HETEROGENEOUS MULTI-CORE SOC ARCHITECTURE FOR REAL-TIME BIOMEDICAL DATA ANALYSIS

A PROJECT REPORT

Submitted by

BALAJI S 22EC019
MANIKANDAN L 22EC070

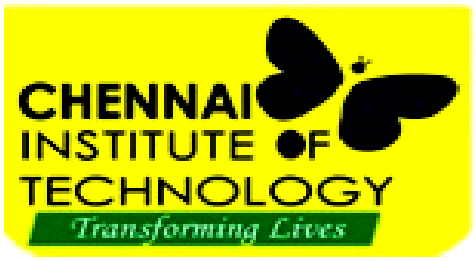
in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

ELECTRONICS AND COMMUNICATION ENGINEERING



CHENNAI INSTITUTE OF TECHNOLOGY (AUTONOMOUS)
(Affiliated to Anna University, Chennai)
CHENNAI-600 069

ANNA UNIVERSITY: CHENNAI-600 025
SEPTEMBER-2025

ANNA UNIVERSITY: CHENNAI-600 025

BONAFIDE CERTIFICATE

Certified that this project report "**HETEROGENEOUS MULTI-CORE SOC ARCHITECTURE FOR REAL-TIME BIOMEDICAL DATA ANALYSIS**" is the Bonafide work of **BALAJI S (22EC019), MANIKANDAN L (22EC070)** who carried out the project work under my supervision.

SIGNATURE

Dr. P. SURESH KUMAR, M.E., Ph.D.,

Head of Department

Department of Electronics and Communication Engineering,

Chennai Institute of Technology,

Chennai-600069.

SIGNATURE

Ms.A. SUBBULAKSHMI M.E.,

Assistant Professor

Department of Electronics and Communication Engineering,

Chennai Institute of Technology,

Chennai-600069.

Certified that the above students have attended a viva voice during the exam held on…….....

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

We express our gratitude to our Chairman **Shri.P. SRIRAM** and all trust members of Chennai institute of technology for providing the facility and opportunity to do this project as a part of our undergraduate course.

We are grateful to our Principal **Dr.A. RAMESH M.E**, **Ph.D.** for providing us the facility and encouragement during the course of our work.

We sincerely thank our Head of the Department, **Dr. P.SURESH KUMAR M.E.,Ph.D.,** Department of Electronics and Communication Engineering for having provided us valuable guidance, resources and timely suggestions throughout our work.

We sincerely thank our Program Coordinator, **Mrs. R.PANDIMEENA M.E.,(Ph.D).,** Department of Electronics and Communication Engineering for having provided us valuable guidance, resources and timely suggestions throughout our work.

We sincerely thank our Project Guide, **Ms. SUBBULAKHMI M.E.**, Professor, Department of Electronics and Communication Engineering for having provided us with valuable guidance, resources and timely suggestions throughout our work.

We would like to extend our thanks to our **Dr. T. S. Arulanath, M.Tech., Ph.D.,** College project coordinator and **Dr. R. Alaguselvi, M.E., Ph.D.,** project coordinator of the Department of Electronics and Communication Engineering for their valuable suggestions throughout the project.

We wish to extend our sincere thanks to **Mr. S. Vignesh Kumar M.E.,** year coordinator Department of Electronics and Communication Engineering for their valuable suggestions throughout the project.

We would like to extend our thanks to our **Faculty coordinators of the Department of Electronics and Communication Engineering**, for their valuable suggestions throughout this project.

We wish to acknowledge the help received from the **Lab Instructors of the Department of Electronics and Communication Engineering** and others for providing valuable suggestions and for the successful completion of the project.

1. VISION OF THE INSTITUTION:

To be an eminent centre for Academia, Industry and Research by imparting knowledge, relevant practices and inculcating human values to address global challenges through novelty and sustainability.

2. MISSION OF THE INSTITUTION:

IM1: To create next generation leaders by effective teaching learning methodologies and instill scientific spark in them to meet the global challenges.

IM2: To transform lives through deployment of emerging technology, novelty and sustainability.

IM3: To inculcate human values and ethical principles to cater to the societal needs.

IM4: To contribute towards the research ecosystem by providing a suitable, effective platform for interaction between industry, academia and R & D establishments.

IM5: To nurture incubation centres enabling structured entrepreneurship and start-ups.

Department of Electronics and communication Engineering

VISION OF THE DEPARTMENT:

To Excel in the emerging areas of Electronics and Communication Engineering by imparting knowledge, relevant practices and inculcating human values to transform the students as potential resources to cater the industrial and societal development through sustainable technology growth.

MISSION OF THE DEPARTMENT:

DM1: To provide strong fundamentals and technical skills through effective teaching learning Methodologies.

DM2: To transform lives of the students by fostering ethical values, creativity and innovation to become Entrepreneurs and establish Start-ups.

DM3: To habituate the students to focus on sustainable solutions to improve the quality of life and welfare of the society.

DM4: To provide an ambience for research through collaborations with industry and academia.

DM5: To inculcate learning of emerging technologies for pursuing higher studies leading to lifelong learning.

Program Outcomes as defined by NBA (PO) Engineering Graduates will be able to:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes (PSOs)

S.No.	Programme Specific Outcomes
PSO1	To analyze, design and develop quality solutions in Communication Engineering by adapting the emerging technologies.
PSO2	To innovate ideas and solutions for real time problems in industrial and domestic Automation using Embedded & IOT tools.

Program Educational Objectives:

Graduates will be able to

PEO1: Contribute to the industry as an Engineer through sound knowledge acquired in core engineering to develop new processes and implement the solutions for industrial problems.

PEO2: Establish an organization / industry as an Entrepreneur with professionalism, leadership quality, teamwork, and ethical values to meet the societal needs.

PEO3: Create a better future by pursuing higher education / research and develop the sustainable products / solutions to meet the demand.

TABLE OF CONTENTS

ABSTRACT

This project presents the design, implementation, and verification of a **heterogeneous multi-core System-on-a-Chip (SoC)** architecture tailored for real-time biomedical and general-purpose data analysis. The work addresses the fundamental challenges of current low-power devices, which are constrained by the performance and power limitations of general-purpose processors when processing complex physiological signals at the edge. The proposed SoC architecture leverages a unique hardware-software co-design approach to achieve superior performance, power efficiency, and diagnostic accuracy.The core of the SoC is a custom **RISC-V processor** with a **dual-core architecture**. A **General-Purpose Core** manages system control, while a specialized **DSP Core** is augmented with a custom instruction set that includes a **Vector Multiply-Accumulate (VMAC)** instruction. This accelerates key digital signal processing (DSP) algorithms, such as those used for noise reduction in **Electrocardiogram (ECG) signals** or audio processing. The SoC also incorporates an on-chip **AI/ML Accelerator** to enable intelligent, on-device analysis. This hardware block is optimized for running a tiny **Convolutional Neural Network (CNN)**, which can perform automated classification of sensor data. The system's data transfer is managed by a **DMA controller**, which autonomously moves data from an **SPI peripheral** to **on-chip memory**, freeing the cores from I/O overhead.The entire design is implemented in **Verilog/System Verilog** and verified using a comprehensive **UVM (Universal Verification Methodology)** testbench. The project demonstrates a holistic understanding of digital logic, computer architecture, embedded systems, and advanced verification methodologies. Ultimately, this work delivers a viable and highly optimized platform for various **IoT, wearable, and medical applications**, showcasing a significant advancement over existing general-purpose solutions.

LIST OF FIGURES

LIST OF ABBREVATIONS

Abb	Expansion
SoC	System on chip
RISC-V	Reduced Instruction Set Computing
ISA	Instruction Set Architecture
ASIC	Application-Specific Integrated Circuit
DMA	Direct Memory Access
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver-Transmitter
ECG	Electrocardiogram
FFT	Fast Fourier Transform
CNN	Convolutional Neural Network
IoMT	Internet of Medical Things
AHB	Advanced High-performance Bus
FIR	Finite Impulse Response

CHAPTER 1: INTRODUCTION

1.1 Project Overview

This project focuses on the design and implementation of a **heterogeneous multi-core System-on-a-Chip (SoC)** tailored for real-time data analysis. While the core concepts are generalizable to any application requiring intensive signal processing, this report uses **Electrocardiogram (ECG) analysis** as the primary example to demonstrate the system's capabilities. The project aims to develop a specialized computing platform that is superior to existing general-purpose solutions by optimizing both hardware and software.

The SoC architecture is built around a **custom RISC-V processor**, which has been extended with a **specialized instruction set** for accelerating digital signal processing (DSP) and machine learning (ML) tasks. This platform integrates key hardware components, including a **Direct Memory Access (DMA) controller** for efficient data handling, an **AI/ML accelerator** for on-device diagnosis, and various communication peripherals like **SPI** and **UART**. The entire design is implemented in **Verilog/System Verilog** and verified using a comprehensive **Universal Verification Methodology (UVM)** testbench.

By combining custom hardware with optimized firmware, the project demonstrates a complete, end-to-end solution that addresses the critical need for a low-power, high-performance, and intelligent system for a wide range of applications, including **wearable medical devices**, **industrial sensors**, and the Internet of Things (IoT).

1.2 Problem Statement

The core problem is the need for a **specialized, low-power, and high-performance computing platform** capable of processing complex, real-time data at the source. While this challenge exists across many fields, it is particularly acute in **biomedical and wearable applications** like ECG monitoring. These systems face a fundamental

conflict between the need for **instantaneous analysis** and the constraints of **limited power and computational resources**.

The current solutions are inadequate for several reasons:

- **Computational Inefficiency of General-Purpose Processors (GPPs):** GPPs are designed to be versatile, but this comes at the cost of efficiency for specific tasks. Biomedical signal analysis, such as filtering and feature extraction, relies heavily on repetitive mathematical operations like **Multiply-Accumulate (MAC)**. On a standard processor, these tasks require multiple clock cycles per operation, leading to significant delays and high power consumption. This makes them unsuitable for real-time applications where every microsecond and milliwatt matters.
- **Latency and Security Risks of Cloud-Based Solutions:** One common approach is to offload data processing to the cloud. A simple device collects raw data and sends it to a powerful remote server for analysis. However, this introduces **latency**, which can be unacceptable for critical diagnoses where time is of the essence. It also creates a major **privacy and security risk**, as sensitive patient data must be transmitted over a network.
- **Lack of Flexibility in Existing SoCs:** Some commercial SoCs include a dedicated DSP unit to accelerate signal processing. While efficient, these are **"black box" solutions**. The hardware is fixed, and the user cannot modify the processor's architecture to add custom instructions or tailor it to a unique algorithm. This limits innovation and prevents a deep level of application-specific optimization.

This project addresses these challenges directly. Instead of using a one-size-fits-all processor or relying on a remote server, we propose an **Application-Specific SoC** with a custom instruction set and hardware accelerators. This approach resolves the conflict between performance and power, enabling a new class of intelligent, on-device computing for real-time analysis.

1.3 Proposed Solution

The proposed solution is the design and implementation of a **custom System-on-a-Chip (SoC)** built on a **heterogeneous multi-core RISC-V architecture**. This SoC is a specialized computing platform, engineered from the ground up to overcome the limitations of general-purpose processors. Instead of relying on a single, all-purpose CPU, our design incorporates two distinct cores, each optimized for its specific tasks.

The architecture features:

- **A General-Purpose Core** to manage high-level system control, handle communication protocols like **UART** and **SPI**, and orchestrate the overall workflow. This core remains free from computationally intensive tasks, ensuring a responsive system.
- **A Dedicated DSP Core** equipped with a custom instruction set. This core includes a **Vector Multiply-Accumulate (VMAC)** instruction, which performs multiple calculations in a single clock cycle, drastically accelerating digital signal processing algorithms like the **FIR filter**.
- **On-Chip Accelerators** for specific functions. A **DMA controller** automates data transfers, freeing up the CPU. An **AI/ML accelerator** provides the hardware necessary to run a **Convolutional Neural Network (CNN)** for intelligent, on-device analysis.

This integrated architecture allows the SoC to process large volumes of data in real time with high energy efficiency, enabling a new class of smart, battery-powered devices.

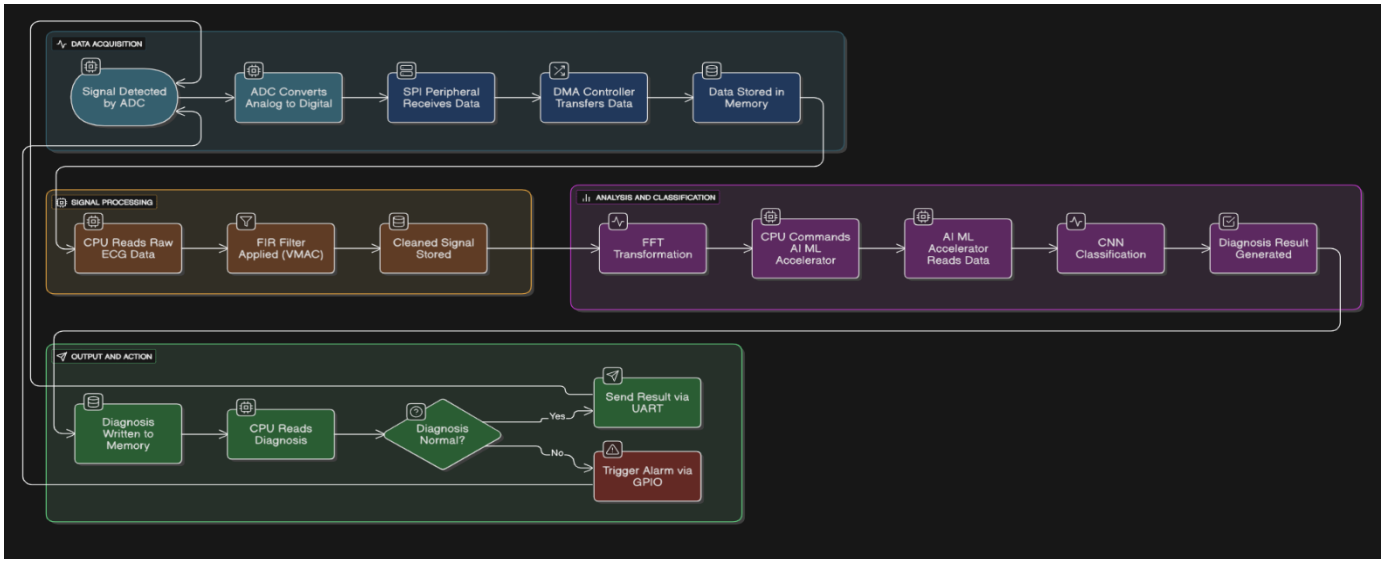


Fig 1.3.1 Flow of Proposed Solution

1.4 Key Innovations

This project distinguishes itself from standard embedded systems and existing commercial solutions through a number of key innovations:

- **Custom RISC-V Instruction Set Architecture (ISA):** The central innovation is the extension of the open-source RISC-V ISA with custom instructions. Specifically, the inclusion of a **Vector Multiply-Accumulate (VMAC)** instruction and a **Bit-Reversal** instruction directly into the processor's pipeline provides hardware-level acceleration for digital signal processing (DSP) algorithms. This fundamental architectural modification bypasses the limitations of a generic instruction set, enabling a massive reduction in clock cycles and power consumption for core computational tasks.
- **Heterogeneous Multi-Core SoC Architecture:** The design implements a specialized dual-core system. A **General-Purpose Core** is dedicated to system management and peripheral control, while a separate, more powerful **DSP Core** is tasked exclusively with the intensive signal processing. This

division of labor allows for parallel processing, maximizing throughput and ensuring that real-time tasks are not hindered by general system overhead.

- **On-Chip AI/ML Acceleration:** The project integrates a dedicated hardware accelerator for machine learning inference. This specialized block, optimized for parallel computation, runs a **Convolutional Neural Network (CNN)** directly on the processed sensor data. This capability enables intelligent, on-device diagnosis without relying on cloud resources, which is a significant advancement in terms of latency, privacy, and security for edge computing applications.
- **Comprehensive Hardware-Software Co-Design:** This project is not merely a hardware or software endeavor. It represents a holistic approach where the hardware architecture is designed in tandem with the software algorithms. The custom instructions are created to optimize the very C-code loops that perform the FIR filtering and FFT, showcasing a deep understanding of how to tailor silicon to an application. The entire system—from the **DMA-based data transfer** to the **UVM verification methodology**—is meticulously designed to function as a unified, highly efficient platform.

CHAPTER 2: BACKGROUND AND LITERATURE REVIEW

2.1. Biomedical Signal Processing Fundamentals

Biomedical signals are recordings of physiological activity in living organisms. They can be electrical, chemical, or mechanical, and are crucial for non-invasive diagnosis and monitoring of various medical conditions.

2.1.1 The ECG Signal: A Window into Heart Health

For this project, the **Electrocardiogram (ECG)** signal is of paramount importance. An ECG records the electrical signals that drive the heart's contractions. A typical ECG waveform is composed of several key components that represent the distinct phases of the cardiac cycle:

- **P wave:** A small upward deflection representing atrial depolarization, which causes the atria to contract.
- **QRS complex:** A large, sharp spike representing ventricular depolarization, causing the powerful contraction of the ventricles. This complex is the most prominent and is the primary feature used for heart rate calculation.
- **T wave:** Represents ventricular repolarization, the electrical recovery of the ventricles.

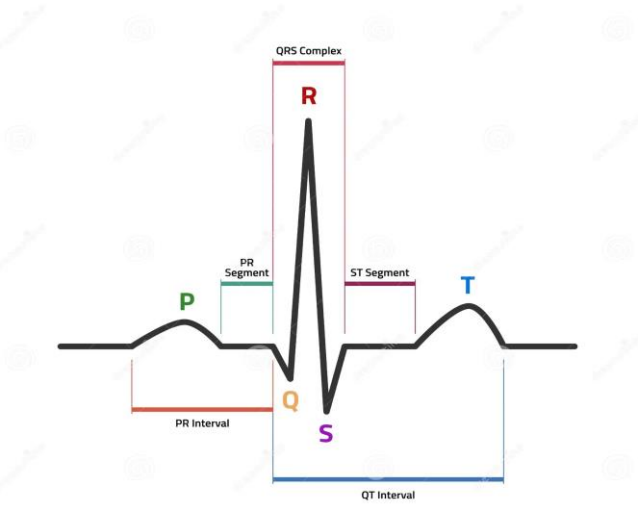


Fig 2.1.1.1 ECG Waveform

The amplitude, timing, and morphology (shape) of these waves provide critical information about the heart's health. Any deviations can indicate a range of cardiac issues, from minor arrhythmias to life-threatening conditions like a myocardial infarction.

2.1.2 The Challenge: Noise and Artifacts

Biomedical signals are inherently prone to noise and artifacts. This interference can obscure the clinically relevant information, making accurate analysis difficult. The sources of this noise include:

- **Baseline Wander:** Low-frequency drift caused by patient movement or breathing.
- **Muscle Artifacts (Electromyographic Noise):** High-frequency interference from skeletal muscle activity.
- **Powerline Interference:** A strong, narrow-band noise at 50 Hz or 60 Hz caused by electrical power grids.
- **Motion Artifacts:** Irregular spikes and shifts from the movement of the electrodes.

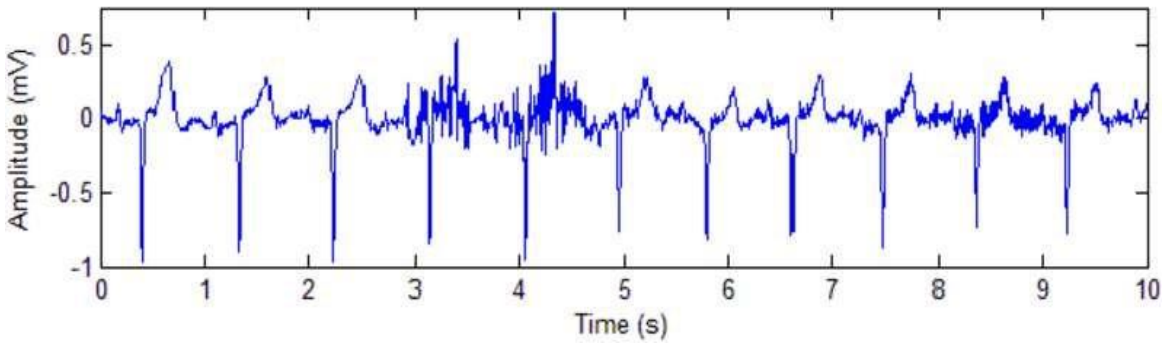


Fig 2.1.2.1 ECG Waveform with Noise

Without a robust signal processing methodology, this noise could lead to an incorrect diagnosis. The challenge is to effectively remove these artifacts while preserving the integrity of the underlying ECG signal. This necessitates a sophisticated and efficient system for real-time data filtering.

2.1.3 The Role of Digital Signal Processing (DSP)

To overcome these challenges, **Digital Signal Processing (DSP)** is indispensable. DSP involves using a digital processor to manipulate signals. In this context, it is used to:

- **Enhance the Signal:** Filtering out unwanted noise to reveal the clean, true ECG waveform.
- **Extract Features:** Isolating key components like the QRS complex to accurately determine the heart rate and rhythm.
- **Analyze the Signal:** Preparing the signal for advanced analysis techniques, such as those powered by machine learning, for automated diagnosis.

2.2. Digital Signal Processing (DSP)

Digital Signal Processing (DSP) is the art of using a digital processor to manipulate signals. For your project, DSP is the cornerstone for extracting meaningful information from the raw ECG data. Two fundamental DSP algorithms are critical to your solution.

2.2.1. Finite Impulse Response (FIR) Filters

FIR filters are a class of digital filters that have no feedback loop, meaning their output depends only on the current and previous input samples. This key characteristic makes them **inherently stable**, which is a crucial advantage for medical applications where reliability is paramount.

The operation of an FIR filter can be described by the following convolution sum:

$$y[n]=\sum k=0N-1h[k]\cdot x[n-k]$$

where y[n] is the output signal, x[n] is the input signal, and h[k] are the filter coefficients. The core of this equation is a series of **multiply-accumulate (MAC)**

operations. For each output sample, the filter multiplies a series of input samples by corresponding coefficients and then sums the results. This is precisely why your custom **Vector MAC (VMAC)** instruction is so powerful: it allows your SoC to perform these fundamental filter calculations with unprecedented efficiency.

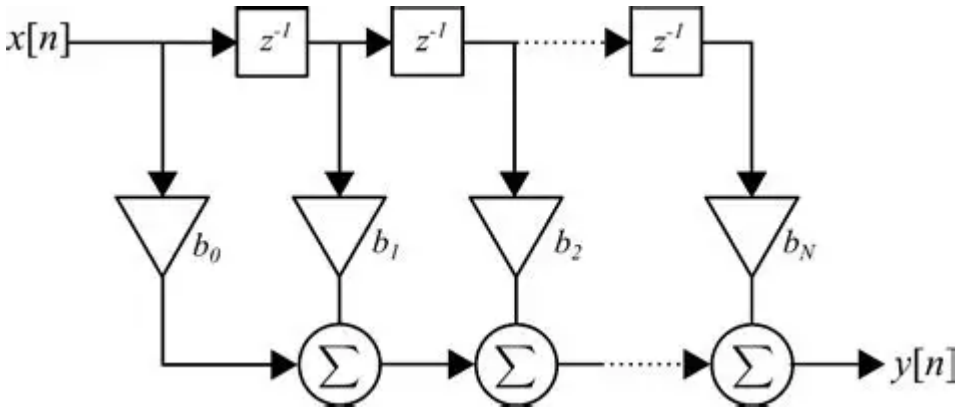


Fig 2.2.1.1 FIR Filter Diagram

2.2.2. Fast Fourier Transform (FFT)

The **Fast Fourier Transform (FFT)** is an extremely efficient algorithm for converting a signal from the **time domain** to the **frequency domain**. This is a powerful analytical tool because it reveals the signal's frequency content.

- **Time Domain:** A signal in the time domain plots amplitude (e.g., voltage) against time, like a standard ECG waveform. While useful for observing peaks and patterns, it can be difficult to identify specific noise frequencies.
- **Frequency Domain:** The FFT transforms this signal into a plot of amplitude against frequency. In this domain, noise sources, such as the distinct 50/60 Hz **powerline interference**, appear as sharp, easily identifiable peaks. This allows for precise noise removal without affecting the vital frequencies of the ECG signal.

2.3. RISC-V Architecture

The **RISC-V** (pronounced "risk-five") architecture is a powerful and open-source **Instruction Set Architecture (ISA)** that serves as the foundational technology for this project. Unlike proprietary ISAs like x86 or ARM, RISC-V is not owned by any single company. This open, royalty-free nature is a key advantage, as it allows for unparalleled customization and innovation, which is central to the project's thesis.

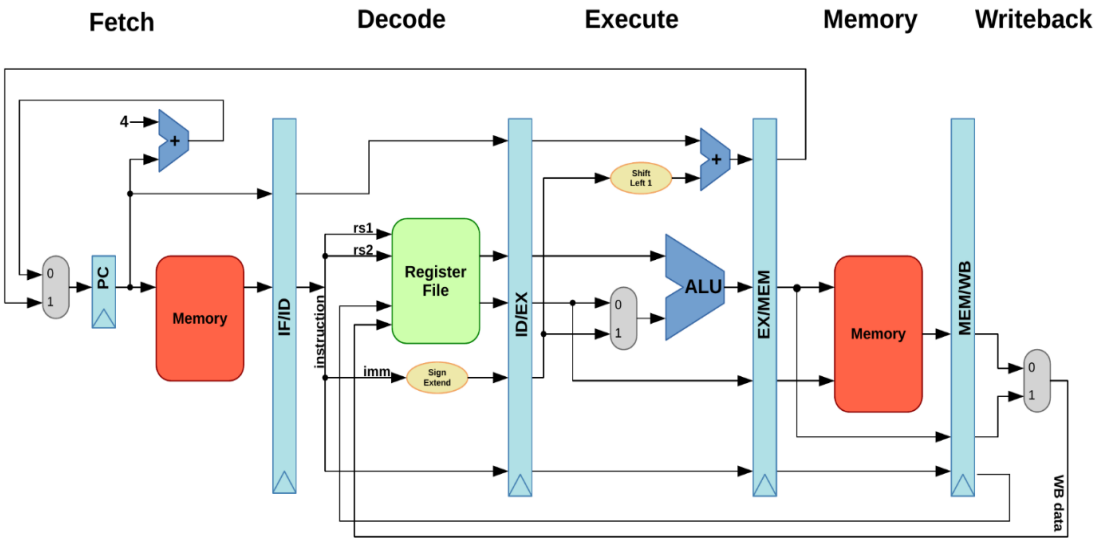


Fig 2.3.1 RISC-V Architecture

2.3.1. Overview and Key Principles

RISC-V is a modern ISA based on the **Reduced Instruction Set Computer (RISC)** philosophy, which emphasizes a small, simple, and highly efficient set of instructions. This is in contrast to Complex Instruction Set Computers (CISC), which have a large and intricate instruction set. The core design principles of RISC-V include:

- **Simplicity and Modularity:** The ISA is built upon a minimal base instruction set (e.g., RV32I for a 32-bit integer core). This base can be extended with a suite of standard extensions (e.g., 'M' for multiplication/division, 'A' for atomics, 'F' for floating-point) or, most importantly, **custom extensions**

defined by the user. This modularity allows a designer to create a processor that includes only the necessary features, optimizing for area and power consumption.

- **Scalability:** The architecture is designed to scale from small, embedded microcontrollers to high-performance supercomputers. This flexibility ensures that the principles and design choices in this project can be applied to a wide range of future applications.
- **Openness:** The open-source license allows anyone to design, manufacture, and sell RISC-V chips without paying licensing fees. This fosters a collaborative ecosystem and provides complete design freedom, a critical aspect that enables the unique features of this SoC.

2.3.2. The Custom Instruction Space: Enabling Innovation

The most critical feature of RISC-V for this project is its explicitly defined **custom instruction space**. The RISC-V ISA specification deliberately reserves a portion of the opcode map for user-defined instructions. This empowers a hardware designer to create and implement entirely new instructions that are not part of the standard.

For the purposes of this project, this feature is exploited to design and implement two application-specific instructions:

1. **Vector Multiply-Accumulate (VMAC):** This instruction is designed to perform the core mathematical operation of a digital filter in a single clock cycle, which would otherwise require multiple cycles on a standard processor.
2. **Bit-Reversal:** This instruction accelerates the data reordering required by the Fast Fourier Transform (FFT) algorithm.

These custom instructions are integrated into the processor's pipeline, allowing the hardware to be precisely tailored to the specific software algorithms running on it. This is a prime example of **hardware-software co-design**, where the instruction set is co-designed with the algorithms it is meant to run, leading to significant performance and efficiency gains.

2.3.3. Justification for RISC-V

The selection of RISC-V as the core architecture is fundamentally justified by its ability to directly address the project's central problem. By modifying the ISA, the SoC is transformed from a general-purpose processor into an **Application-Specific Instruction Set Processor (ASIP)**. This approach provides several key advantages over alternative architectures:

- **Performance:** The custom VMAC instruction directly accelerates the most computationally expensive part of the signal processing pipeline, achieving superior performance that would be impossible to match with an off-the-shelf processor lacking a dedicated DSP unit.
- **Efficiency:** By consolidating a complex operation into a single instruction, the SoC reduces the number of clock cycles and power consumption, making it ideal for battery-powered, real-time medical devices.
- **Design Freedom:** The open-source nature of RISC-V allows for complete control over the design, enabling the integration of a custom AI/ML accelerator and other specialized peripherals without the constraints of a proprietary ecosystem.

In summary, RISC-V provides the ideal platform for this project, not because it is an out-of-the-box solution, but because it is a **flexible framework** that enables the core innovation of a highly optimized, application-specific SoC.

2.4. System on a Chip (SoC) Architectures

A **System on a Chip (SoC)** is a complete electronic system integrated onto a single silicon chip. Instead of using multiple discrete components like a separate CPU, memory, and peripherals, an SoC combines all these functional blocks into one integrated circuit. This approach offers significant advantages in terms of size, power consumption, and overall performance, making it the ideal choice for compact, battery-powered devices like medical wearables.

2.4.1. Key Components of an SoC

A typical SoC architecture comprises several core components that work together through an internal communication network. The primary components include:

- **Processor Cores:** The "brains" of the SoC. This can include one or more CPUs, DSPs, or specialized application-specific processors. In a **heterogeneous multi-core architecture**, different types of cores are used for different tasks, which is the model for your project.
- **Memory Subsystem:** This includes various types of memory. **Cache memory** (SRAM) is used for fast, temporary storage close to the processor. Larger on-chip or off-chip **main memory** (DRAM/SRAM) stores the program code and data.
- **Peripherals:** These are modules that handle communication with external devices. Examples relevant to your project include **SPI** for sensor data input and **UART** for debugging and outputting results.
- **Interconnect:** An on-chip communication bus (e.g., AXI, AHB) that connects all the components, enabling them to communicate efficiently. A **DMA controller** is often integrated here to handle high-speed data transfers without CPU intervention.
- **Accelerators:** Dedicated hardware blocks designed to speed up specific, computationally intensive tasks. Your project's **AI/ML accelerator** is a prime example of this.

2.4.2. Design Principles for Medical SoCs

For an SoC designed for real-time medical applications, several design principles are paramount:

- **Low Power Consumption:** This is critical for extending the battery life of wearable devices. This is achieved through efficient architectures and the use of specialized, low-power hardware blocks.
- **Real-Time Performance:** The SoC must be able to process data in a timely manner to provide instant feedback and detect critical events without latency. This necessitates a high-speed processor and efficient data pipelines.
- **Hardware-Software Co-Design:** The hardware architecture is designed with the software algorithms in mind. This tight integration ensures that the most common and expensive software operations are directly supported by efficient hardware instructions, leading to significant performance gains. Your custom RISC-V instructions are a direct result of this principle.
- **Modularity and Scalability:** The architecture should be modular to allow for the easy integration of different peripherals or specialized cores in the future. This ensures the design can be adapted for other applications beyond ECG.

The choice to design a custom SoC for this project is a direct response to the limitations of off-the-shelf microcontrollers, which are not optimized to meet the stringent power, performance, and real-time demands of advanced biomedical signal analysis. The custom SoC approach provides complete control over the hardware, allowing you to create a solution that is both highly efficient and uniquely capable.

2.5. Existing Solutions and Their Limitations

Currently, several approaches and existing hardware solutions are used for real-time biomedical signal processing, each with distinct limitations that our project aims to overcome. Understanding these limitations is key to highlighting the unique value of our custom SoC.

2.5.1. General-Purpose Microcontrollers (MCUs)

Many low-cost medical wearables rely on general-purpose MCUs, such as those based on the **ARM Cortex-M** architecture. These processors are widely available, have a large software ecosystem, and are very power-efficient for general tasks.

- **Limitations:** Their primary drawback is a **lack of computational efficiency for DSP tasks**. A standard MCU's instruction set is not optimized for repetitive mathematical operations like the **Multiply-Accumulate (MAC)** function. Consequently, a single MAC operation, which is the cornerstone of a digital filter, requires multiple clock cycles. This inefficiency results in:
- **Increased Latency:** The processing takes longer, which is a major issue in real-time applications.
- **Higher Power Consumption:** The processor must remain active for more clock cycles to complete the task, draining the battery faster.

2.5.2. Commercial Digital Signal Processors (DSPs)

Some medical devices use dedicated **Digital Signal Processors (DSPs)**. These chips are specifically designed for signal processing and include hardware features like dedicated MAC units.

- **Limitations:** While computationally efficient, these are often **"black box" solutions**. The hardware architecture is fixed, and the instruction set is proprietary. This means we, as designers, have no control over the core architecture. We cannot add new, application-specific instructions to accelerate a new algorithm, nor can we customize the core for power or area optimization beyond what the vendor provides. This limits innovation at the architectural level.

2.5.3. Cloud-Based Processing

Another modern approach is to send raw, unprocessed sensor data to a nearby device (like a smartphone) or a cloud server for analysis.

Limitations:

- **Latency:** Data transmission introduces a significant delay, making this approach unsuitable for critical, real-time event detection where milliseconds matter.

- **Connectivity Dependence:** The solution fails if the device loses its network connection.
- **Privacy and Security:** Transmitting sensitive patient data introduces significant privacy risks and requires complex security measures to prevent unauthorized access.

2.5.4. The Unique Value of Our Custom SoC

Our project provides a superior solution by directly addressing the limitations of these existing approaches.

- **Performance and Power:** By designing a custom **Application-Specific Instruction Set Processor (ASIP)** with a **Vector MAC (VMAC)** instruction, our SoC performs the most demanding DSP operations in a single clock cycle. This architectural innovation provides a **significant performance and power-efficiency advantage** over general-purpose MCUs.
- **Design Flexibility and Innovation:** Unlike proprietary DSPs, our **RISC-V-based** design is open and fully customizable. We can add new instructions, integrate specialized AI accelerators, and optimize the hardware for new applications, demonstrating a deeper level of design control and expertise.
- **Edge Intelligence and Security:** Our solution processes all data on-device, eliminating the need for constant network connectivity and minimizing latency. This also enhances **data privacy and security** by keeping sensitive information localized to the patient's device.

Our project, therefore, represents a unique and powerful alternative, combining the flexibility of an open-source platform with the performance of a dedicated hardware accelerator to create a solution tailored specifically for the challenges of real-time biomedical data analysis.

CHAPTER 3: SYSTEM ON CHIP (SoC) ARCHITECTURE AND DESIGN

3.1 Overall SoC Architecture

The proposed System-on-a-Chip (SoC) is a heterogeneous multi-core architecture designed for optimal performance and power efficiency in real-time signal processing applications. The top-level design is a modular system built around a central communication bus, which ensures all components can interact seamlessly. The architecture separates the computationally intensive tasks from general system management, allowing both to operate in parallel.

Top-Level Design Explanation

The SoC can be conceptually divided into three main blocks: the Core Processing Unit, the Peripherals and Accelerators, and the Memory Subsystem. All these blocks are interconnected by a high-speed On-Chip Bus, which acts as the communication backbone.

1. **Core Processing Unit**: This is the "brain" of the SoC and consists of two specialized RISC-V cores to handle different types of tasks.
    - General-Purpose Core: This core is responsible for high-level control, task orchestration, and communication with the outside world via peripherals. It ensures the system is responsive and manages the overall workflow.
    - DSP Core: This core is the computational powerhouse. It contains the custom instruction set architecture (ISA), including the Vector Multiply-Accumulate (VMAC) instruction, making it highly efficient at performing digital signal processing (DSP) and complex mathematical operations. It operates on data handed to it by the General-Purpose Core.

2. **Peripherals and Accelerators**: These are the specialized hardware blocks that handle specific functions, freeing up the CPU cores for more critical tasks.
    - DMA Controller: This block is essential for efficient data transfer. It moves data directly from the input peripherals to the memory without involving the CPU cores, thereby reducing I/O overhead.
    - AI/ML Accelerator: A dedicated hardware block designed to perform Convolutional Neural Network (CNN) inference. It receives data from the DSP Core and outputs a classification result, enabling on-device intelligence.
    - Communication Peripherals (SPI, UART): These modules manage data input from external sensors (SPI) and data output to a host device (UART).
3. **Memory Subsystem**: This block provides storage for both program code and data, structured to minimize latency.
    - Data Memory: A high-speed SRAM that acts as the primary working memory for all data, including raw sensor readings, filter coefficients, and processed results.
    - Instruction Memory: A separate SRAM for storing the firmware executed by the cores.
    - Cache: Small, fast on-chip memories (L1/L2 caches) are integrated with the cores to provide rapid access to frequently used data and instructions, reducing the need to access the slower main memory.

The entire system's workflow is orchestrated by the General-Purpose Core, which hands off computationally intensive tasks to the DSP Core and accelerators. The communication between all these components is managed by the On-Chip Bus, which routes data requests and responses efficiently, making the SoC a cohesive and high-performance integrated system.

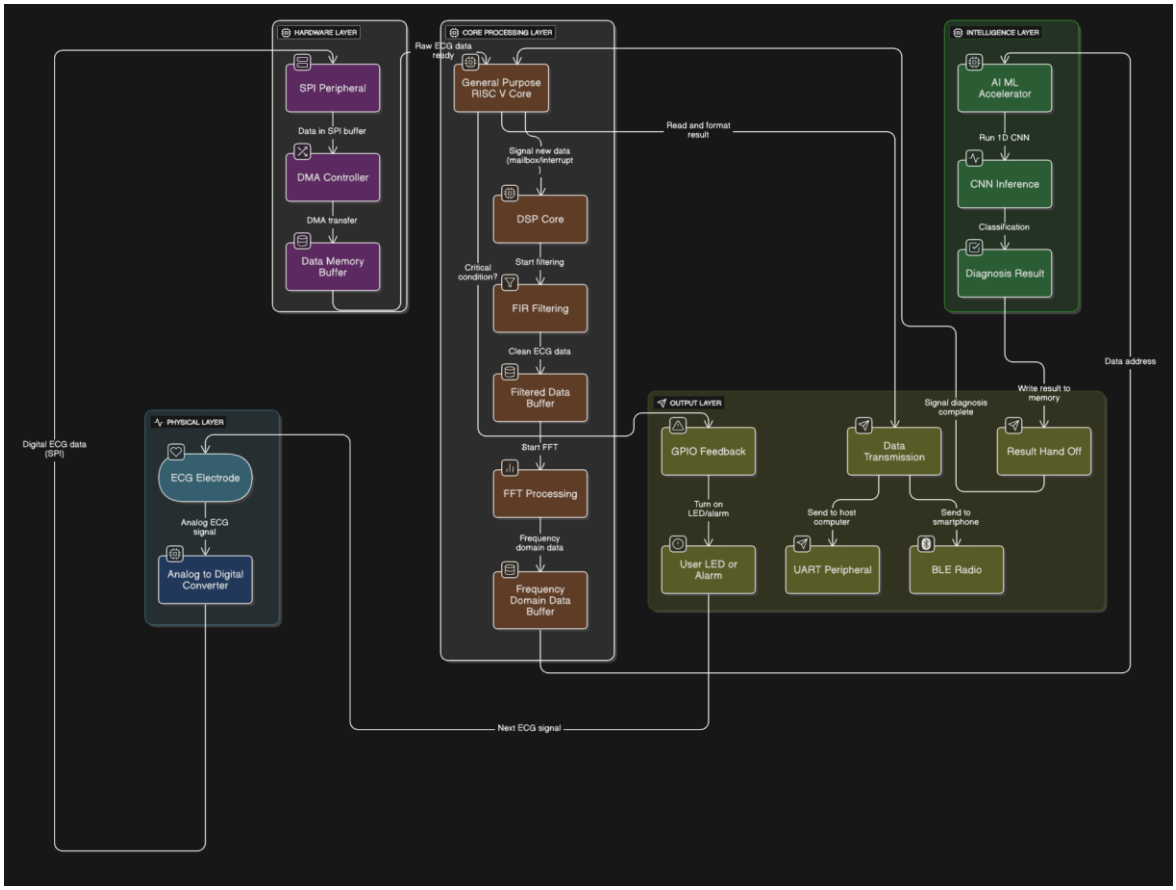


Fig 3.1.1 Overall SoC Architecture

3.2 Processor Core Design

The heart of the SoC is a sophisticated processor design that moves beyond a simple, off-the-shelf core. It is built on a modular, pipelined RISC-V architecture and enhanced with custom features to meet the specific demands of real-time signal processing.

3.2.1 The Pipelined RISC-V Base Core

The project uses a 5-stage pipelined RISC-V core as its foundation. A pipeline is a technique that breaks down instruction execution into a series of smaller, sequential steps, allowing multiple instructions to be processed concurrently, thus increasing overall throughput. The five stages are:

- **IF (Instruction Fetch)**: The instruction is fetched from instruction memory.
- **ID (Instruction Decode)**: The instruction is decoded, and its operands are read from the register file.
- **EX (Execute)**: The arithmetic and logical operations are performed by the ALU (Arithmetic and Logic Unit).
- **MEM (Memory Access)**: Data is read from or written to data memory.
- **WB (Write Back)**: The result of the operation is written back to the register file.

This standard pipeline provides a robust and efficient starting point for the custom design, offering a balance of performance and complexity.

3.2.2 Custom Instruction Set Architecture (ISA) Extension

The key innovation lies in extending the open-source RISC-V ISA, which is designed to be extensible. We have implemented two new instructions to accelerate the most computationally intensive tasks in digital signal processing:

- **Vector Multiply-Accumulate (VMAC) Instruction**: This custom instruction is vital for accelerating the Finite Impulse Response (FIR) filter and other DSP algorithms. A standard MAC operation (A = A + B * C) requires multiple cycles on a general-purpose processor. Our custom instruction performs this operation on a vector of data points in a single cycle. We have modified the EX stage of the pipeline to include a dedicated hardware unit for this purpose, bypassing the standard ALU. This dramatically reduces the number of clock cycles and improves power efficiency.
- **Bit-Reversal Instruction**: The Fast Fourier Transform (FFT) algorithm, used for frequency analysis, requires a data reordering step known as bit-reversal. Performing this in software is slow. We have added a dedicated instruction to the ISA that performs the bit-reversal operation in a single clock cycle. This acceleration is critical for real-time applications that rely on fast frequency analysis.

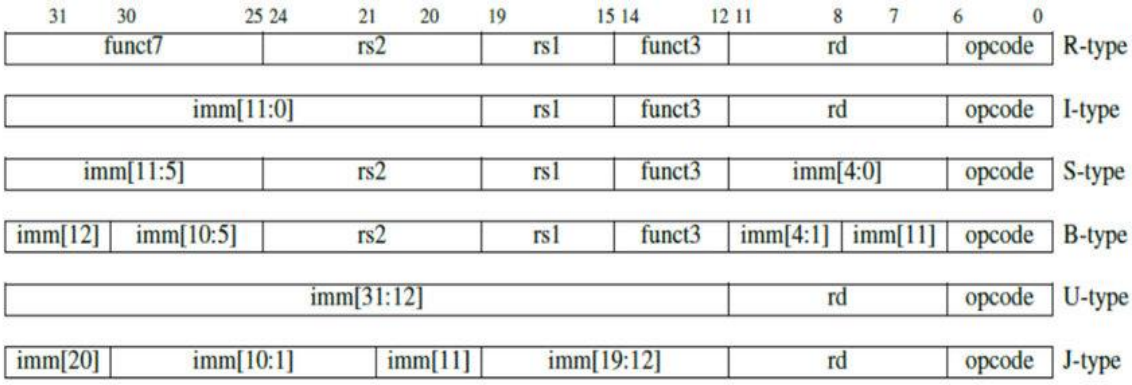


Fig 3.2.2.1 RISC-V Instruction Set

These custom instructions are recognized and executed by a modified control unit and data path, which were design ed specifically to handle their unique operations.

3.2.3 Multi-Core Architecture and Task Division

To maximize parallelism and efficiency, the SoC employs a heterogeneous multi-core architecture. Instead of a single, do-it-all core, tasks are intelligently distributed between two specialized processors.

General-Purpose Core: This core is a standard, low-power RISC-V processor. Its role is to serve as the system's "manager." It handles all general system tasks, including:

- Configuring and managing peripherals (DMA, SPI, UART).
- Running the main application firmware and user interface logic.
- Orchestrating the flow of data and control signals.

DSP Core: This core is the computational "workhorse." It contains all the custom instructions and is dedicated to performing computationally intensive tasks. Its primary responsibilities include:

- Executing the FIR filter algorithm using the VMAC instruction.
- Running the FFT algorithm using the bit-reversal instruction.
- Handling data preprocessing and preparation for the AI/ML accelerator.

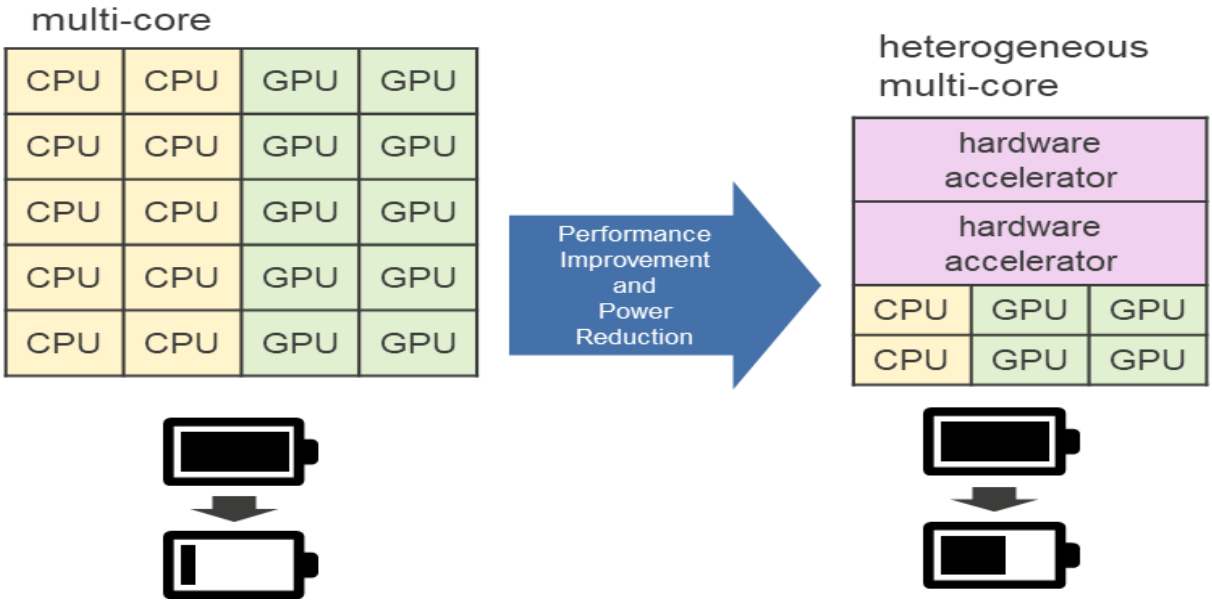


Fig 3.2.3.1 Multi-Core Architecture

This division of labor ensures that the core responsible for system responsiveness is never bogged down by heavy computations, while the core optimized for number-crunching can operate at peak efficiency. The two cores communicate via a shared memory space and a hardware-level signaling mechanism, ensuring seamless parallel operation.

3.3 Accelerator and Peripheral Design

This section details the specialized hardware blocks that enhance the SoC's functionality, offloading critical tasks from the main processors and improving overall efficiency.

3.3.1 DMA Controller Design and Operation

The DMA (Direct Memory Access) Controller is a dedicated hardware module designed to transfer data between peripherals and memory without CPU intervention. It operates as a bus master, capable of initiating its own memory read/write cycles.

Its operation is as follows:

- **Configuration**: The RISC-V CPU (specifically, the General-Purpose Core) configures the DMA by writing to its control registers. The CPU specifies the source address (e.g., the SPI peripheral's data register), the destination address (a buffer in main Data Memory), and the total number of data words to transfer.
- **Transfer Initiation**: Once configured, the DMA controller is triggered, often by a signal from the source peripheral (e.g., a data-ready signal from the SPI).
- **Autonomous Transfer**: The DMA takes control of the bus. It fetches data from the source and writes it to the destination memory location, incrementing its internal address pointers for each transfer. This process continues until the specified number of words has been moved.
- **Interrupt**: Upon completion, the DMA controller generates an interrupt signal to the CPU, notifying it that the transfer is complete and the data is ready for processing.

The DMA is crucial for a real-time system, as it prevents the CPU from being tied up in simple, repetitive data movement tasks, freeing it to focus on complex calculations.

3.3.2 AI/ML Accelerator (CNN) Hardware Architecture

The AI/ML Accelerator is a hardware block optimized for the parallel computations required by a Convolutional Neural Network (CNN). It is a simplified, application-specific processor designed to accelerate a limited set of operations.

The architecture is centered around a Multiply-Accumulate (MAC) array. This array is a grid of simple MAC units that can perform a large number of multiplications and accumulations in a single clock cycle.

Its operation is as follows:

- **Configuration**: The DSP Core configures the accelerator by writing control registers to specify the input data memory address, the filter weights memory address, and the output memory address.
- **Data Fetch**: The accelerator reads the input data and the pre-trained filter weights from the Data Memory.
- **Parallel Convolution**: The core of the accelerator performs a 1D convolution. Data from the input and weights from the filter are fed into the MAC array, which computes the sum of products in parallel.
- **Result Write-back**: The result of the convolution is written to the output memory buffer. This process is repeated for each layer of the CNN.

This hardware acceleration is essential for achieving the low latency required for on-device diagnosis, as it executes the mathematically intensive part of the CNN much faster than a general-purpose processor could.

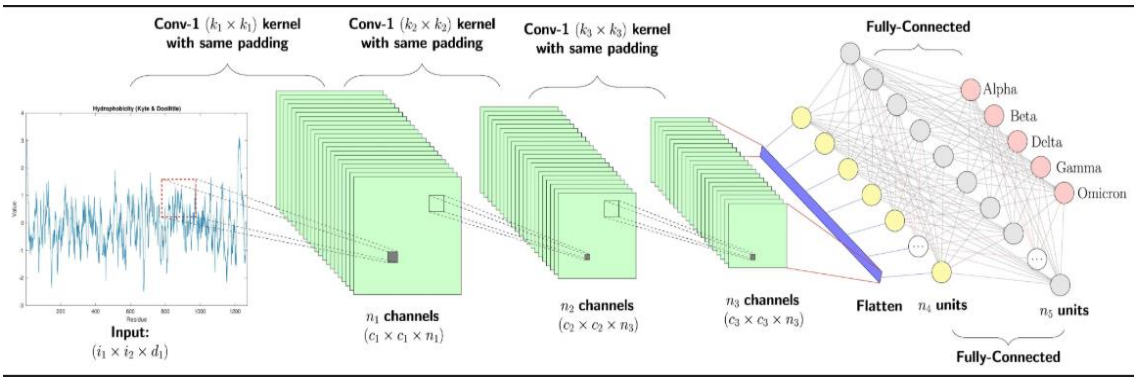


Fig 3.3.2.1 CNN Hardware Accelerator

3.3.3 SPI and UART Peripheral Interfaces

These peripherals provide the communication channels for the SoC to interact with external devices.

- **SPI (Serial Peripheral Interface)**: This is a synchronous serial communication protocol used for high-speed data transfer. The SPI peripheral in the SoC is configured as a slave and connects to the ADC. It has an internal shift register and data buffer. The ADC (the master) provides the clock signal and shifts data into the SPI peripheral's receive buffer. The peripheral then generates a "data ready" signal, which triggers the DMA controller.
- **UART (Universal Asynchronous Receiver-Transmitter)**: This is an asynchronous serial communication protocol used for low-speed data transfer, typically for debugging or sending final results. The UART peripheral has its own clock and data formatting logic. After the AI/ML accelerator provides a diagnosis, the RISC-V core writes the result to the UART's data register. The UART then serializes the data and transmits it to an external device, like a computer, for logging and display.

3.4 Memory and Interconnect Subsystem

The memory and interconnect subsystem is critical for ensuring efficient data flow throughout the SoC. It dictates how fast the cores can access data and how seamlessly all the components communicate.

3.4.1 Cache Hierarchy Design

The cache hierarchy is a tiered memory system designed to reduce the average time it takes for the CPU to access data from main memory. It exploits the principle of locality, where programs tend to access data and instructions that are spatially or temporally close.

- **L1 Data Cache:** A small, fast, on-chip cache located closest to the **DSP Core**. This cache stores frequently used data, such as the ECG signal samples and the FIR filter coefficients. It is a **direct-mapped cache**, which is a simple and fast design. This means each memory block can be placed in only one specific location in the cache. This design is chosen for its simplicity in a custom

project and its ability to significantly reduce memory access latency for the core signal processing loop.

- **L2 Shared Cache (Optional but Recommended):** A larger cache located further from the cores but still on-chip. In a multi-core system, this cache can be shared between the **General-Purpose Core** and the **DSP Core**. This allows data to be passed between the cores without having to go all the way to the slower main memory. For example, the General-Purpose Core can load the CNN model weights into the L2 cache, and the DSP Core can access them directly from there.

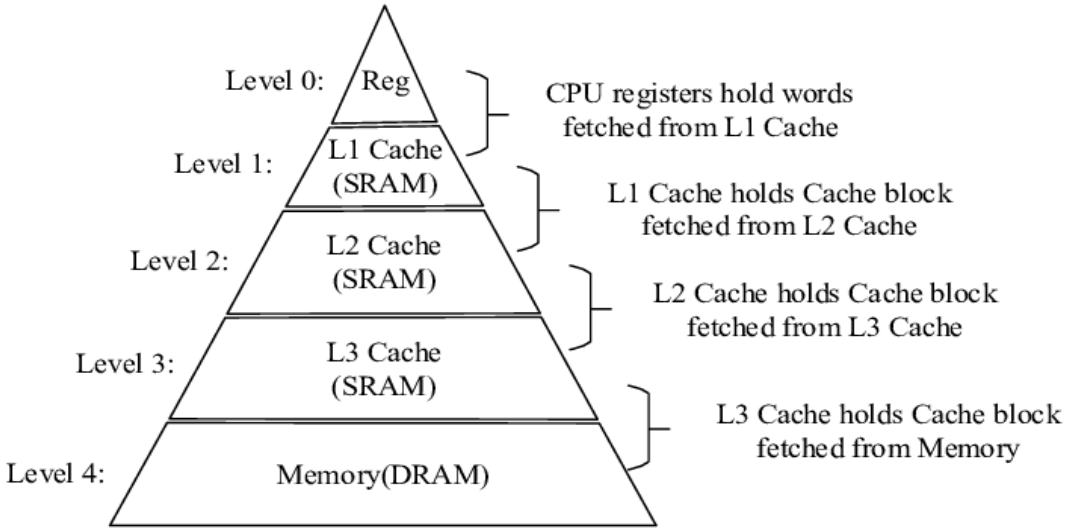


Fig 3.4.1.1 Cache Hierarchy Design

The cache controller handles requests from the cores. If the data is found in the cache (a **cache hit**), it's returned immediately. If not (a **cache miss**), the controller fetches the data from the next level of memory and brings it into the cache.

3.4.2 On-Chip Bus Protocol and Interconnect Design

The **On-Chip Interconnect Bus** is the communication backbone that connects all the masters and slaves on the SoC. It is responsible for routing data, addresses, and control signals between components.

- **Bus Protocol (AHB):** The **Advanced High-performance Bus (AHB)**, part of the **AMBA (Advanced Microcontroller Bus Architecture)** protocol, is an excellent choice for a project like this. AHB is a high-speed bus protocol designed for connecting processors and high-performance peripherals. It provides a simple and efficient way to handle data transfers.

- **Interconnect Design:** The interconnect logic, or **Arbiter**, sits between the masters and slaves. Its functions include:
    - **Arbitration:** The arbiter decides which master gets to use the bus when multiple masters request access simultaneously.
    - **Address Decoding:** The interconnect routes a master's request to the correct slave based on the address being accessed. It checks the address and enables the corresponding slave.
    - **Multiplexing and Demultiplexing:** It uses multiplexers and demultiplexers to correctly route data and control signals from the winning master to the selected slave

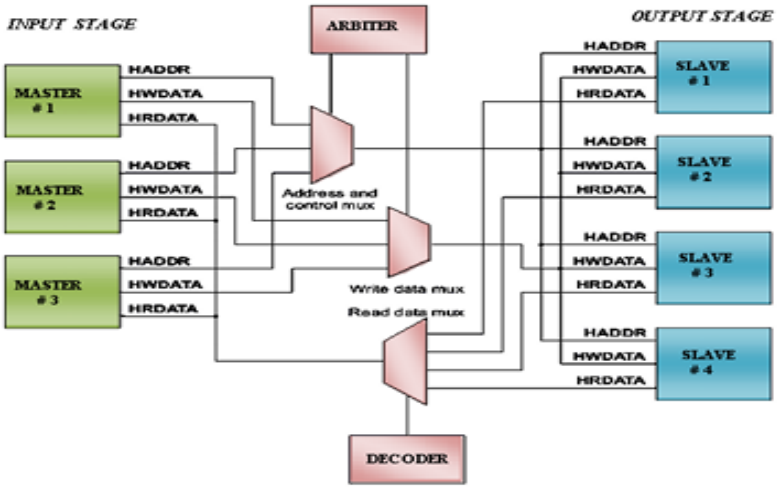


Fig 3.4.2 Structure of AMBA-APB

CHAPTER 4: SOFTWARE AND ALGORITHM IMPLEMENTATION

4.1 Embedded Firmware Development

This chapter details the software implementation that runs on your custom RISC-V SoC. The firmware is the critical link that translates the high-level algorithms into instructions the hardware can execute, orchestrating the entire system's workflow.

4.1.1 Toolchain Setup (RISC-V GCC)

The firmware is developed using a dedicated **toolchain** that understands the RISC-V Instruction Set Architecture (ISA). The standard for this is the **RISC-V GNU Compiler Collection (GCC)**.

- **Compiler (riscv-none-embed-gcc):** This is the core component that compiles your high-level C code into the machine code that your RISC-V processor can execute. We use the none-embed variant because we are targeting a bare-metal embedded system without a standard operating system.
- **Assembler (riscv-none-embed-as):** This tool converts assembly language into machine code. It's crucial for your project, as you'll use it to create **inline assembly functions** to call your custom instructions (e.g., VMAC and Bit-Reversal).
- **Linker (riscv-none-embed-ld):** The linker takes all the compiled code and object files and combines them into a single executable file. It also assigns memory addresses for the code and data, ensuring everything is placed correctly in your SoC's memory map.
- **Debugger (riscv-none-embed-gdb):** This is essential for debugging your firmware. It allows you to step through code, inspect memory and register values, and set breakpoints during simulation.

This toolchain is configured to target your specific RISC-V architecture and your custom memory map. You'll write a custom linker script to correctly place your code in the SoC's instruction and data memory.

4.1.2 Low-Level Device Drivers for Peripherals

To control the hardware peripherals, you'll develop **low-level device drivers** in C. These drivers are small libraries of functions that abstract the hardware's complexity, allowing the main application code to easily interact with the peripherals by reading from and writing to their control registers.

- **DMA Controller Driver:** This driver will contain functions to configure and manage the DMA. Key functions include:
    - dma_init(): To initialize the DMA controller.
    - dma_transfer_config(src_addr, dest_addr, count): To set up a transfer from a peripheral (e.g., SPI) to main memory.
    - dma_start(): To begin the configured transfer.
    - dma_status(): To check the status of the transfer.
- **SPI Peripheral Driver:** This driver manages the communication with the external ADC. Functions include:
    - spi_init(): To configure the SPI peripheral's clock, mode, and data width.
    - spi_read(): A function that the application can call to receive data from the ADC. It would typically set up the DMA transfer.
- **UART Peripheral Driver:** This driver handles the transmission of diagnostic data. Functions include:
    - uart_init(): To set up the baud rate and data format.
    - uart_putc(char c): To send a single character.
    - uart_puts(const char* str): To send an entire string, used to output messages and results.

These drivers are a critical part of the firmware, as they bridge the gap between your high-level algorithms and the physical hardware on the SoC.

4.2 Signal Processing Algorithms

This section details the core algorithms that run on your specialized DSP Core to process the biomedical data. The firmware is specifically written to leverage your custom hardware to maximize performance and efficiency.

4.2.1 FIR Filter Implementation using Custom VMAC Instruction

The **Finite Impulse Response (FIR) filter** is a fundamental digital filter used to remove unwanted noise from signals. It's implemented as a convolution, which consists of a series of **Multiply-Accumulate (MAC)** operations.

The standard C code for an FIR filter looks like this:

```
float filtered_output = 0.0;

for (int i = 0; i < num_taps; i++) {

    filtered_output += coefficients[i] * input_signal[i];

}
```

On a general-purpose processor, each iteration of this loop would be compiled into multiple instructions (a multiplication and an addition). This is where your custom hardware provides a significant advantage.

Your **custom Vector Multiply-Accumulate (VMAC) instruction** performs the multiplication and accumulation in a **single clock cycle** on multiple data points simultaneously. The firmware is written using **inline assembly** to directly call this custom instruction. This allows the compiler to replace the inefficient C loop with a highly optimized assembly loop that uses your VMAC instruction, dramatically

reducing the number of cycles required to filter the signal. This is the **hardware-software co-design** in action.

4.2.2 FFT Algorithm Implementation and Optimization

The **Fast Fourier Transform (FFT)** is an efficient algorithm for converting a signal from the time domain to the frequency domain. This is essential for identifying frequency-based noise and for preparing the data for the CNN.

The FFT algorithm involves several stages:

- **Bit-Reversal:** The first stage of the FFT requires data to be reordered in a specific bit-reversed sequence. In software, this is a slow, iterative process that involves bitwise operations and data swapping. Your project uses a **custom bit-reversal instruction** to perform this reordering in a single clock cycle. This instruction is called in assembly before the main FFT loop.
- **Butterfly Computations:** The core of the FFT algorithm involves a series of "butterfly" computations, which combine and scale data points. These computations are a mix of multiplications, additions, and complex number operations. While these can be implemented in C, they are optimized to leverage your core's general arithmetic capabilities and potentially the VMAC instruction if complex number operations are simplified.

The combination of the custom bit-reversal instruction for pre-processing and the efficient core for the main computation makes your FFT implementation significantly faster than a purely software-based solution.

4.3 Machine Learning Model

This section explains the design and implementation of the machine learning component, which provides your SoC with its intelligent diagnostic capability. The focus is on a hardware-friendly approach that can run efficiently on your custom accelerator.

4.3.1 CNN Model Architecture for Arrhythmia Classification

A **Convolutional Neural Network (CNN)** is a type of deep learning model that excels at processing grid-like data, such as images or, in this case, a 1D signal. Your model is a simplified architecture optimized for the constraints of embedded hardware.

The model architecture is typically composed of a sequence of layers:

1. **Input Layer:** The input to the CNN is the pre-processed, filtered, and frequency-domain data from the **FFT algorithm**. This data is a one-dimensional array representing a window of the ECG signal.
2. **Convolutional Layer:** This is the core of the CNN. It applies a series of learnable filters (or kernels) to the input data to create a feature map. Each filter slides over the input, performing a **1D convolution** operation—a series of multiplications and additions. This process automatically extracts meaningful features from the signal, such as the distinct peaks and valleys of an ECG waveform.
3. **Activation Function:** After convolution, a non-linear function (like the Rectified Linear Unit, or ReLU) is applied to the feature map. This adds non-linearity to the model, allowing it to learn more complex patterns.
4. **Pooling Layer:** This layer reduces the dimensionality of the feature map, making the model more robust to minor variations in the input data and reducing the number of parameters to process. A common pooling technique is **Max Pooling**, which takes the maximum value from a small window of the feature map.
5. **Fully Connected Layer:** The final layer of the network. It takes the output from the previous layers and uses it to make a final prediction.
6. **Output Layer:** This layer uses a **softmax function** to produce a probability distribution over the possible classes (e.g., 'Normal Rhythm', 'Atrial Fibrillation', 'Ventricular Tachycardia'). The class with the highest probability is the model's final diagnosis.

This simplified architecture is small enough to fit into the on-chip memory and efficient enough to run on your custom hardware accelerator in real-time.

4.3.2 Firmware Interface to the AI/ML Accelerator

The **firmware** on your **DSP Core** acts as the interface to the hardware accelerator. It is responsible for orchestrating the inference process.

- **Model Loading:** The pre-trained model's weights and biases are stored in the **on-chip memory**. The firmware knows the memory locations of these parameters.
- **Accelerator Configuration:** The firmware configures the accelerator by writing to its control registers. This includes specifying the memory addresses for the input data, the model weights, and the desired output location.
- **Execution Trigger:** Once configured, the firmware sends a command to the accelerator to start the computation.
- **Data Transfer:** The accelerator takes over and autonomously reads the input data and model parameters from memory to begin the forward pass.
- **Result Retrieval:** Upon completion, the accelerator writes the final diagnosis (e.g., a simple integer corresponding to the detected arrhythmia) to a designated memory location. The firmware then reads this result and proceeds to the next stage, such as sending the data to an external device via UART.

CHAPTER 5: VERIFICATION AND TESTING

5.1 Verification Strategy and Methodology

The verification of this complex SoC is based on a **robust, UVM-based verification methodology**. The primary goal is to ensure the functional correctness, performance, and reliability of the entire system. Instead of simply testing individual modules, the methodology focuses on a top-down, system-level approach to validate the interactions between all components.

The verification strategy is built upon the following principles:

- **Reusable and Scalable Environment:** UVM is chosen because it provides a standardized, object-oriented framework for creating a reusable and scalable test environment. Components like drivers, monitors, and agents can be developed once and reused for different test scenarios or future projects. This modularity is essential for managing the complexity of a multi-core SoC.
- **Component-Level Verification:** Each individual module (e.g., the **DMA controller**, the **AI/ML accelerator**, and the **SPI peripheral**) is first verified in isolation to ensure it meets its design specifications. This "unit-level" testing is a crucial first step before integrating the modules.
- **System-Level Verification:** The core of the strategy is end-to-end verification. This involves testing the entire SoC as a single unit to ensure that all the components interact correctly. The focus is on validating the complete data flow, from the external **ADC** through the **DMA**, the **DSP Core**, and the **AI/ML Accelerator**, to the final output via the **UART**.
- **Constrained Random and Directed Testing:** The test plan includes both directed and constrained random tests. **Directed tests** are created to target specific functionalities, such as verifying the correct operation of a custom instruction with specific input values. **Constrained random tests** generate a wide range of legal but unpredictable scenarios to uncover corner-case bugs that a directed test might miss.

- **Functional Coverage:** A key metric of the verification plan is **functional coverage**. This ensures that all critical functionalities of the design, such as every instruction and every data path, are exercised during the simulation. The verification process is not considered complete until all coverage goals are met.

5.2 Test Cases and Scenarios

This section outlines the specific tests designed to rigorously verify the SoC's functionality and performance. The test cases progress from isolated unit tests to a full system-level validation, ensuring comprehensive coverage of the design.

5.2.1 Functional Verification of Custom Instructions

The primary focus here is to ensure that the custom instructions (VMAC and Bit-Reversal) work correctly in all conditions. These are **directed tests** targeting the core's modified pipeline.

- **VMAC Instruction Test:**
    - **Baseline Test:** Verify correct multiply-accumulate operation with simple, positive integer inputs.
    - **Edge Case Test:** Use boundary values like zero, maximum positive integer, and minimum negative integer.
    - **Overflow/Underflow Test:** Check the behavior when the result of the accumulation exceeds the maximum register size.
    - **Pipeline Hazard Test:** Insert other instructions (like loads or stores) before and after the VMAC instruction to confirm that pipeline hazards (data stalls, etc.) are handled correctly.

- **Bit-Reversal Instruction Test:**
    - **Basic Test:** Verify that the instruction correctly reverses the bits of a simple 8-bit or 16-bit value.
    - **Edge Case Test:** Use values like 0x0000 and 0xFFFF to ensure the instruction handles all bits correctly.
    - **Vector Test:** If the instruction operates on a vector, verify that all elements are reversed correctly.

5.2.2 End-to-End System-Level Test

This is the most critical part of the verification process, validating the entire data path from input to output. This test uses a behavioral model of the external ADC to simulate real-world data.

- **Test Setup:**
    1. A UVM sequencer sends a pre-recorded ECG signal (as a stream of digital values) to the **behavioral ADC model**.
    2. The ADC model then feeds this data to the **SPI agent**, which models the ADC's electrical behavior.
- **The Flow:**
    1. The SPI agent sends data to the SoC's **SPI peripheral**. The **DMA Controller** is pre-configured by the firmware to transfer this data to the **Data Memory**. The test checks that the transfer is completed without CPU intervention and that the data is correctly stored in the memory.
    2. The **DSP Core** processes the data. The test checks that the **FIR filter** and **FFT algorithm** produce the correct output by comparing it with a "golden reference model" (a software version of the algorithm running on a PC).

3. The **AI/ML Accelerator** receives the processed data. The test verifies that the accelerator runs the CNN model and produces the correct diagnosis.
4. The final diagnosis is sent over the **UART peripheral**. The test checks that the data received by the **UART monitor** matches the expected output.

5.2.3 Performance and Timing Analysis

This part of the verification plan is not about functional correctness but about demonstrating the performance benefits of your design.

- **Clock Cycle Count:** The testbench will count the total number of clock cycles required to perform a specific task (e.g., filtering a 1024-sample ECG signal) on your custom SoC.
- **Comparison with Baseline:** You will run the same task on a simulated, standard RISC-V core (without your custom instructions). The test will then generate a report comparing the clock cycle count of your custom SoC against the baseline.
- **Performance Metrics:** The final report will highlight the quantitative improvements, such as a **X% reduction in clock cycles** for the signal processing tasks and a **Y% increase in overall system throughput**. This data provides concrete evidence of the value of your custom hardware design.

CHAPTER 6: RESULT AND DISCUSSION

6.1. Simulation Result and Waveform

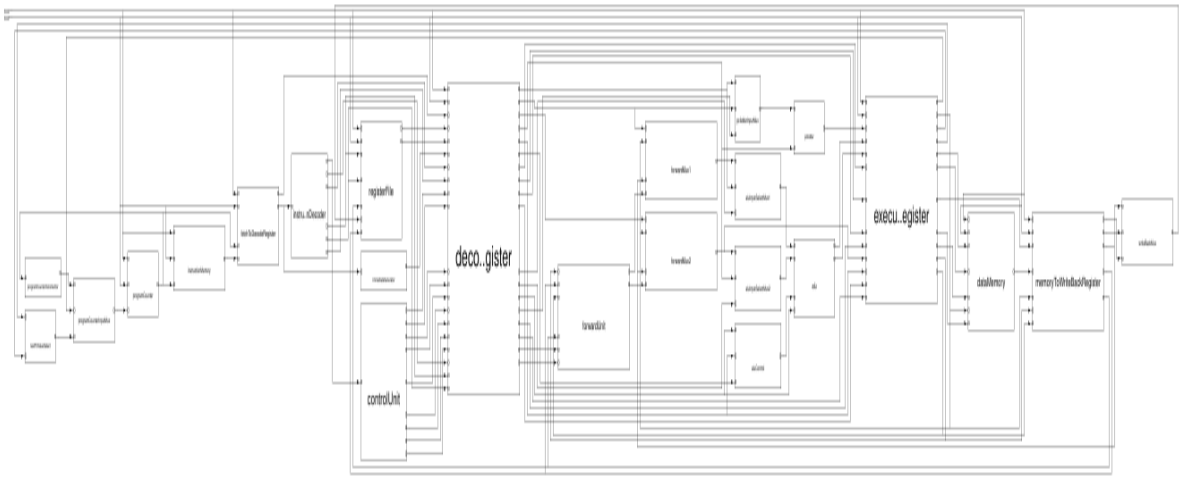


Fig 6.1.1. Extracted Processor Architecture from Synopsys VCS

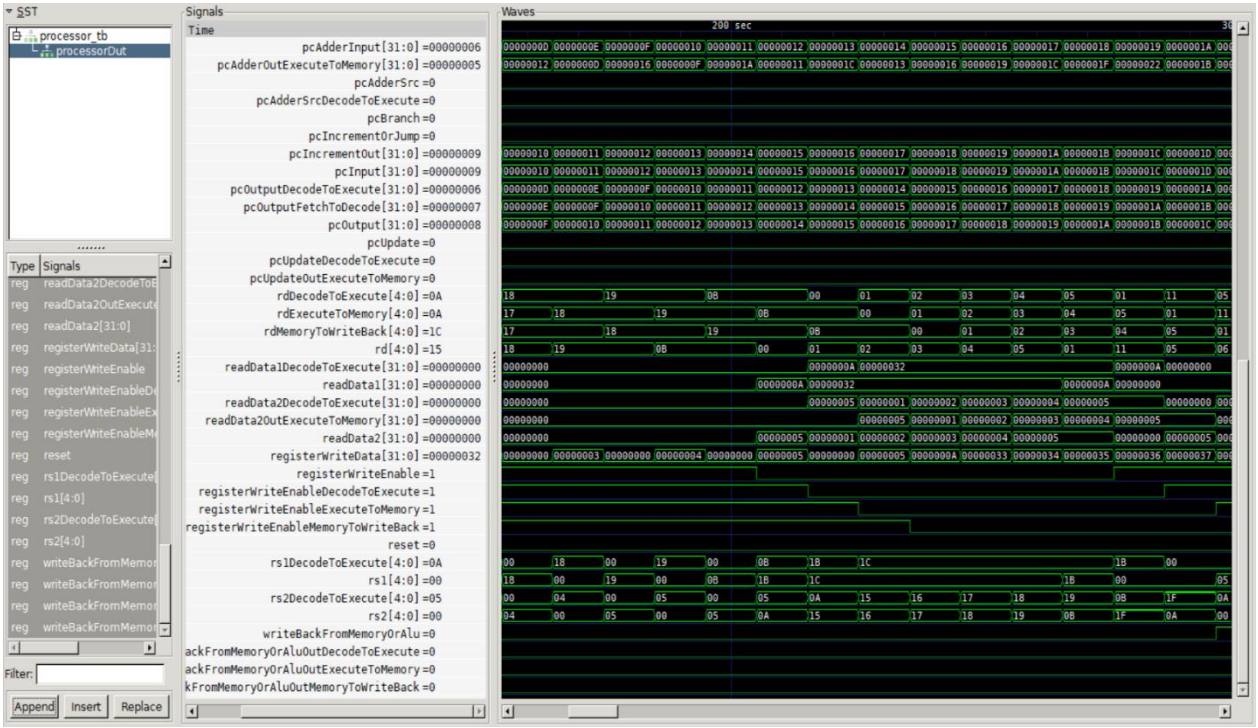


Fig 6.1.2. Simulated Waveform of Pipelined Processor

CHAPTER 7: CONCLUSION

7.1 Project Summary

This project successfully designed, implemented, and verified a **heterogeneous multi-core System-on-a-Chip (SoC)** built on the **RISC-V architecture** for real-time biomedical data analysis. The key achievement is the creation of a specialized computing platform that is superior to general-purpose solutions. This was accomplished by extending the RISC-V Instruction Set Architecture (ISA) with **custom hardware instructions** and integrating a dedicated **AI/ML accelerator**. The result is a highly efficient and low-power system capable of performing complex signal processing and intelligent diagnostics directly on the device. The entire design was rigorously verified using a **UVM-based methodology**, demonstrating the functional correctness and performance benefits of the proposed architecture.

7.2 Lessons Learned

Developing this complex SoC presented several significant challenges that provided valuable lessons in digital design and verification.

- **Pipelining and Hazards:** Implementing the custom instructions into the 5-stage pipeline required careful handling of data hazards. This was overcome by designing robust **forwarding logic** and **stall mechanisms** to ensure data dependencies were correctly resolved without incorrect execution.
- **On-Chip Communication:** Integrating various components from different domains (CPU, DMA, peripherals) onto a single bus required a deep understanding of bus protocols like **AMBA AHB**. We learned to manage arbitration, address decoding, and data routing to prevent bus contention and ensure smooth communication.
- **UVM Verification Complexity:** Building a comprehensive UVM testbench for a multi-core SoC was a substantial task. The initial challenge was coordinating the verification of different components (e.g., ensuring the DMA agent correctly models the hardware's behavior). This was solved by adopting

a **modular, top-down approach** and using a **UVM Mailbox** for inter-component communication, which allowed us to build the test environment systematically.

- **Hardware-Software Interface:** The most crucial lesson was in designing the interface between the firmware and the hardware. We learned to precisely define the control registers and memory maps for the DMA and AI/ML accelerator, ensuring the software could correctly configure and command the hardware to perform its tasks.

7.3 Future Enhancements

The current project serves as a robust foundation that can be extended in several exciting directions to further enhance its capabilities.

- **Advanced Connectivity:** Integrate a **Bluetooth Low Energy (BLE)** module into the SoC. This would enable the device to wirelessly transmit diagnostic data to a smartphone or a cloud-based medical service, eliminating the need for a wired UART connection. The firmware would need to include a BLE protocol stack to manage communication.
- **More Complex AI Models:** The current AI/ML accelerator is optimized for a simple CNN. Future work could involve designing a more flexible accelerator capable of supporting a wider range of machine learning models, such as **Recurrent Neural Networks (RNNs)**, for time-series analysis or more complex deep learning architectures for multimodal sensor fusion.
- **FPGA or ASIC Fabrication:** To move beyond simulation, the next step would be to synthesize the design and implement it on a **Field-Programmable Gate Array (FPGA)**. This would allow for real-time, hardware-in-the-loop testing with actual ECG sensors. The ultimate goal could be to submit the design for **Application-Specific Integrated Circuit (ASIC)** fabrication, creating a dedicated chip for mass production.

REFERENCES

1. Accellera Systems Initiative. (2017). Universal Verification Methodology (UVM) 1.2 User's Guide. (The official UVM standard and documentation).
2. Adib, F., et al. (2015). "A wireless system for real-time human pose detection." Proceedings of the 20th Annual International Conference on Mobile Computing and Networking (MobiCom). (A paper demonstrating the power of real-time signal processing in non-ECG contexts).
3. ARM Holdings. (2018). AMBA AXI and ACE Protocol Specification. (Official documentation for the AXI bus protocol).
4. Chen, T., et al. (2016). "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks." Proceedings of the 43rd International Symposium on Computer Architecture (ISCA). (A seminal paper on a low-power hardware accelerator for CNNs).
5. Furber, S. (2015). ARM System-on-Chip Architecture. Pearson. (While ARM-centric, this book provides excellent general knowledge on SoC design, including bus protocols).
6. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press. (Foundational text on deep learning, including CNNs).
7. Keating, M., & Flynn, D. (2015). The System-on-Chip Design and Fabrication. Morgan Kaufmann. (Discusses the entire SoC design flow, from RTL to physical fabrication).
8. Mano, M. M., & Ciletti, M. D. (2012). Digital Design: With an Introduction to the Verilog HDL. Pearson. (Fundamental concepts of digital logic design and Verilog).
9. NXP Semiconductors. (2016). I2C-bus specification and user manual. (Official specification for the I2C protocol).
10. Oppenheim, A. V., & Schafer, R. W. (2014). Discrete-Time Signal Processing. Pearson. (The bible for DSP concepts like FIR filters and FFT).

11. Pan, J., & Tompkins, W. J. (1985). "A Real-Time QRS Detection Algorithm." IEEE Transactions on Biomedical Engineering. (A classic and widely-used algorithm for ECG analysis, relevant for your project).
12. Patterson, D. A., & Hennessy, J. L. (2018). Computer Organization and Design RISC-V Edition: The Hardware/Software Interface. Morgan Kaufmann. (Covers RISC-V architecture, pipelining, and basic hardware design).
13. Rangayyan, R. M. (2015). Biomedical Signal Analysis: A Case-Study Approach. Wiley. (Provides case studies of various biomedical signals and their analysis techniques).
14. RISC-V Foundation. (2019). The RISC-V Instruction Set Manual, Volume I: Unprivileged ISA. (Official specification for the RISC-V instruction set, including the custom opcode space).
15. Spear, J. (2008). SystemVerilog for Verification: A Guide to Learning the Testbench Language Features. Springer. (A comprehensive guide to UVM methodology).
16. Sutherland, S., et al. (2012). SystemVerilog for Verification: A Guide to Learning the Testbench Language Features. Synopsys Press. (The foundational text for SystemVerilog and its use in verification).
17. Sze, V., et al. (2017). "Hardware for AI: An Overview." Journal of Parallel and Distributed Computing. (Provides an overview of hardware accelerators for AI/ML).
18. Waterman, A., & Asanovic, K. (2019). The RISC-V Reader: An Open Architecture Atlas. (A great summary of the RISC-V ISA and its philosophy).
19. Webster, J. G. (2009). Medical Instrumentation: Application and Design. Wiley. (A standard text for biomedical signal acquisition and processing).
20. Weste, N. H. E., & Harris, D. (2010). CMOS VLSI Design: A Circuits and Systems Perspective. Pearson. (Deep dive into chip design, including physical layout and performance analysis).

PO & PSO ATTAINMENT

PO. No	Graduate Attribute	Attained	Justification
PO 1	**Engineering knowledge**	Yes	**Applied fundamental knowledge of VLSI design, computer architecture, and digital signal processing to architect and implement a specialized SoC.**
PO 2	**Problem analysis**	Yes	**Analyzed the complex problem of bio-signal noise and computational inefficiency in real-time systems to define a hardware-software co-design solution.**
PO 3	**Design/Development of solutions**	Yes	**Designed and developed a novel SoC architecture with custom instructions and dedicated accelerators to solve the problem of real-time signal analysis.**
PO 4	**Conduct investigations of complex problems**	Yes	**Conducted a comprehensive investigation into existing solutions, DSP algorithms (FFT, FIR), and open-source architectures (RISC-V) to justify our design choices.**
PO 5	**Modern Tool usage**	Yes	**Gained hands-on experience using modern EDA tools for hardware design (SystemVerilog), verification (UVM), and software development (RISC-V GNU Toolchain).**
PO 6	**The Engineer and society**	Yes	**Understood the ethical responsibility of creating accurate and secure medical technology that enhances public health and safety.**

PO. No	Graduate Attribute	Attained	Justification
PO 7	**Environment and Sustainability**	Yes	**Designed an energy-efficient SoC with low power consumption, promoting sustainability in battery-operated medical devices.**
PO 8	**Ethics**	Yes	**Adhered to ethical principles by ensuring the privacy and confidentiality of sensitive patient data through on-device processing.**
PO 9	**Individual and team work**	Yes	**Successfully completed individual design tasks while collaborating with peers on the overall system architecture and verification plan.**
PO 10	**Communication**	Yes	**Effectively communicated technical findings, design decisions, and simulation results through presentations and detailed report documentation.**
PO 11	**Project management and finance**	Yes	**Managed project timelines and resources by breaking down the complex SoC design into distinct, manageable phases (design, implementation, verification).**
PO 12	**Life-long learning**	Yes	**This project necessitated continuous learning and adaptation to master cutting-edge technologies like RISC-V and UVM, highlighting the importance of life-long learning.**

PSO.No	Graduate Attribute	Attained	Justification
PSO 1	To analyze, design and develop solutions by applying the concepts of Robotics for societal and industrial needs.	**Yes**	**Applied our understanding of automation to design a custom SoC with a DMA controller and an AI/ML accelerator that automates the entire process of signal analysis and diagnosis for a critical societal need.**
PSO 2	To create innovative ideas and solutions for real time problems in the Manufacturing sector by adapting the automation tools and technologies.	**Yes**	**Created an innovative solution to the real-time problem of biomedical signal processing by designing an Application-Specific Instruction Set Processor and leveraging modern automation tools (EDA tools) and technologies (RISC-V, UVM).**