design after integration.

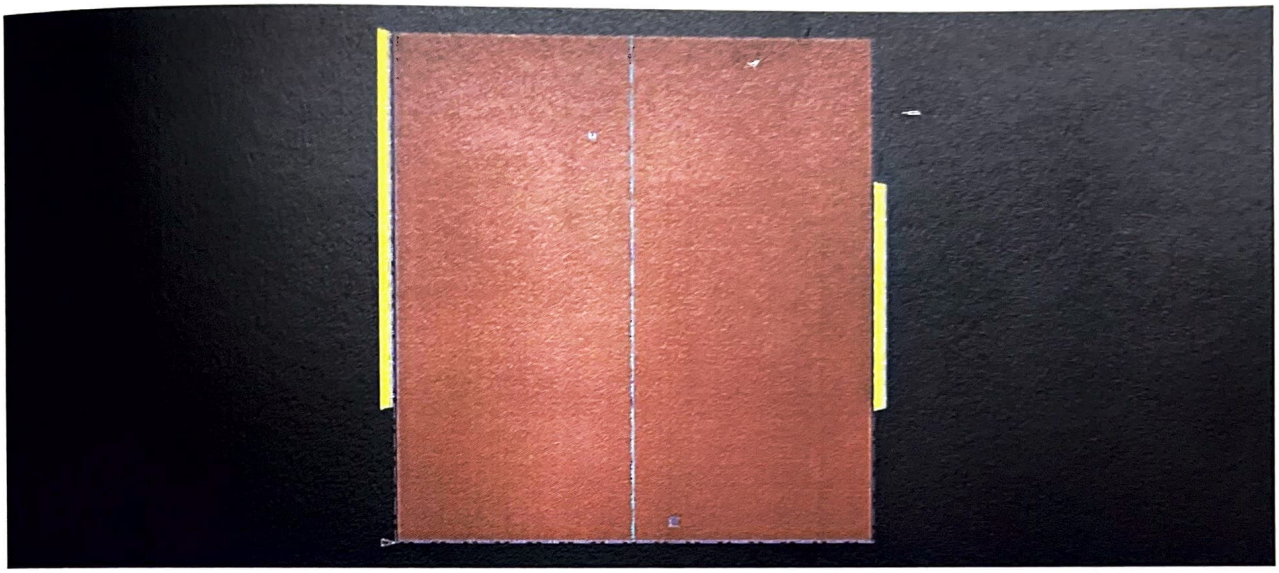

Figure 4.21 Pmesh after integration

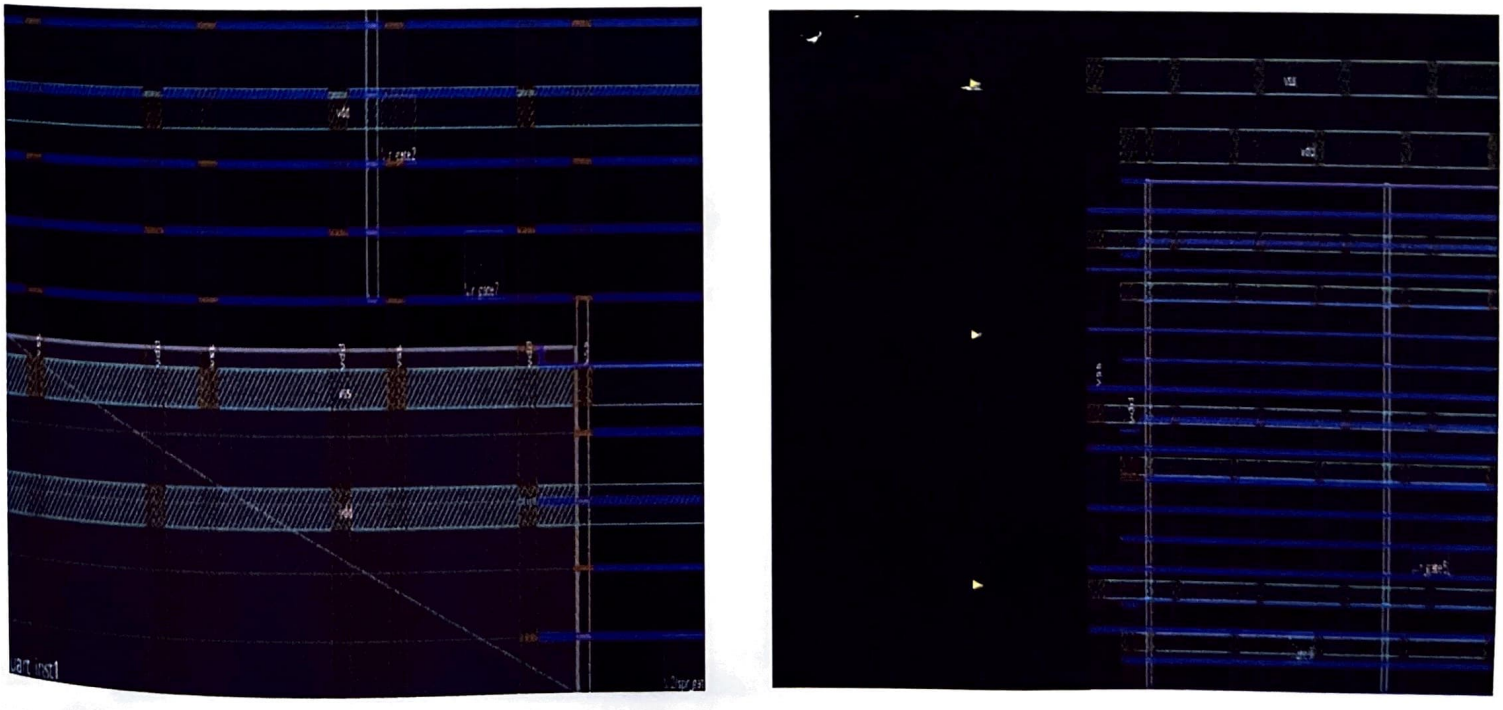

Figure 4.22 Detailed view of Pmesh

4. Placement

The command >*make place* does the placement of the design. The command >*read_db pnr/place.inn* gives the floorplan of the design.in innovus terminal. Figure 5.23 shows the instances of the design. Figure 5.24 shows the placement sof the design.

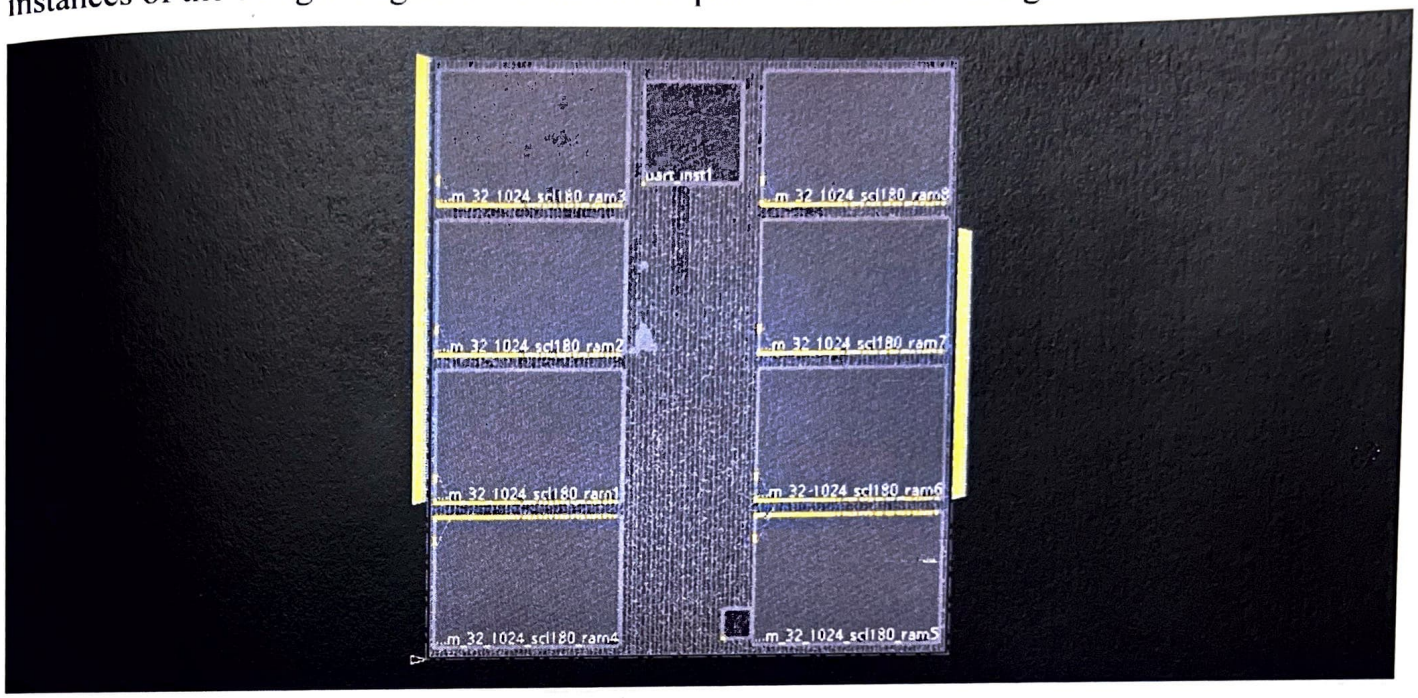

Figure 4.23 Instances

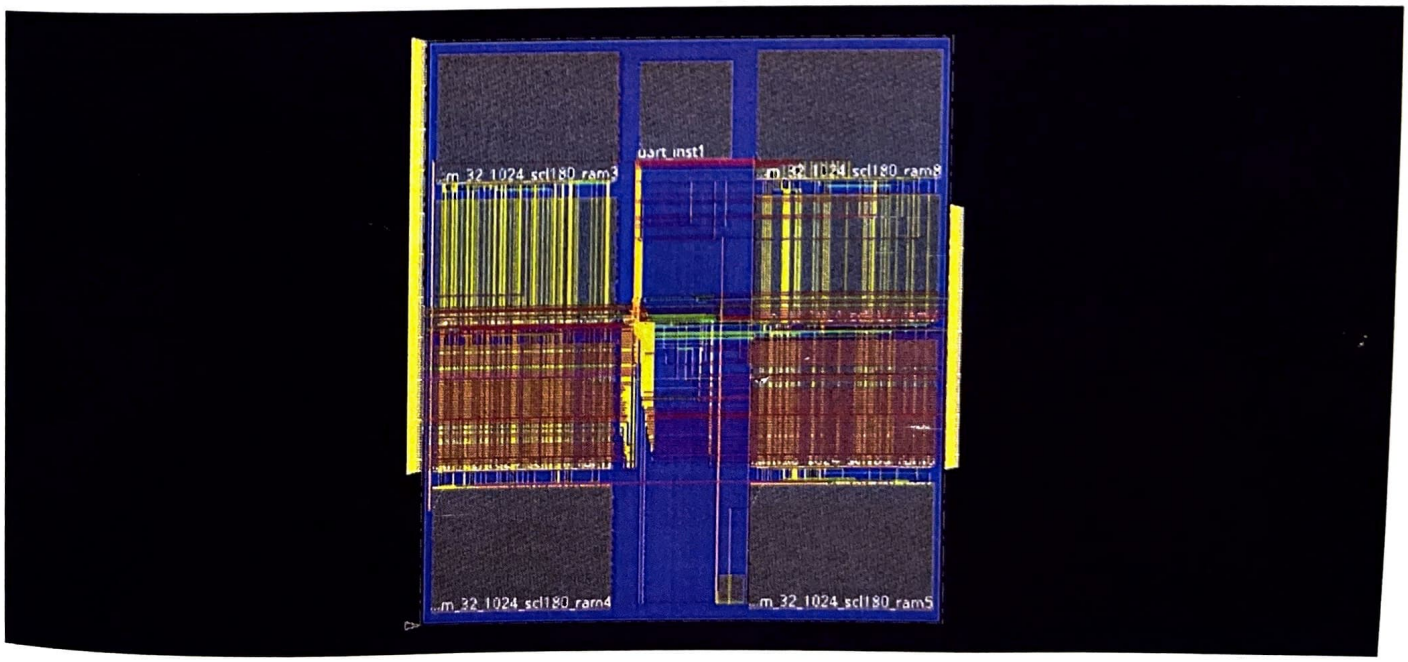

Figure 4.24 Placement after integration

5. CTS

CTS is done to balance the unbalanced clock tree. The command >*make cts* does the CTS of the design. The command >*read_db pnr/postCTS.inn* gives the CTS of the design.in innovus terminal. Figure 5.25 shows the CTS of the design. Figure 5.26 shows the unbalanced clock. Figure 5.27 shows the balanced clock. The sdc constraints for clock is given by create_clock -name {clk} -period 8.000 -waveform { 0.000 4.000 } [list [get_ports {clk}]].

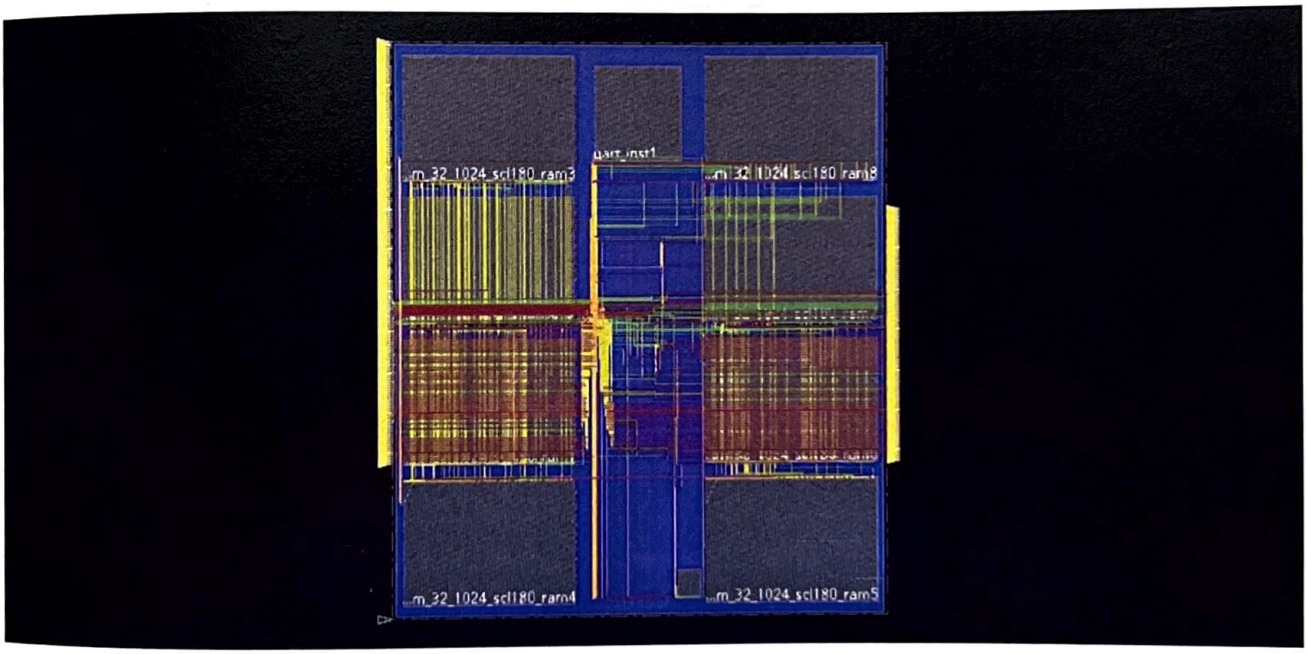

Figure 4.25 CTS after integration

Figure 4.26 Unbalanced clock

5. CTS

CTS is done to balance the unbalanced clock tree. The command *>make cts* does the CTS of the design. The command *>read_db pnr/postCTS.inn* gives the CTS of the design.in innovus terminal. Figure 5.25 shows the CTS of the design. Figure 5.26 shows the unbalanced clock. Figure 5.27 shows the balanced clock. The sdc constraints for clock is given by create_clock -name {clk} -period 8.000 -waveform { 0.000 4.000 } [list [get_ports {clk}]].

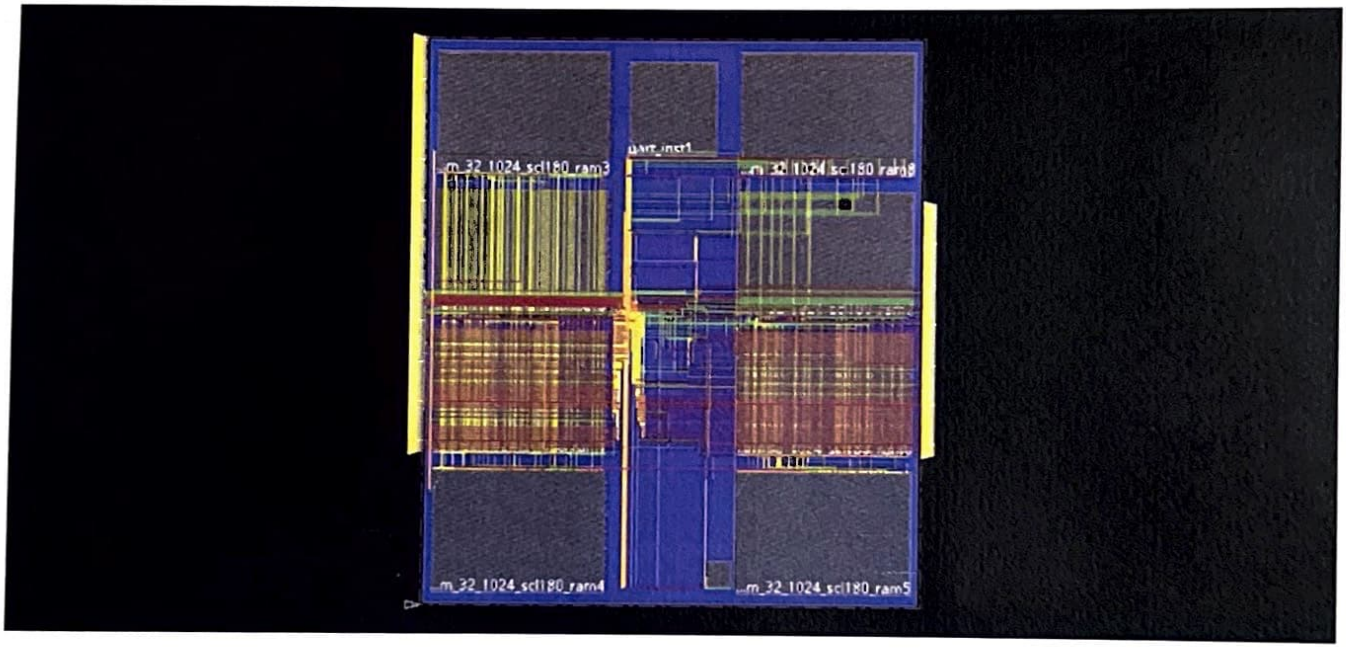

Figure 4.25 CTS after integration

Figure 4.26 Unbalanced clock

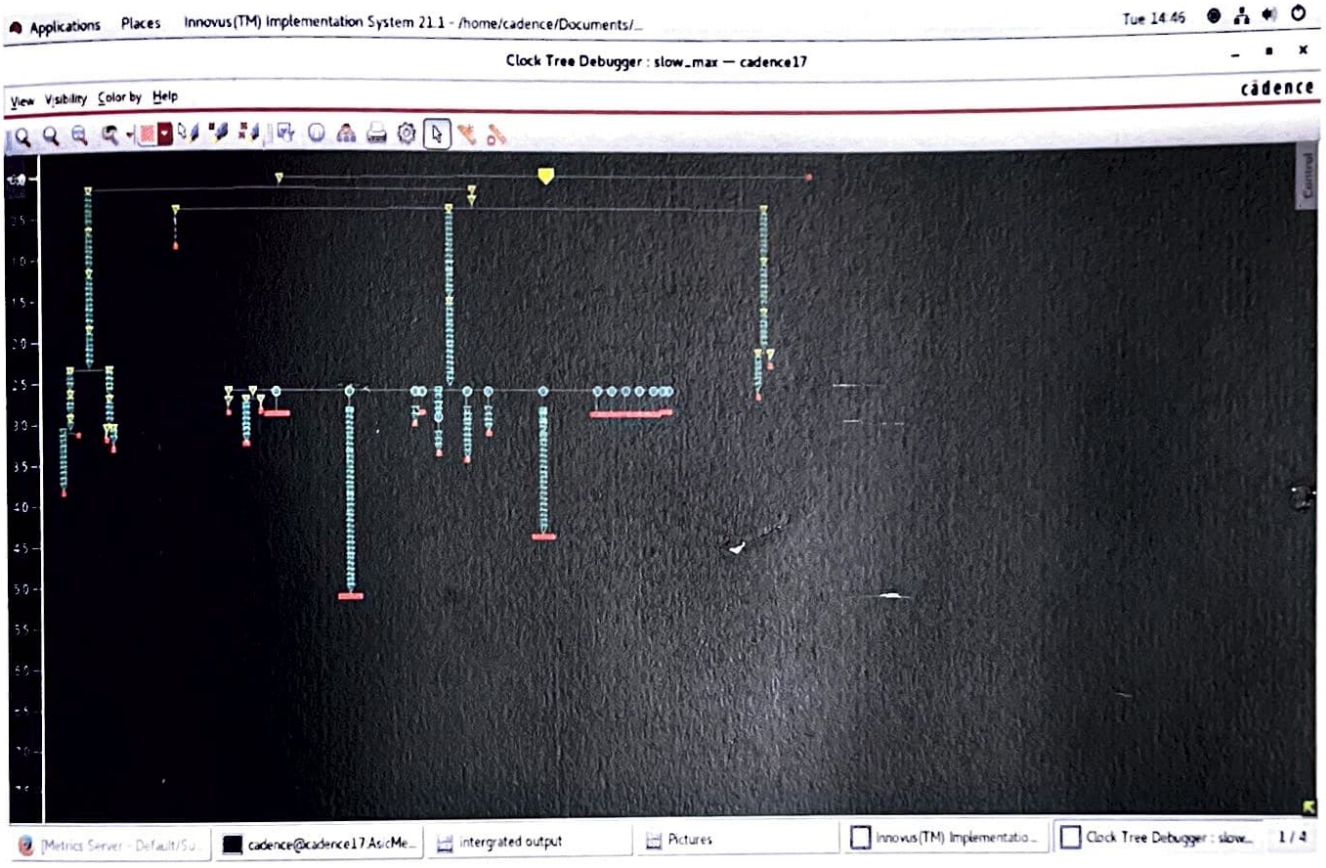

Figure 4.27 Balanced clock

6. Routing

Figure 5.28 shows the routed design of Digital Core .It is obtained from innovus terminal from the command >read_db pnr/routeCompleted.inn.

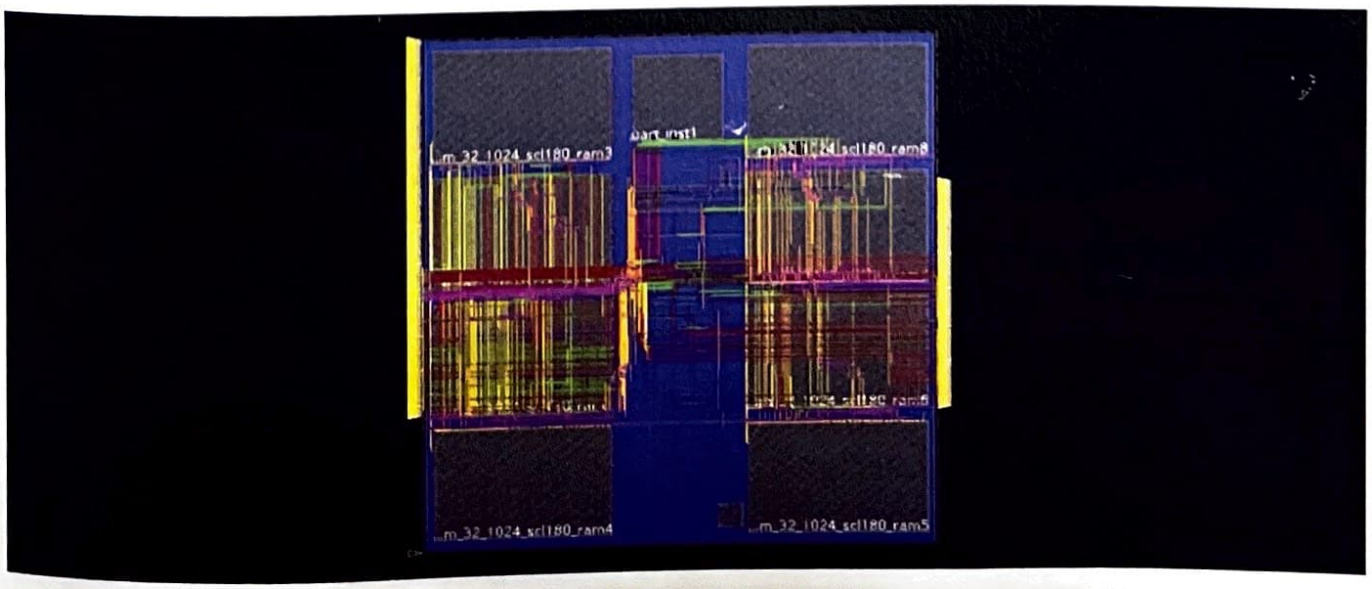

Figure 4.28 Routed Digital Core

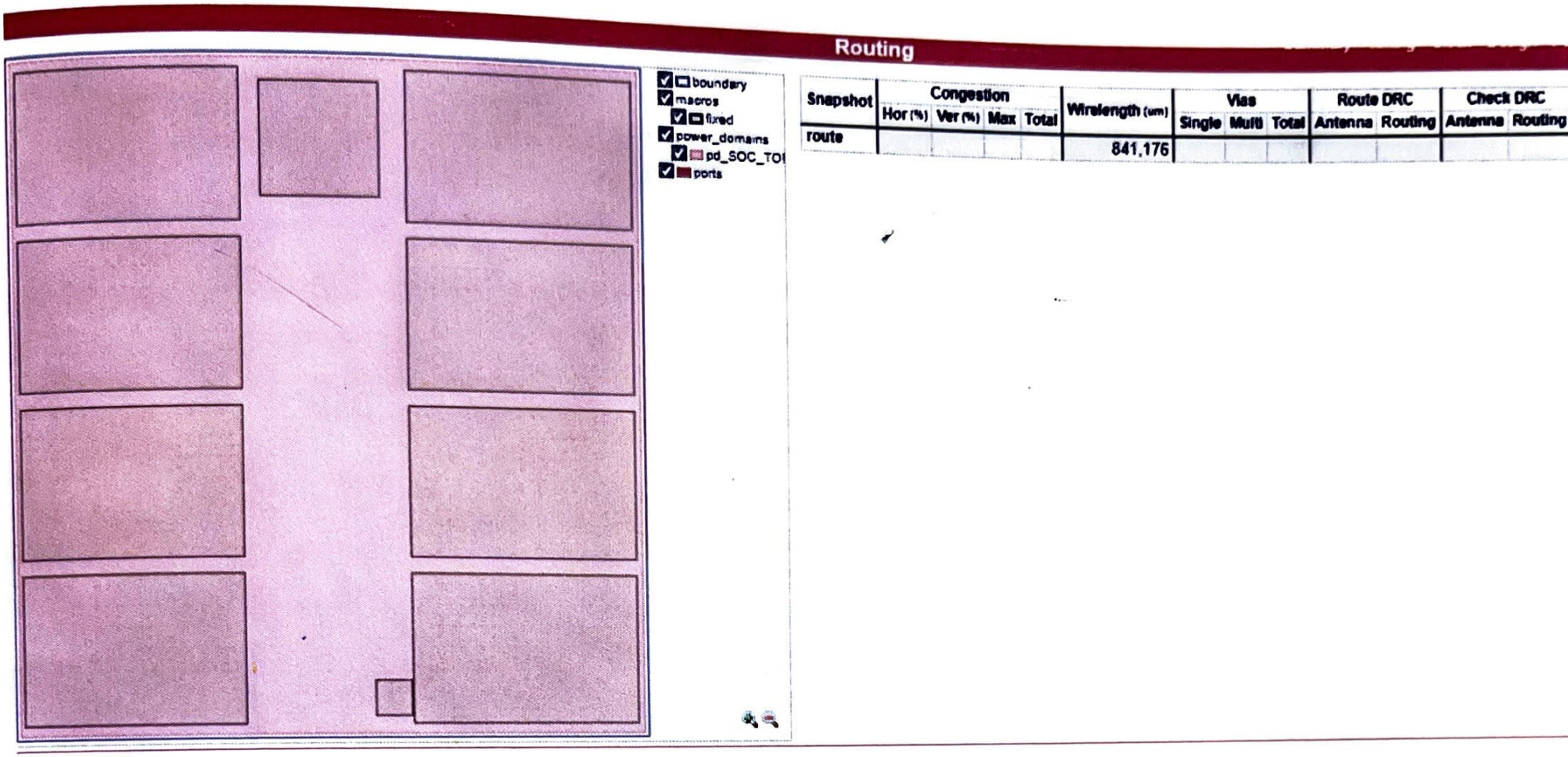

Figure 4.29- Routed Digital Core in html

This completes the Stage 2 of the project.

CHAPTER 5

CONCLUSION

5.1 Summary

This project reports the ASIC synthesis of the Digital core of the processor, describing how the ASIC was designed and obtained results. The Digital core is built is with 3-stage pipeling. The estimates from the physical synthesis indicate that the implemented processor has an area around 6,998,809 um^2, switching and leakage power around 1.60 mW and 1.27 mW, respectively.

The Digital Core is based on the RISC-V ISA, which is an open-source instruction set architecture that is designed to be simple, extensible, and scalable. The Digital Core implements the RV32I instruction set, which includes support for integer, as well as hardware support for compressed instructions. The Digital Core is a 3-stage pipeline processor that is designed to achieve high performance while minimizing power consumption. The pipeline includes separate fetch, decode, and execute stages, with support for branch prediction and data forwarding.The Digital Core includes a range of features that make it well-suited for use in embedded systems and IoT devices, including a small footprint, low power consumption, and support for a range of system-level interfaces and peripherals. The processor also includes a range of security features, such as support for a secure boot process and hardware-based cryptography.

The SCR1 supports a range of system-level interfaces and peripherals, including UART, SPI, I2C, GPIO, and timers. It also includes hardware support for interrupt handling and a memory management unit (MMU) for virtual memory management. In this project the Digital core is integrated with UART and I2C. The SCR1 includes a range of security features, such as support for a secure boot process, hardware-based cryptography, and a memory protection unit (MPU) for enforcing memory access permissions.

Overall, the SCR1 is a versatile and capable RISC-V processor that is well-suited for use in a wide range of applications. Its small footprint, low power consumption, and support for a range of system-level interfaces and security features make it an attractive option for system designers who need a flexible and scalable processor core for their designs.

A possible improvement in the Digital core would be to add new RISC-V extensions. The Digital core implements RV32I instruction set, but there are many others: M, A, F, D, Ztso, and more.

5.2 Future Scope.

The future scope of SCR1 is promising, as the RISC-V architecture is gaining popularity in the semiconductor industry due to its open-source nature, scalability, and flexibility. The SCR1, being a RISC-V processor, is likely to find increasing use in a wide range of applications, including IoT devices, embedded systems, and high-performance computing.

One of the key advantages of the RISC-V architecture is its scalability, which allows for the creation of processors with varying levels of complexity and performance. The Digital core, being a small and low-power processor, is well-suited for use in low-cost and low-power embedded systems, such as sensors, controllers, and wearables. However, the RISC-V architecture also supports the creation of more powerful processors, which could be used in higher-end applications, such as servers, networking equipment, and high-performance computing systems.

Another advantage of the RISC-V architecture is its open-source nature, which allows for greater collaboration and innovation within the semiconductor industry. The availability of open-source tools and software for RISC-V processors, such as compilers, debuggers, and operating systems, is likely to drive the development of new applications and use cases for the core and other RISC-V processors.

Finally, the Core includes a range of security features, such as support for a secure boot process, hardware-based cryptography, and a memory protection unit (MPU) for enforcing memory access permissions. As security becomes an increasingly important concern in the semiconductor industry, the inclusion of these features is likely to make the Core more attractive to designers who need a processor core with robust security capabilities.

The core is designed to be easily integrated into larger system-on-chip (SoC) designs, which makes it well-suited for use in complex devices that require multiple processors and other IP blocks. As the demand for complex, feature-rich devices continues to grow, we can expect to see more SoCs that include RISC-V processors like the SCR1.

Overall, the future scope of the core is bright, as the RISC-V architecture continues to gain momentum in the semiconductor industry and the demand for low-power, highly configurable processors grows in a wide range of applications.