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

| | | |
|---|---|---|
| **Presented by :** | BALAJI S | 22EC019 |
| | MANIKANDAN L | 22EC070 |

**Guided By :** Ms.SUBBULAKSHMI M.E.,

## PROBLEM DEFINITION

Current systems for real-time biomedical data analysis are inefficient and pose security risks.

- **Computational Inefficiency:** General-purpose processors (GPPs) are slow and power-hungry for crucial tasks like signal filtering, causing unacceptable delays in real-time applications.

- **Latency & Security:** Relying on the cloud for data processing introduces delays and exposes sensitive patient data to security risks.

- **Fixed Hardware:** Existing solutions use pre-built processors that can't be modified. This prevents us from creating specialized hardware for our specific needs.

- **Inaccurate Diagnosis:** Calculation errors in bio-signals often occur due to the inefficient and incomplete removal of noise, affecting diagnostic accuracy.

# EXISTING SOLUTION

- **General-Purpose Microcontrollers**: These are not designed for computationally intensive tasks, making them slow and power-hungry for complex signal processing.

- **Off-the-Shelf DSP Processors**: DSP processors are fast, but they have a fixed Instruction Set Architecture (ISA) that cannot be modified with application-specific instructions.

- **Cloud-Based Solutions**: They offer powerful processing but introduce unacceptable latency and significant privacy risks for sensitive data.

- **Existing Hardware Accelerators**: These often focus on a single task, such as a CNN accelerator, but are not integrated into a complete System-on-Chip (SoC) that also includes general processing cores and peripherals.

# LITERATURE REVIEW

| S.No | Paper Title/Year | Authors name | Discussion |
|---|---|---|---|
| 1. | A Real-Time QRS Detection Algorithm (1985) | J. Pan & W.J. Tompkins | This classic paper presents a real-time, software-based algorithm for detecting QRS complexes in ECG signals. While influential, it relies on a general-purpose processor computational power. The algorithm is effective for its time but would be inefficient and power-hungry for continuous monitoring on a battery-powered device compared to a hardware-accelerated solution. |
| 2. | Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks (2016) | V. Sze, et al. | This work proposes a specialized hardware accelerator for CNNs. The discussion focuses on a dedicated architecture for deep learning. However, it is a standalone accelerator, not a complete SoC with a general-purpose core, DSP capabilities, and peripherals. Your project combines these elements, offering a more complete solution. |

# LITERATURE REVIEW

| S.No | Paper Title/Year | Authors name | Discussion |
|---|---|---|---|
| 3. | ARM System-on-Chip Architecture (2015) | S. Furber | This book, while a key reference for SoC design, discusses a commercial, off-the-shelf architecture (ARM). ARM cores are powerful but have a fixed Instruction Set Architecture (ISA). Your project's unique value is its custom, extensible RISC-V ISA, which allows for application-specific instructions and a deeper level of hardware-software co-design. |
| 4. | A wireless system for real-time human pose detection" (2015) | F. Adib, et al. | This paper demonstrates the use of a system for real-time signal processing in a non-ECG context. While it highlights the need for real-time processing, the hardware solution is often based on standard chips. Your project's core innovation is the custom silicon design, which provides a level of optimization that cannot be achieved with off-the-shelf components. |

# OBSERVATIONS FROM EXISTING METHODOLOGIES

- **Hardware-Level Optimization:** We must use custom hardware to solve the inefficiency of general-purpose processors.

- **Noise Removal combined with Accelerated Analysis:** A single platform is needed that can both clean the raw signal and then run advanced analysis on it at high speed.

- **Full System Integration:** The entire system—including processing, memory, and peripherals—must be integrated on a single chip to reduce latency and power consumption.

# PROPOSED SOLUTION

- A custom **System-on-a-Chip (SoC)** built on a **heterogeneous multi-core RISC-V architecture**.

- **Dual-Core Design:**

  **General-Purpose Core:** Manages system control and peripherals.

  **Dedicated DSP Core:** Handles all intensive signal processing tasks.

- **Hardware Acceleration:**
  - **Custom Instructions:** Our DSP core features a **Vector Multiply-Accumulate (VMAC)** instruction for ultra-fast filtering.
  - **Dedicated Accelerators:** A **DMA controller** automates data transfer, and a specialized **AI/ML accelerator** handles on-device CNN analysis.

# FIR FILTER (V-MAC)

- Raw ECG signals are too noisy for accurate analysis. We need to remove noise efficiently.
- We use a **FIR filter** because it's stable and effective at removing noise.
- We designed a custom **V-MAC instruction** for our processor. This instruction is the key to accelerating the FIR filter.
- The V-MAC instruction makes our SoC significantly **faster** and more **power-efficient** than a standard processor for this specific task.

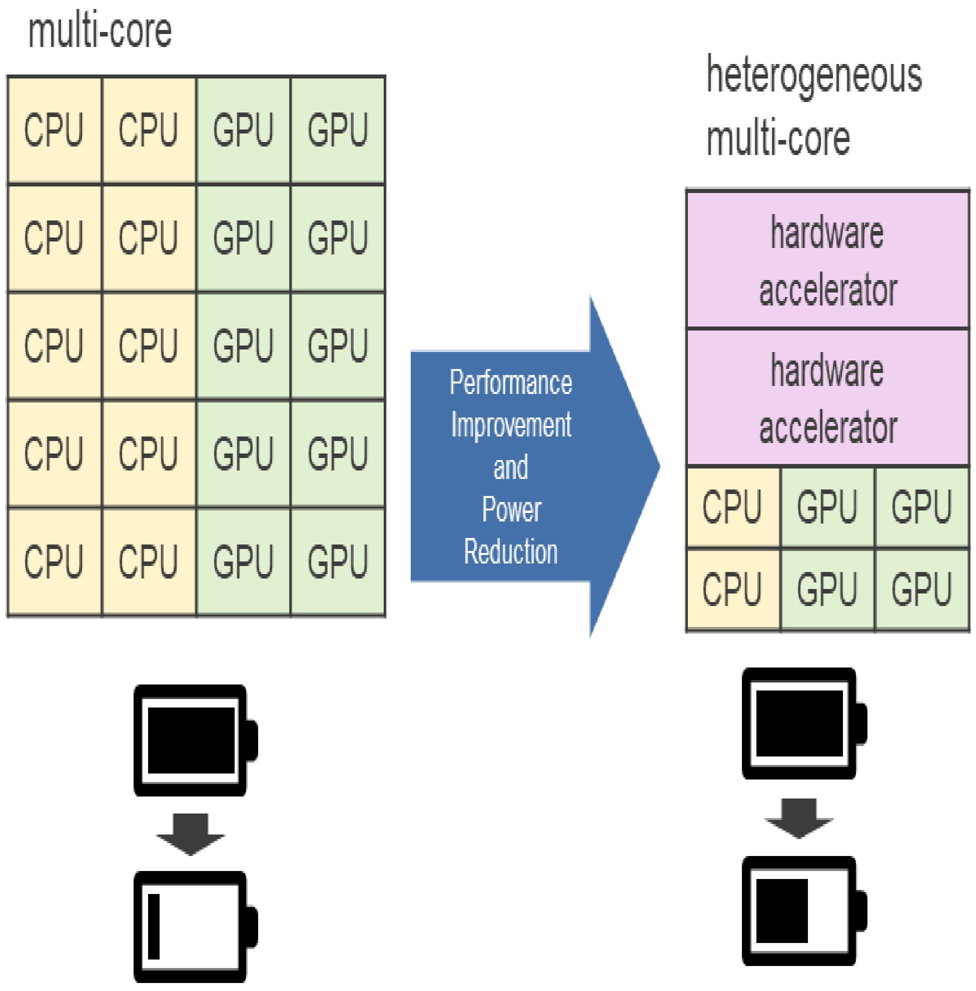
## FAST FOURIER TRANSFORM (FFT)

- ECG signals contain hidden noise frequencies that are hard to remove with simple filtering.
- We use **FFT** to analyze the signal's frequency content and precisely locate noise.
- We designed a **custom Bit-Reversal instruction** to accelerate the FFT algorithm.
- Our SoC can perform frequency analysis and prepare data for AI with greater **speed** and **efficiency** than a standard processor.

## HARDWARE ACCELERATOR(CNN)

- It performs automated diagnosis by running a pre-trained **CNN** model on the data.
- It accelerates the core mathematical computations of the **CNN**, such as convolutions, far more efficiently than a CPU.
- It enables **on-device intelligence**, allowing the system to make a diagnosis in real time without a network connection.
- It ensures **data privacy and security** by processing all sensitive information locally on the chip.

# HETEROGENEOUS MULTI-CORE

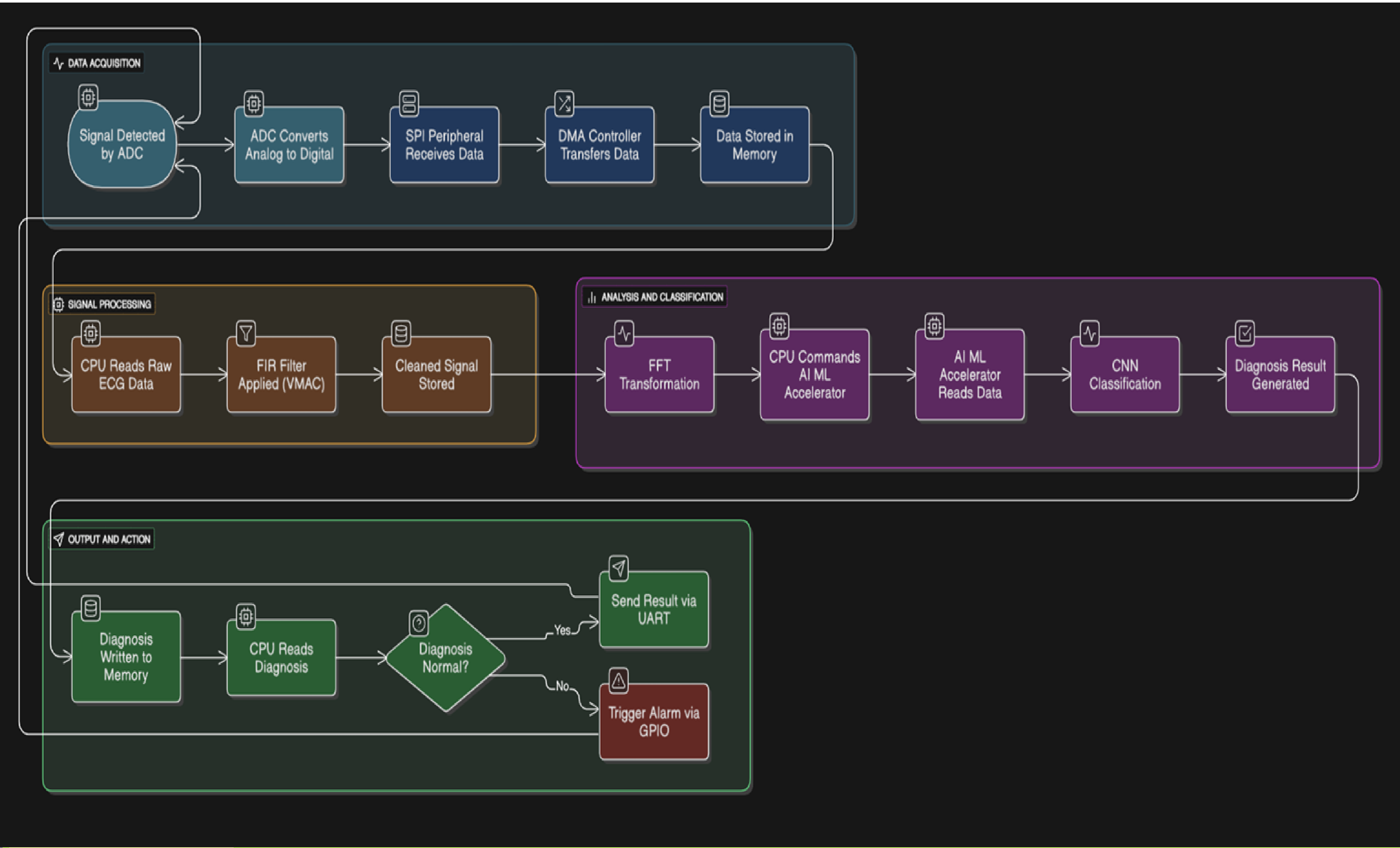


The heterogeneous multi-core architecture is designed to divide labor between two specialized cores.

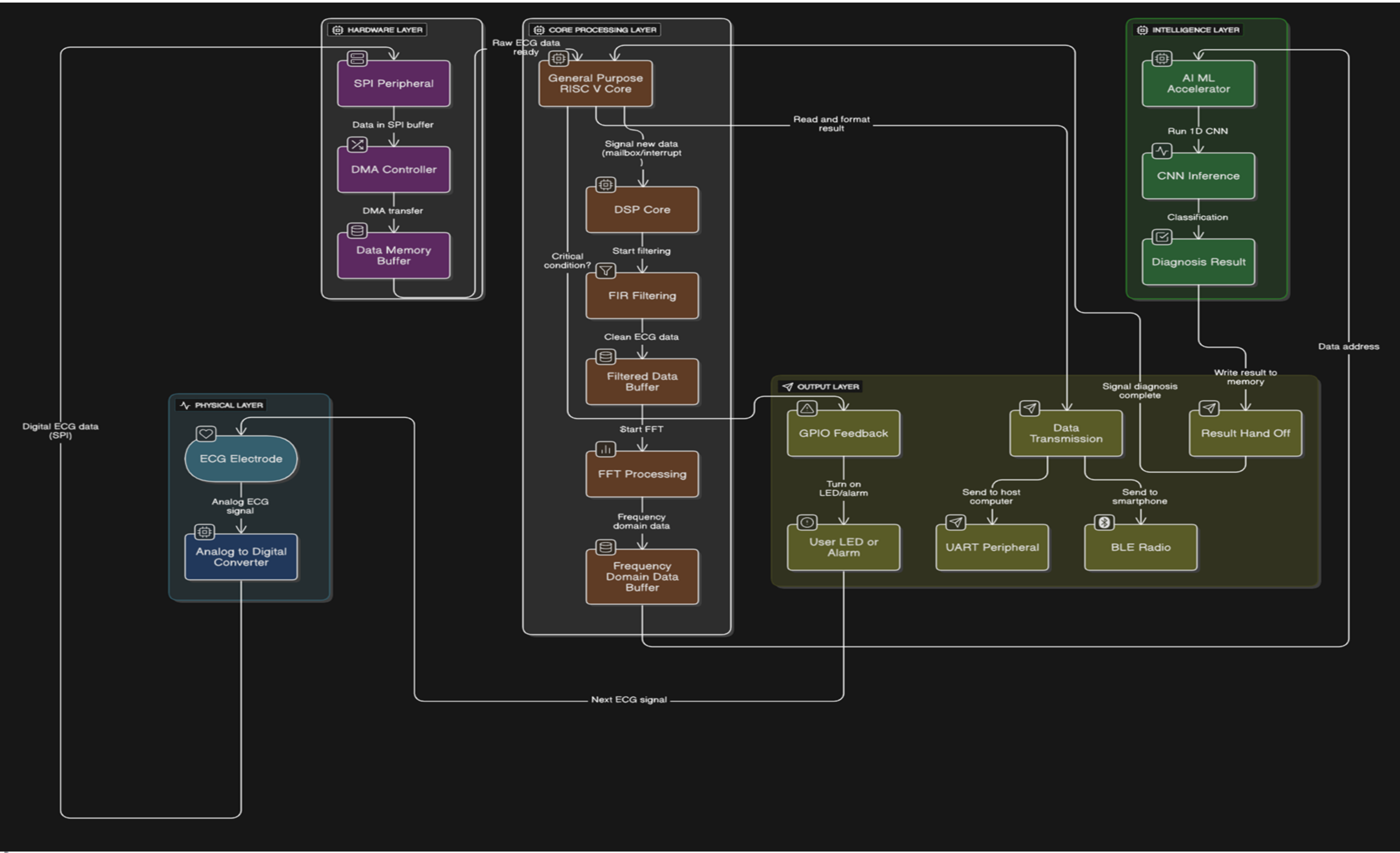
- **General-Purpose Core**:Manages all high-level system control and peripherals.
- **DSP Core**:Handles the intensive signal processing and data analysis tasks.

This division ensures that the system is both highly responsive and extremely efficient at its core functions.

# PROPOSED METHODOLOGY OVERVIEW



# SOC EXECUTION LAYERS



# COMPONENTS REQUIRED:

**Hardware Design Tools:**

- **HDL Language:** SystemVerilog
- **HDL Simulator:** Synopsys VCS & Verdi
- **Synthesis Tool:** Synopsys Design Compiler

**Software Development Tools:**

- **RISC-V GCC Toolchain**

**Verification Environment:**

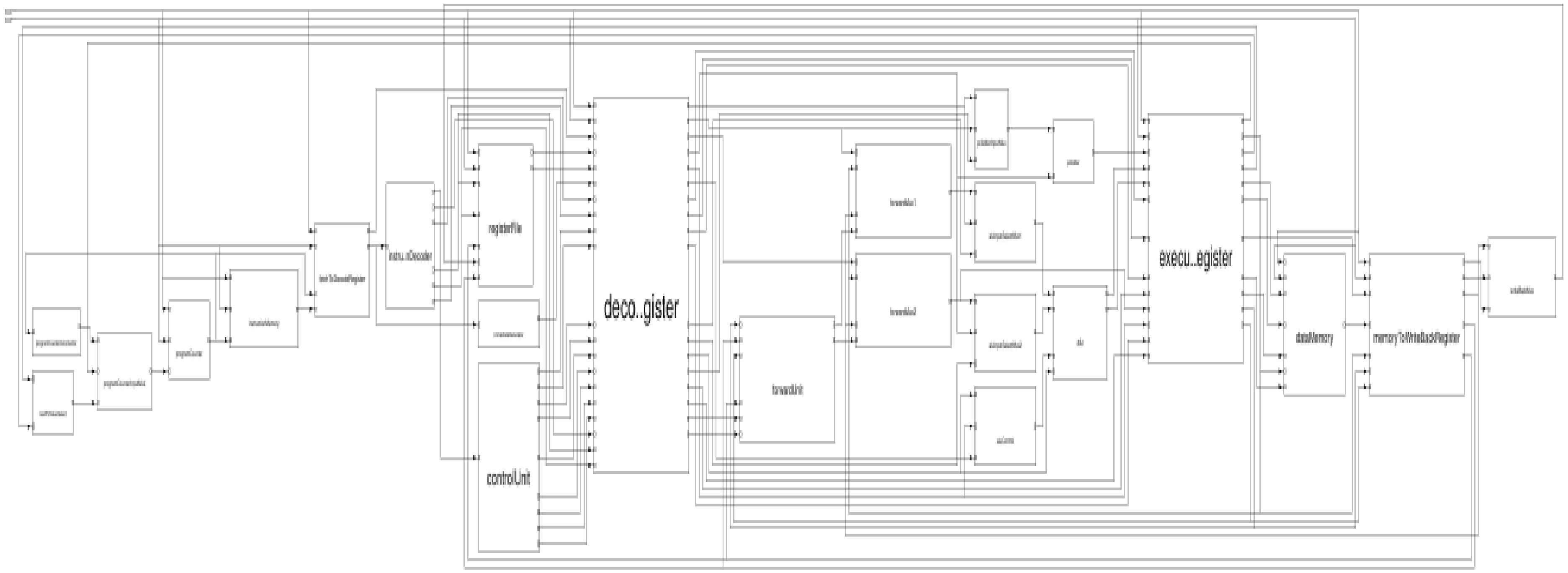
- **UVM (Universal Verification Methodology)**
- **Assertions and Coverages**

## PROGRESS

**Current Project Status:**

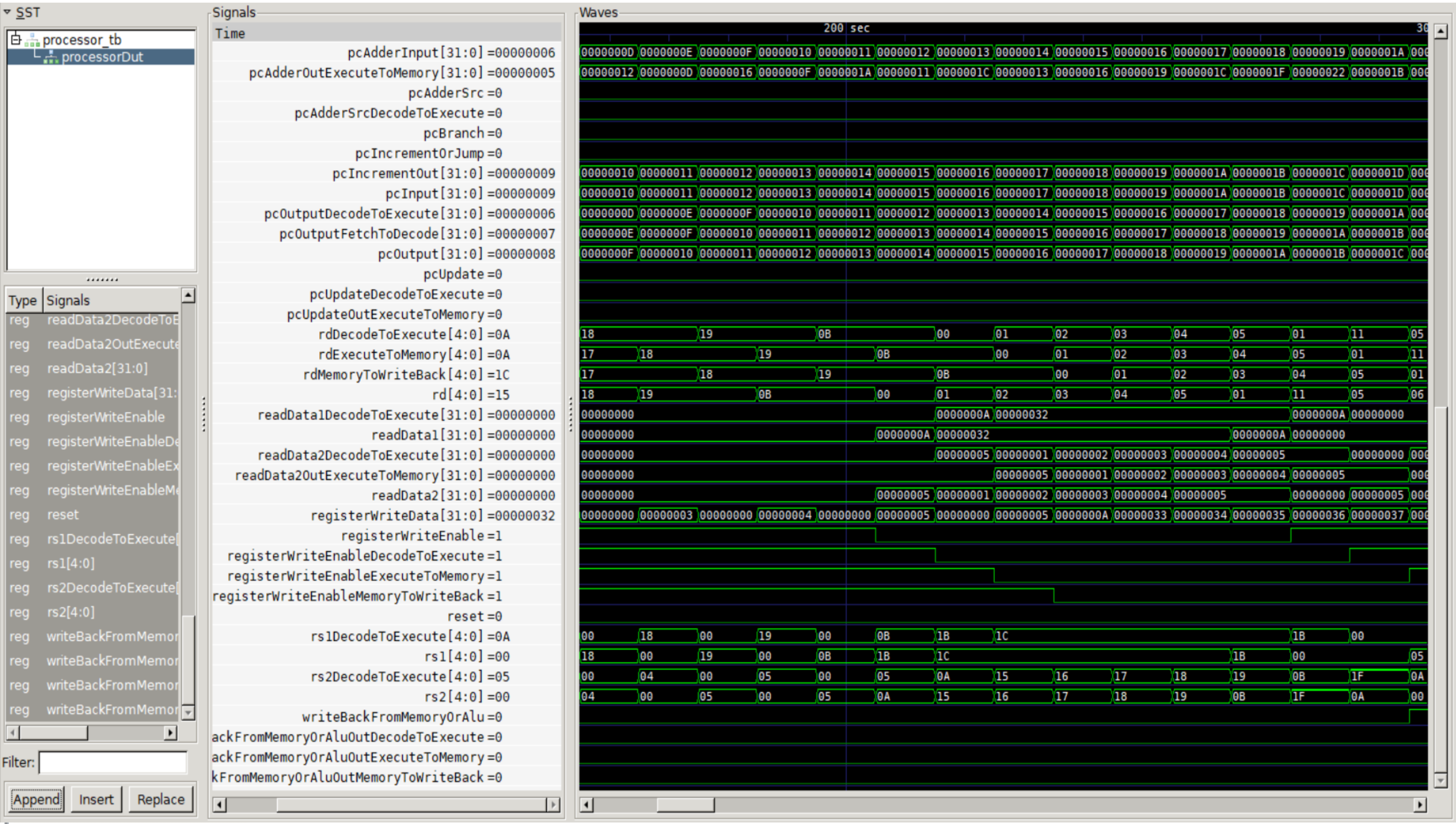
- We have successfully completed the design of the **RISC-V RV32I pipelined processor**, which is the foundational core of our SoC.
- The core is now ready for the next phase, which involves integrating our custom hardware, peripherals, and the second core to create the full **System-on-a-Chip**.

**Extracted Processor Architecture from Synopsys VCS:**



# PROGRESS

## Simulated Waveform of Pipelined Processor:



```
reg  readData2DecodeToE
reg  readData2OutExecute
reg  readData2[31:0]
reg  registerWriteData[31:
reg  registerWriteEnable
reg  registerWriteEnableDe
reg  registerWriteEnableEx
reg  registerWriteEnableMe
reg  reset
reg  rs1DecodeToExecute[
reg  rs1[4:0]
reg  rs2DecodeToExecute[
reg  rs2[4:0]
reg  writeBackFromMemor
reg  writeBackFromMemor
reg  writeBackFromMemor
reg  writeBackFromMemor
```

```
Time
pcAdderInput[31:0] =00000006                 0000000D 0000000E 0000000F 00000010 00000011 00000012 00000013 00000014 00000015 00000016 00000017 00000018 00000019 0000001A 000
pcAdderOutExecuteToMemory[31:0] =00000005    00000012 0000000D 00000016 0000000F 0000001A 00000011 0000001C 00000013 00000016 00000019 0000001C 0000001F 00000022 0000001B 000
pcAdderSrc =0
pcAdderSrcDecodeToExecute =0
pcBranch =0
pcIncrementOrJump =0
pcIncrementOut[31:0] =00000009               00000010 00000011 00000012 00000013 00000014 00000015 00000016 00000017 00000018 00000019 0000001A 0000001B 0000001C 0000001D 000
pcInput[31:0] =00000009                      00000010 00000011 00000012 00000013 00000014 00000015 00000016 00000017 00000018 00000019 0000001A 0000001B 0000001C 0000001D 000
pcOutputDecodeToExecute[31:0] =00000006      0000000D 0000000E 0000000F 00000010 00000011 00000012 00000013 00000014 00000015 00000016 00000017 00000018 00000019 0000001A 000
pcOutputFetchToDecode[31:0] =00000007        0000000E 0000000F 00000010 00000011 00000012 00000013 00000014 00000015 00000016 00000017 00000018 00000019 0000001A 0000001B 000
pcOutput[31:0] =00000008                     0000000F 00000010 00000011 00000012 00000013 00000014 00000015 00000016 00000017 00000018 00000019 0000001A 0000001B 0000001C 000
pcUpdate =0
pcUpdateDecodeToExecute =0
pcUpdateOutExecuteToMemory =0
rdDecodeToExecute[4:0] =0A                   18 19 0B 00 01 02 03 04 05 01 11 05
rdExecuteToMemory[4:0] =0A                   17 18 19 0B 00 01 02 03 04 05 01 11
rdMemoryToWriteBack[4:0] =1C                 17 18 19 0B 00 01 02 03 04 05 01
rd[4:0] =15                                  18 19 0B 00 01 02 03 04 05 01 11 05 06
readData1DecodeToExecute[31:0] =00000000     00000000 0000000A 00000032 0000000A 00000000
readData1[31:0] =00000000                    00000000 0000000A 00000032 0000000A 00000000
readData2DecodeToExecute[31:0] =00000000     00000000 00000005 00000001 00000002 00000003 00000004 00000005 00000000 000
readData2OutExecuteToMemory[31:0] =00000000  00000000 00000005 00000001 00000002 00000003 00000004 00000005 000
readData2[31:0] =00000000                    00000000 00000005 00000001 00000002 00000003 00000004 00000005 00000000 00000005 000
registerWriteData[31:0] =00000032            00000000 00000003 00000000 00000004 00000000 00000005 00000000 00000005 0000000A 00000033 00000034 00000035 00000036 00000037 000
registerWriteEnable =1
registerWriteEnableDecodeToExecute =1
registerWriteEnableExecuteToMemory =1
registerWriteEnableMemoryToWriteBack =1
reset =0
rs1DecodeToExecute[4:0] =0A                  00 18 00 19 00 0B 1B 1C 1B 00
rs1[4:0] =00                                 18 00 19 00 0B 1B 1C 1B 00 05
rs2DecodeToExecute[4:0] =05                  00 04 00 05 00 05 0A 15 16 17 18 19 0B 1F 0A
rs2[4:0] =00                                 04 00 05 00 05 0A 15 16 17 18 19 0B 1F 0A 00
writeBackFromMemoryOrAlu =0
ackFromMemoryOrAluOutDecodeToExecute =0
ackFromMemoryOrAluOutExecuteToMemory =0
kFromMemoryOrAluOutMemoryToWriteBack =0
```

# TESTING PLAN

- **Systematic Functional Testing:** We used a rigorous verification plan to systematically test the functionality of every component and every custom instruction.
- **Full System-Level Verification:** We performed an end-to-end test of the entire data flow, from the simulated sensor to the final diagnosis.
- **Performance Analysis:** We ran tests to quantify our improvements by comparing the clock cycle count of our custom SoC against a standard processor.
- **Corner-Case Coverage:** Our testing included a variety of scenarios to ensure that the system works correctly even under non-ideal or unexpected conditions.

## CONCLUSION

- **Architectural Innovation:** We designed and built a custom **RISC-V SoC** with **Vector MAC (V-MAC)** and **Bit-Reversal** instructions to accelerate signal processing.
- **Performance & Efficiency:** Our design delivers superior **speed** and **power efficiency** by performing complex calculations in a single clock cycle.
- **On-Device Intelligence:** The SoC enables **real-time diagnosis** using an on-chip **CNN**, keeping sensitive data secure and private.
- **Key Achievement:** We proved that a **custom hardware-software solution** is a superior approach for specialized applications compared to general-purpose or fixed, proprietary platforms.

# REFERENCES

1. **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).
2. **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).
3. **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).
4. **Oppenheim, A. V., & Schafer, R. W.** (2014). *Discrete-Time Signal Processing*. Pearson. (The bible for DSP concepts like FIR filters and FFT).
5. **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).
6. **Waterman, A., & Asanovic, K.** (2019). *The RISC-V Reader: An Open Architecture Atlas*. (A great summary of the RISC-V ISA and its philosophy).
7. **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).

# REFERENCES

8. **Spear, J.** (2008). *SystemVerilog for Verification: A Guide to Learning the Testbench Language Features*. Springer. (A comprehensive guide to UVM methodology).

9. **Accellera Systems Initiative.** (2017). *Universal Verification Methodology (UVM) 1.2 User's Guide*. (The official UVM standard and documentation).

10. **Goodfellow, I., Bengio, Y., & Courville, A.** (2016). *Deep Learning*. MIT Press. (Foundational text on deep learning, including CNNs).

11. **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).

12. **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).

13. **Furber, S.** (2015). *ARM System-on-Chip Architecture*. Pearson. (While ARM-centric, this book provides excellent general knowledge on SoC design, including bus protocols).

# THANK YOU