ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25may1092

Implementation of Robust Real-Time RV32I Processor

K.Venkanna Niadu[1]; Akurathi Yamini[2];
Yalla Naga Venkata Satya Ajay Kumar[3];
Srinivasula Subhadra Supraja[4];Gudiwada Sushma[5]

[1]Assitant professor Dept.of ECE DNR College of Engineering & Technology, Bhimavaram
[2,3,5]Dept.of ECE DNR College of Engineering & Technology, Bhimavaram
[4]Srinivasula Subhadra Supraja Dept.of ECE DNR College of Engineering & Technology, Bhimavaram

Publication Date: 2025/05/29

Abstract: In this project focuses on enhancing 32-bit RV32I Version 2.0 processor robustness and real – time capabilities through the integration of advanced error handling and real-time capabilities through the integration of advanced error handling mechanisms and features tailored for Real-Time Operating System (RTOS) support. To improve reliability, a sophisticated Interrupt Error Checker (IEC) is combined with robust Error Correction Codes (ECC). The IEC classifies interrupts, performs error checks related to interrupt handling, and provides detailed error information, while ECC protects data integrity in memory and registers. This synergistic combination creates a multi-layered defense against errors, crucial for mission critical systems. Simultaneously, the design addresses RTOS requirements by focusing on deterministic execution and low-latency interrupt handling. Techniques for deterministic execution include predictable instruction timing, cache management strategies (locking, partitioning, scratchpad memory), and simplified pipeline design. Low-latency interrupts are achieved through fast dispatch, prioritized interrupts, interrupt nesting, and minimized overhead. Additional RTOS-related features, such as atomic operations and hardware task management support, are also considered. This combined approach aims to create a processor capable of reliable operation in demanding real-time environments, ensuring both integrity and timely responsiveness to critical events

Keywords: *RV32I, RTOS, IEC.*

How to Cite: *K.Venkanna Niadu;Akurathi Yamini; Yalla Naga Venkata Satya Ajay Kumar; Srinivasula Subhadra Supraja; Gudiwada Sushma; (2025) Implementation of Robust Real-Time RV32I Processor. International Journal of Innovative Science and Research Technology*,10(5), 2306-2312. https://doi.org/10.38124/ijisrt/25may1092

I. INTRODUCTION

The RV32I Real-Time Robust Processor is a specialized implementation of the RISC-V 32-bit Integer (RV32I) architecture designed to meet the stringent demands of real-time applications, including industrial automation, automotive systems, and aerospace control. This processor integrates real-time computing capabilities with a robust architecture that enhances fault tolerance, power efficiency, and predictable execution. It adheres to the RV32I instruction set, ensuring compatibility with the open-source RISC-V ecosystem while incorporating enhancements such as real-time task scheduling, low-latency interrupt handling, and resilience to transient faults. The Real-Time RV32I Robust Processor include: • Deterministic Execution: Predictable instruction timing and real-time scheduling support. • Low-Latency Interrupt Handling: Optimized for quick response to time-critical tasks. • Fault-Tolerant Design: Error detection and correction mechanisms for high reliability. • Energy Efficiency: Optimized power management to support embedded and IoT applications. • Modular and Scalable Architecture: Customizable for different real-time workloadsparentheses, following the example. Some components, such as multi-leveled equations, graphics, and tables are not prescribed, although the various table text styles are provided. The formatter will need to create these components, incorporating the applicable criteria that follow.

II. RV32I PROCESSOR

A. RV32I

The RV32I is the 32-bit integer base instruction set of the RISC-V architecture, an open-source and highly modular instruction set architecture(ISA).It forms the foundation for many RISC-V processors and is designed to be simple, efficient, and extensible.

B. RV32I ISA Formats

The RV32I version of the RISC-V ISA serves as the foundation for 32-bit RISC-V processors, providing a streamlined and efficient instruction set architecture for various computing applications.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25may1092

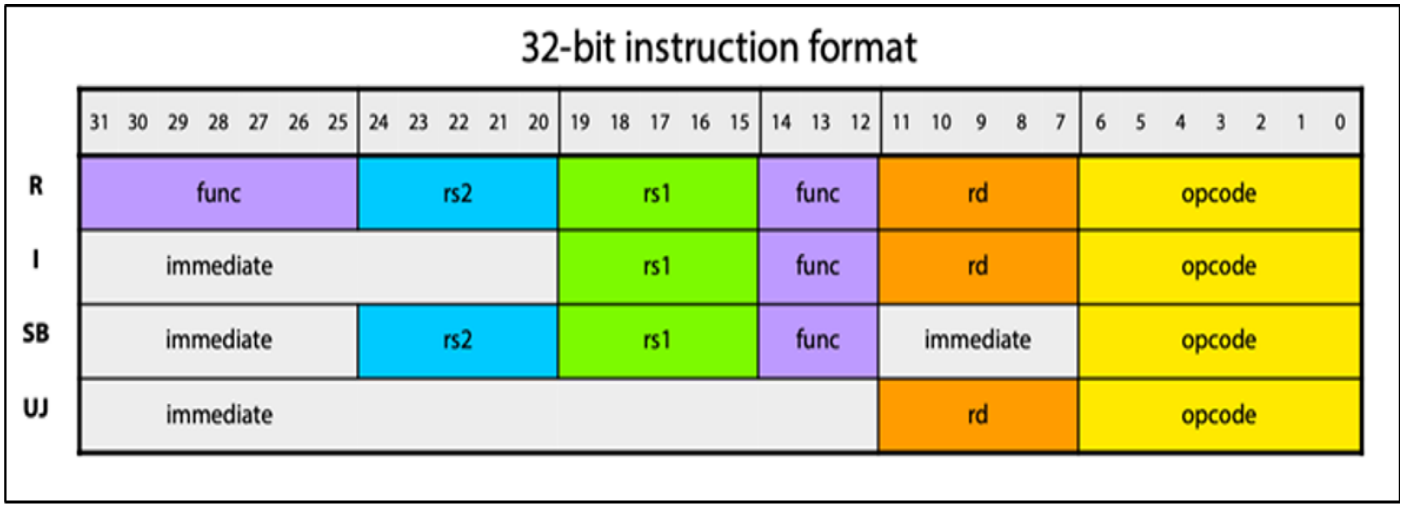

Fig 1 RV32I ISA Formats

Table 1 Reg-Reg type Instruction Format

Register- Register(R-type) Instruction Format					
31 25	24 20	19 15	14 12	11 7	6 0
funct7	rs2	rs1	funct3	rd	opcode

Register-type RV32I ISA V 2.0. It has six fields. The R-type format, designed for arithmetic and logical operations, utilizes registers as both operands and result destinations. With fields including the opcode for operation identification, register source index (rs1 and rs2), a destination registers index (rd), and additional function codes (funct3 and funct7), it enables efficient execution of operations like addition, subtraction, and bitwise logical operations.

Table 2 Immediate type Instruction Format

Immediate type Instruction Format					
31	20 19	15 14	12 11	7 6	0
imm[11:0]	rs1	funct3	rd	opcode	

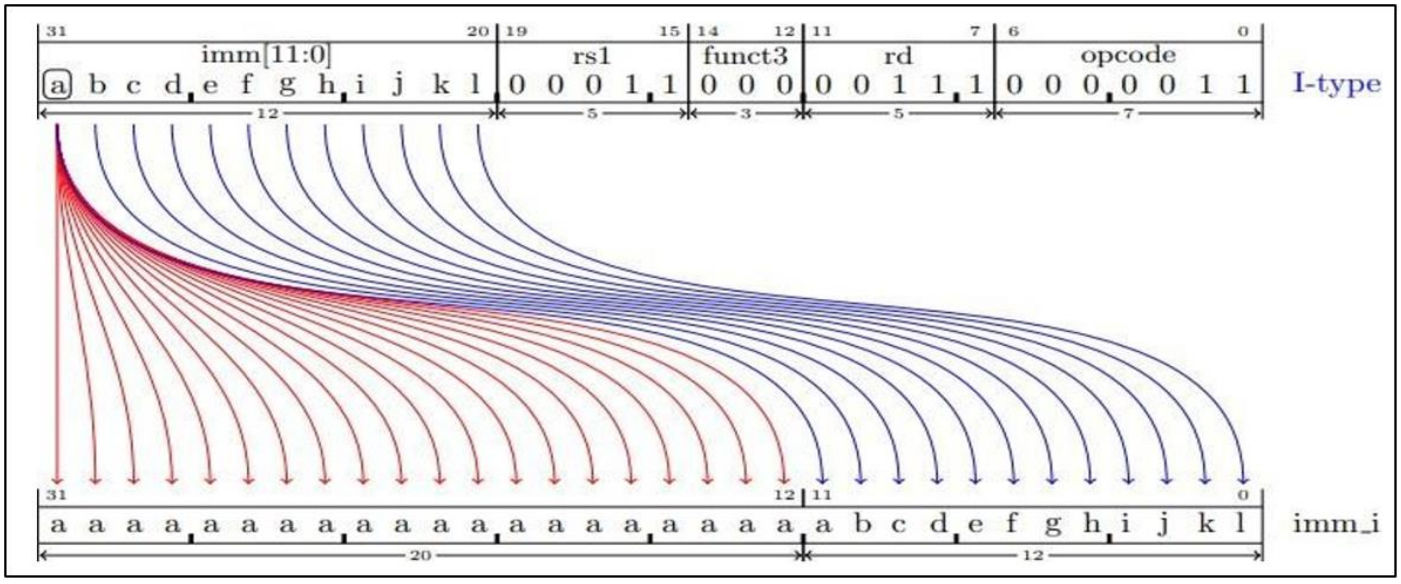

Fig 2 Decoding an I-type Instruction

Immediate-type RV32I ISA V 2.0. the I-type format facilitates operations involving immediate values alongside a single register operand. Employing fields such as the immediate value (imm), a source register index (rs1), a destination register index (rd), and a function code (funct3), it enables instructions like immediate addition (add) and bitwise immediate logical operations (ori).

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25may1092

Table 3 Store (S-type) RV32I Instruction Format

Store (S-type) RV32I Instruction Format					
31 25	24 20	19 15	14 12	11 7	6 0
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode

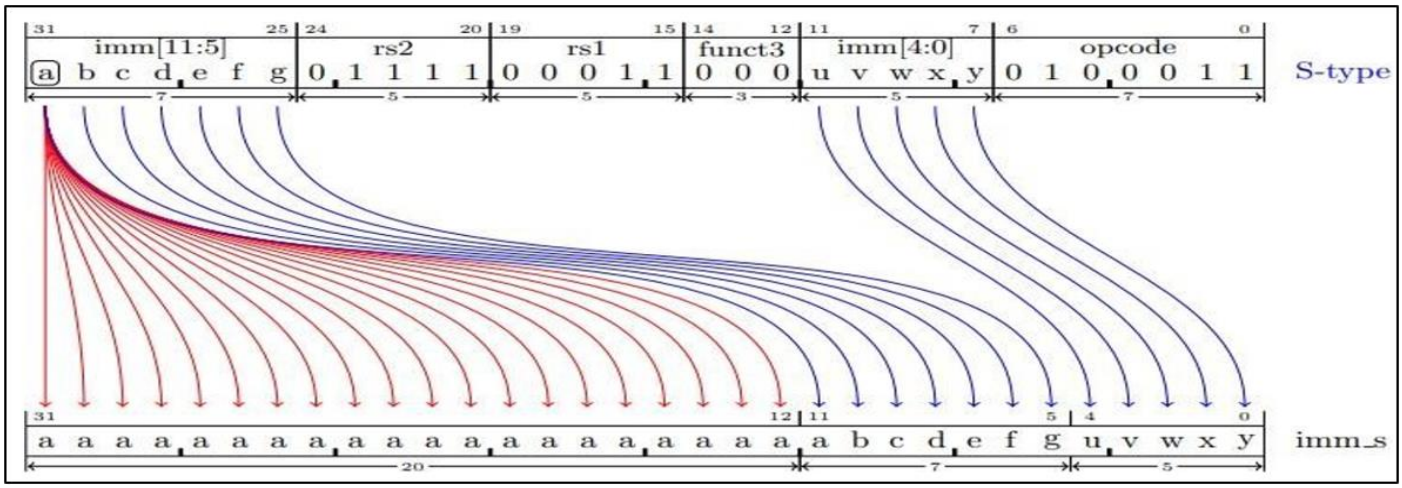

Fig 3 Decoding an S-type Instruction

Store-type RV32I ISA V 2.0. The S-type format, optimized for memory stores, involves transferring register values to memory locations, utilizing source register indices (rs1 and rs2), an immediate offset (imm), and a function code (funct3) to define memory addresses and access instructions such as store word (sw).

Table 4 Branch (B-Type) RV32I Instruction Format

Branch (B-Type) RV32I Instruction Format							
31	30 25	24 20	19 15	14 12	11 8	7	6 0
imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode

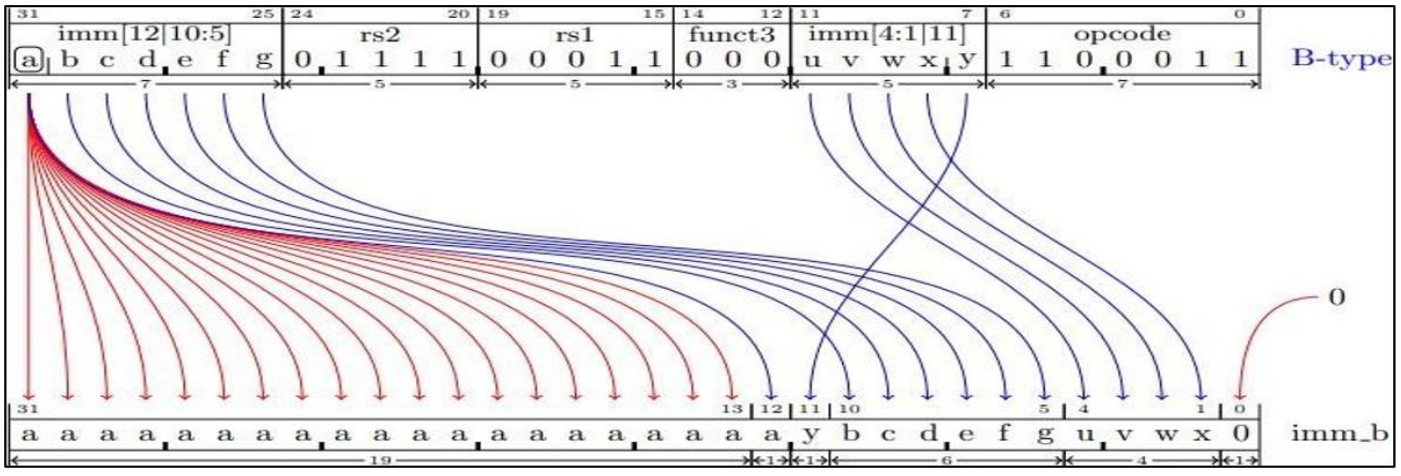

Fig 4 Decoding a B-type Instruction

Branch-type RV32I ISA V2.0. the B-type format supports conditional branches, utilizing source register indices (rs1 and rs2), an immediate branch offset (imm), and a function code (funct3) to evaluate conditions and execute branch instructions like branch if equal (beq).

Table 5 Jump Type Instruction Format

Upper Immediate (U-type) & JUMP (J-type) RV32I Instruction Format					
31 30	21 20	19	12 11	7 6	0
imm[20]	imm[10:1]	imm[11]	imm[19:12]	rd	opcode

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25may1092

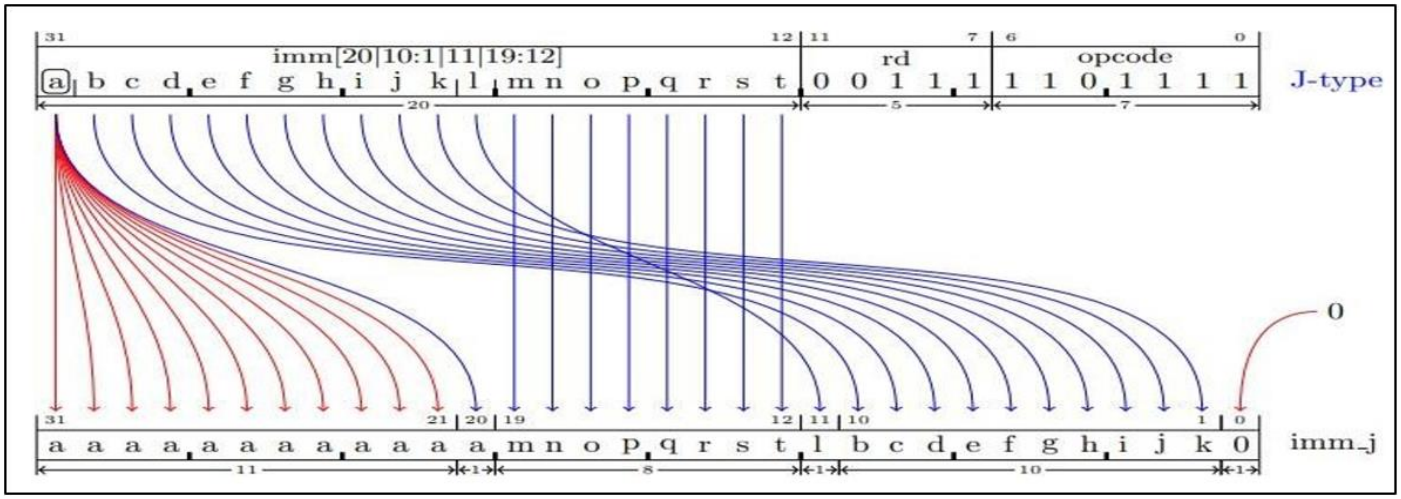

Fig 5 Decoding a J-type Instruction

Table 6 Upper Type Instruction Format

31 12	11 7	6 0
imm[31:12]	rd	opcode

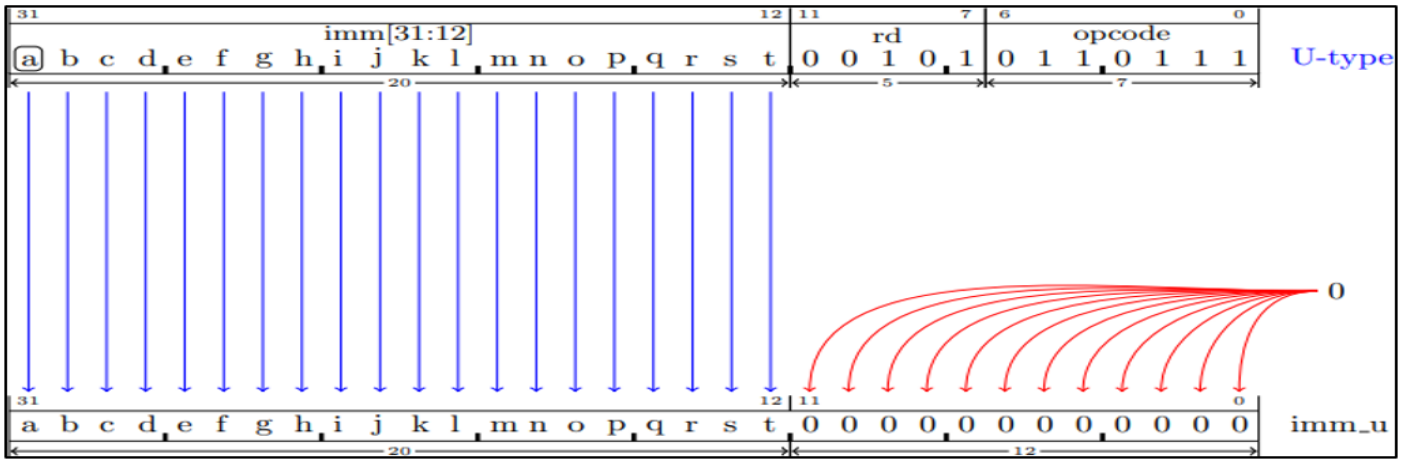

Fig 6 Decoding U-type Instruction

U-type and J-type RV32I ISA V 2.0, the decoding of U-type format and J-type format are similar to each other U-type is used for operations on the upper immediate value in a register and J-type facilitates unconditional jumps through immediate address offsets, featuring a destination register index (rd) and an immediate value (imm) for instructions like load upper immediate (lui).

- R-Type : register operations (rd, rs1, rs2, funct, opcode)
- I-Type : immediate operations & loads (rd, rs1, imm, funct, opcode)
- SB-Type : branching (rs1, rs2, imm, funct, opcode)
- UJ-Type : jump instructions (rd, imm, opcode)

III. METHODOLOGY

➢ *ECC Memory Module*

This module implements a memory system with ECC (Hamming code) to detect and correct single-bit errors in the stored data.

➢ *Error Detection and Correction*

When reading data, the generate ECC function is used to recalculate the ECC code for the stored data.

➢ *Hamming Code Generation*

The generate ECC function calculates the 3-bit ECC code for the 8-bit data using XOR operations.

Data Encoding Before data is stored in memory Parity bits are calculated and embedded into the data word using the Hamming Code algorithm. These parity bits are positioned at specific locations within the memory word (addresses that are powers of 2: 1, 2, 4, 8, etc.). For example, a data word of 8 bits might expand to 12 bits after adding 4 parity bits. Each parity bit ensures that a specific subset of bits (including itself) has either an even or odd number of 1s, depending on the chosen parity scheme. Data Storage The encoded word (data + parity bits) is stored in memory. This added redundancy allows the system to detect and correct errors later during data retrieval. Data Retrieval and Error Detection When data is read from memory. The system recalculates the parity bits using the retrieved data and compares them to the stored parity bits. If all

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25may1092

recalculated parity bits match the stored values, the data is considered error-free. If there's a mismatch, it indicates an error, and the exact location of the error can be determined. Error Correction Hamming Code identifies the position of the faulty bit using the parity bits. This is done as follows. The parity bits are used to form a binary number, where each bit represents whether a parity check passed or failed. The binary number indicates the position of the erroneous bit in the memory word. Single Error Correction, Double Error Detection (SECDED) Many memory systems implement an extended form of Hamming Code called SECDED, which Detects and corrects single-bit errors. Detects, but does not correct, double-bit errors (to signal more serious faults). Adds an additional overall parity bit to the encoded word for distinguishing between single-bit and multi-bit errors.

➢ *Example in Memory*

Suppose a 4-bit data word (1011) is to be stored in a memory system using Hamming Code with 3 parity bits:

➢ *Calculate parity bits for the positions P1, P2, P4P_1, P_2, P_4:*

- P1P_1: Covers bits 1, 3, 5, 7 → Value: 1.
- P2P_2: Covers bits 2, 3, 6, 7 → Value: 0.
- P4P_4: Covers bits 4, 5, 6, 7 → Value: 0.

➢ *Encoded word: 1010011 (positions: P1P2D3P4D5D6D7P_1 P_2 D_3 P_4 D_5 D_6 D_7).*

- If a bit flips during storage (e.g., position 5 changes from 1 to 0), the parity checks on retrieval will fail.
- The error position is determined (binary 101 → decimal position 5), and the memory system corrects the error by flipping the bit back.

➢ *Advantages of Hamming Code in Memory Systems*

- High Reliability: Corrects single-bit errors and detects double-bit errors efficiently.
- Low Overhead: Requires relatively few additional bits for parity.
- Real-Time Operation: Detects and corrects errors on-the-fly during read operations.
- Hamming Code's simplicity and effectiveness make it an integral part of memory systems in servers, embedded systems, and mission-critical applications.

IV. DESIGN PROCESS

1.Processor Core: This is the central processing unit responsible for executing instructions and managing the overall operations of the system. 2. Error Handling Layer: 1. This layer ensures reliability by detecting, classifying, and managing errors during processing. 2. It includes mechanisms like: 1. Interrupt Error Checker (IEC): Checks for interrupt-related errors, classifies interrupts, and provides detailed error reports. 2. Error Correction Codes (ECC): Protects data integrity in memory and registers through robust coding mechanisms. 3. RTOS Layer: 1. Represents the Real-Time Operating System support integrated into the processor. 2. Provides deterministic execution, low-latency interrupt handling, and features like atomic operations and task scheduling, which are essential for real-time environments. 4.Mission-Critical Application Layer: 1. Designed for high-priority applications that require reliable, timely, and efficient execution. 2. Focuses on robust operations to meet the demands of mission-critical systems.

V. SIMULATION RESULTS

Using Xilinx Vivado, the design for the Interrupt checker and Real-Time Operating System (RTOS).

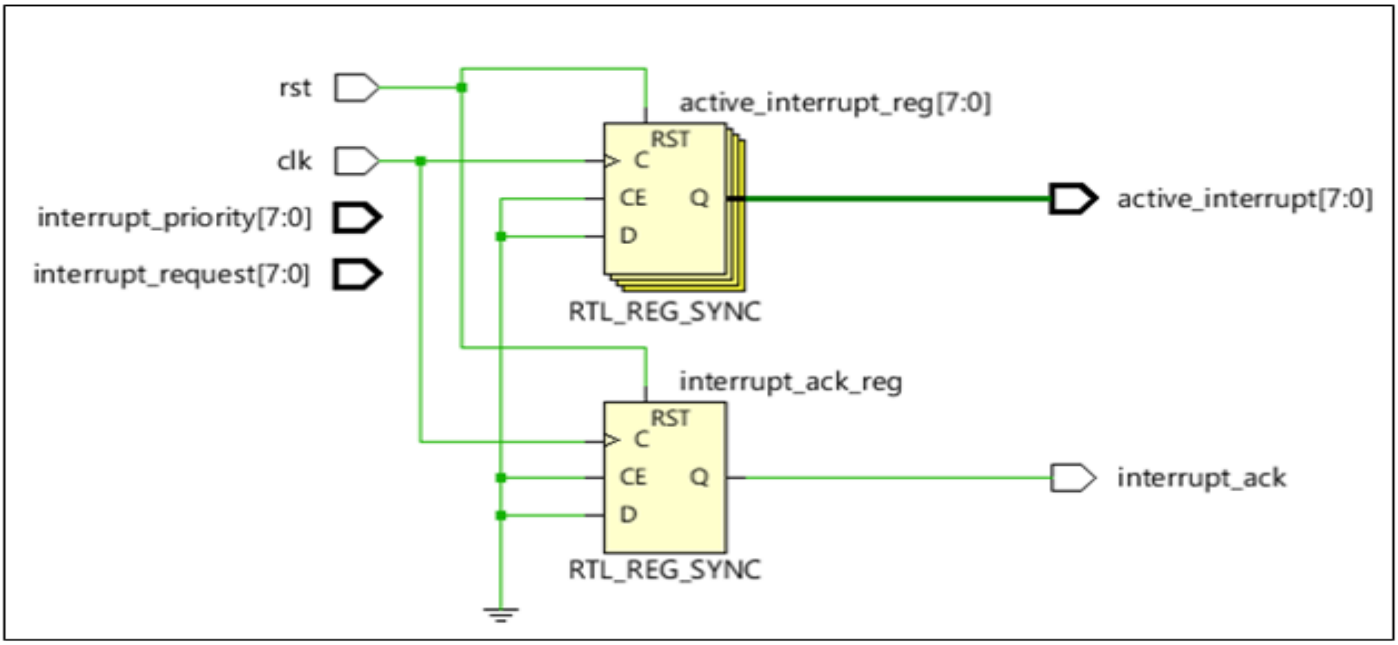

Fig 7 Schematic of Interrupt Controller

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25may1092

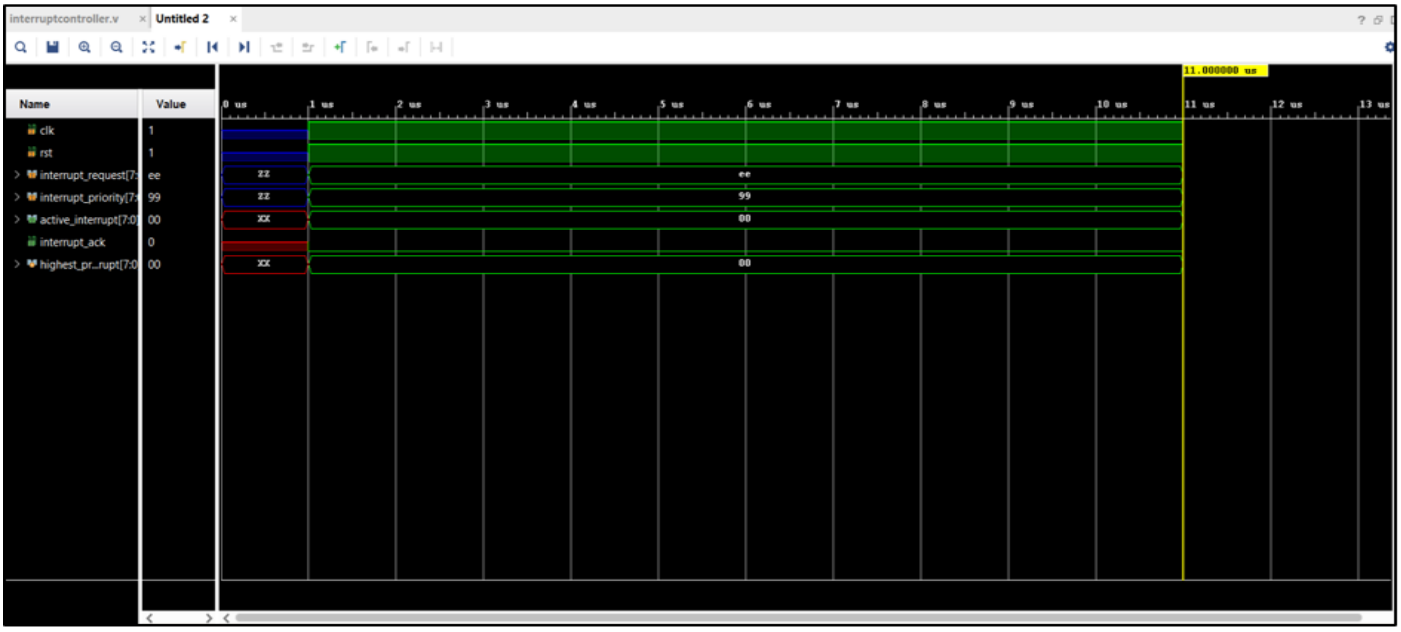

Fig 8 Simulation Design of Interrupt Controller

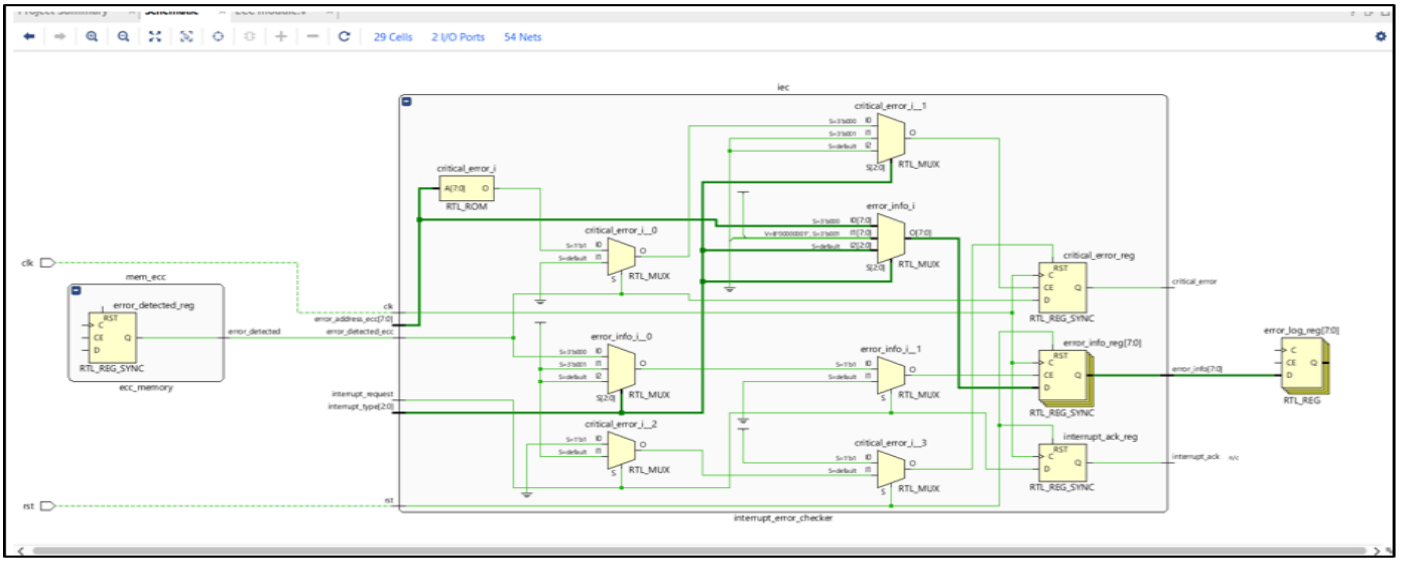

Fig 9 Schematic of IEC and ECC

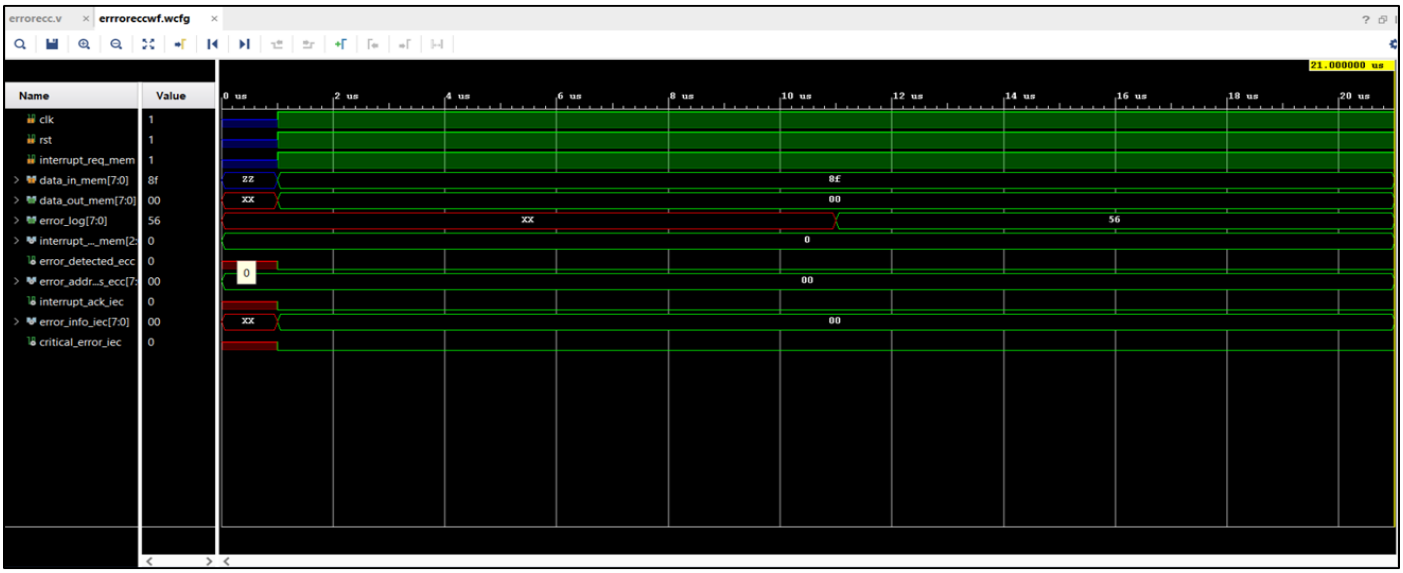

Fig 10 Simulation Design of IEC and ECC

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25may1092

VI. CONCLUSION

This design transforms the RV32I processor into a robust, RTOS-ready platform capable of balancing reliability and real-time responsiveness. By addressing both error resilience (via IEC/ECC) and determinism (through pipeline/cache optimizations), it provides a foundation for deploying mission-critical systems in unpredictable environments. Future work includes silicon validation and benchmarking against industry-standard RTOS workloads.

REFERENCES

[1]. S. Prabhakaran, M. N and V. Veda Narayanan, "Design and Analysis of a Multi Clocked Pipelined Processor Based on RISC-V," 2022 International Conference on Communication, Computing and Internet of Things (IC3IoT), Chennai, India, 2022, pp. 1-5, doi: 10.1109/IC3IOT53935.2022.9767960.J.

[2]. Waterman, K. Asanovic and SiFive Inc., "The RISC-V Instruction Set Manual Volume I: Unprivileged ISA," Document Version 20191213, EECS Dept, University of California, Berkeley, CA, USA, December 13, 2019.

[3]. Thanga Dharsni, K. S. Pande and M. K. Panda, "Optimized Hazard Free Pipelined Architecture Block for RV32I RISC-V Processor," 2022 3rd International Conference on Smart Electronics and Communication (ICOSEC), Trichy, India, 2022, pp. 739-746, doi: 10.1109/ICOSEC54921.2022.9952122.

[4]. W. Zhang, Y. Zhang and K. Zhao, "Design and Verification of Three-stage Pipeline CPU Based on RISC-V Architecture," 2021 5th Asian Conference on Artificial Intelligence Technology (ACAIT), Haikou, China, 2021, PP. 697-703, doi: 10.1109/ACAIT53529.2021.9731161.

[5]. K. Dennis et al., "Single cycle RISC-V micro architecture processor and its FPGA prototype," 2017 7th International Symposium on Embedded Computing and System Design (ISED), Durgapur, India, 2017, pp. 1-5, doi: 10.1109/ISED.2017.8303926.