DESIGN AND ANALYSIS OF 32 – BIT HIGH SPEED RISC PROCESSOR USING PIPELINING TECHNIQUE

K.G.VENKATA KRISHNA[1], ANUSHA BATHALA[2] , ASIRI NAIDU KOYYANA[3], VAMSI KRISHNA REDDY.S[4]

Assistant Professor (C[1]), UG Student[2,3,4]

Department of Electronics and Communication Engineering
Krishna University College of Engineering and Technology
Krishna University
Rudravaram, Machilipatnam.
Ph.: 9966963399
EId: kgvk.aca.ece@kru.ac.in, Eid : anushabathala2005@gmail.com , Eid : koyyanaasirinaidu6305@gmail.com, Eid: sunkarivamsikrishnareddy@gmail.com

To Cite this Article
K.G.VENKATA KRISHNA[1], ANUSHA BATHALA[2] , ASIRI NAIDU KOYYANA[3], VAMSI KRISHNA REDDY.S[4]
"**DESIGN AND ANALYSIS OF 32 – BIT HIGH SPEED RISC PROCESSOR USING PIPELINING TECHNIQUE.**" *Journal of Science and Technology, Vol. 10, Issue 04-April 2025, pp34-44*

Article Info
Received: 26-01-2025 Revised: 05-04-2025 Accepted: 15-04-2025 Published:25 -04-2025

ABSTRACT

The Reduced Instruction Set Computer (RISC) architecture prioritizes a compact, efficient set of instructions that execute in uniform cycles, enabling high-performance computing across devices—from embedded systems to supercomputers. This research presents the design and implementation of a **32-bit RISC processor** leveraging a **pipelining technique** to enhance throughput while maintaining computational efficiency. The proposed processor supports **16 arithmetic and logical operations** and adheres to the Harvard architecture, featuring separate data and instruction memories for concurrent access.

The processor incorporates a **32-bit ALU** for arithmetic and logical computations, along with **general-purpose registers** and a dedicated **flag register** to track carry, zero, and parity statuses. The pipelined design divides instruction execution into multiple stages (fetch, decode, execute, memory access, and write-back), significantly improving instruction throughput. Each instruction is encoded uniformly, and the design supports multiple addressing modes for flexibility.

All modules are implemented in **Verilog HDL**, with individual components rigorously verified before integration into the top-level design. Functional validation and synthesis are performed using **Xilinx Vivado 2023.1**, with simulation results confirming correct operation. The proposed 32-bit RISC processor demonstrates a balance between simplicity, performance, and scalability, making it suitable for applications demanding low-latency processing.

Keywords: RISC, 32-bit processor, pipelining, arithmetic-logic unit (ALU), Verilog, Xilinx Vivado

INTRODUCTION

People began looking at other options as the controller architecture in CISC became more difficult and the performance was also below expectations. It had been discovered that speed is eliminated when a CPU communicates with memory. The only way to improve CPI was to keep the instruction set as basic as possible. Simple in terms of appearance rather than functionality. Because of this, relatively few instructions—probably only load and store—in a typical RISC architecture require the CPU to access data from memory. In the end, pipelining increased performance by a new dimension just by adding a few extra registers, which lowers CPI and raises throughput. As a result, the command can be successfully carried out in a single clock cycle.

It's a widespread misconception that instructions are simply deleted to produce a smaller set of instructions when the term "Reduced Instruction Set Computer" is used. The size of RISC instruction sets has actually increased over time, and currently many of them contain a greater number of instructions than many CISC CPUs. The word "Reduced" in that sentence refers to the fact that compared to the "complex instructions" of CISC CPUs, which may require multiple data memory cycles to execute a single instruction, any given instruction performs less work, accomplishing at most one data memory cycle. Because RISC design uses streamlined machine instructions for commonly utilized functions, research has demonstrated that it significantly increases computer performance.

In RISC-based systems, the following characteristics are usual.

1) Pre-fetching: Pre-fetching is the process of adding a subsequent instruction or instructions to an event queue before the current instruction is

finished.

2) Pipelining: This technique enables for the issuance of an instruction before the one that is now being executed has finished.

A processor that can send out several instructions at once is said to be doing superscalar activity.

LITERATURE REVIEW

Design and Comparison of Multiplier using Vedic Sutras Authors: S. Lad and V. S. Bendre **Conference:** 2019 5th International Conference on Computing, Communication, Control, and Automation (ICCUBEA)

This paper explores high-speed multipliers for real-time applications, emphasizing Vedic mathematics-based designs. The study compares three Vedic sutras—Urdhva Tiryakbhyam, Ekanyunena Purvena, and Ekadhikena Purvena—for optimizing area, speed, and power. A 16- bit Vedic multiplier implemented in Verilog (Xilinx Vivado 17.1) demonstrates that Ekadhikena Purvena reduces area by **70.26%** and increases speed by **90.41%** compared to Urdhva Tiryakbhyam. The Nikhilam sutra offers optimal power efficiency. These findings are critical for digital signal processing and image processing applications.

Summary**:** Vedic multipliers enhance speed and reduce area, with Ekadhikena Purvena and Nikhilam sutras offering the best performance trade-offs.

Design and Implementation of a 16-Bit RISC Processor on FPGA Authors: **Balpande Vishwas**

V. et al. Year: **2015**

This work presents a **16-bit RISC processor** based on Harvard architecture, designed using Verilog HDL. The processor employs a hardwired control unit for faster operation compared to microprogrammed designs. Key components, including an ALU (built structurally from half- adders) and memory modules (behaviorally modeled), are integrated for efficient execution. The design supports scalability to **32-bit or 64-bit** architectures with minimal modifications.

Summary: The processor achieves high performance through an optimized datapath and modular design, enabling future expansion.

Design & Verification of a 16-bit RISC Processor for SOC Applications Authors: **Seung Pyo Jung et al.** Conference: 2008 International SoC Design Conference

This paper introduces **a** 16-bit RISC processor with Harvard architecture, a **5-stage pipeline**, and debug logic for SOC applications. The processor replaces the 8051 architecture, reducing instruction complexity and improving speed. Validated via FPGA implementation, it supports ADPCM vocoder and SOLA algorithms. Future work includes developing a compiler and GDB server for high-level debugging. Summary**:** The RISC-V based design outperforms the 8051, offering better speed and scalability for embedded systems.

2.1 EXISTINGMETHOD

The Existing method presents a **16-bit pipelined RISC processor** designed using **Verilog HDL** and implemented on the **Xilinx ISE** platform. The processor employs a **4-stage pipeline architecture** to enhance instruction throughput by overlapping fetch, decode, execute, and write-back operations. The design focuses on simplicity and efficiency, incorporating key components such as an **Arithmetic Logic Unit (ALU)**, **register file**, **data memory unit**, and a **control unit**. The ALU supports **13 instructions**, including arithmetic operations like addition and subtraction (using a **Carry Look-Ahead Adder** for reduced delay), logical operations (AND, OR, XOR, etc.), and shift operations. For multiplication, the processor utilizes a **Wallace Tree Multiplier**, chosen for its parallel processing capability, which improves speed and reduces power consumption compared to traditional methods.

Memory management in this design is handled through a **256 × 16-bit register file** (512 bytes), using **direct addressing** for data access. The control unit generates synchronized clock signals and read/write enable signals to coordinate pipeline stages and memory operations. A **4-bit flag register** monitors ALU results, indicating conditions such as negative outputs (sign flag), parity (odd/even), and zero results. The processor achieves a **maximum clock frequency of 468.75 MHz** with a **power dissipation of 0.082W**, demonstrating high-speed performance while maintaining energy efficiency.

Disadvantages of the Existing Method

1. **Limited Data Width (16-bit):**
   - The processor's **16-bit architecture** restricts its ability to handle larger data types, which are common in modern applications. A **32-bit or 64-bit design** would offer better compatibility with contemporary software and higher computational precision.
2. **Basic Addressing Modes:**
   - The processor supports only **direct addressing**, limiting flexibility in memory access. Advanced addressing modes (e.g., **indirect, indexed, or relative addressing**) would improve programming efficiency and data manipulation capabilities.
3. **Pipeline Hazards:**
   - While pipelining improves speed, it introduces **data and control hazards**. The paper does not detail **advanced hazard mitigation techniques** (e.g., dynamic branch prediction or out-of-order execution), which could further enhance performance.
4. **No Cache Memory:**
   - The absence of an **instruction or data cache** means memory access delays could bottleneck performance, especially in data-intensive applications.
5. **Limited Instruction Set (13 Instructions):**
   - The ALU supports only **13 basic operations**, omitting complex functions like **floating-point arithmetic** or **hardware division**, which are essential for broader applications.
6. **No Superscalar or Multi-Core Support:**
   - The design does not explore **parallel instruction execution (superscalar)** or **multi-core processing**, which are critical for modern high-performance computing.

PROPOSED METHOD

The function of the processor is to execute each and every instruction set efficiently as per the machine language. ALU is the combinational circuit which means Arithmetic and Logical Unit. This unit is designed to perform various numbers using various instruction sets. In Processor, ALU inputs consist of instruction (machine word) which is operation code (opcode) and some operands. So the opcode tells the ALU which and what operation is to be performed then these operands are used in the operation.

There is a small set of data holding place that is known as Register bank. The ALU stores the result of operation in accumulator which later on is placed in a storage register and it checks the bits and indicates whether the operation was performed successfully. If not successfully executed then some type of status will be shown i.e. even known as Z-Flag or status register. Its function is to execute programs and operate efficiently for the data stored in memory. A processor has a set of instructions which is nothing but a command to perform a task in a computer. The control unit holds the instruction to be executed. In CPU, the registers such as address register, data register and an instruction register is present. The performance of the CPU is to fetch, decode and execute the operations on memory according to the registers. The task of IR includes the decoding the op-code, determining the instruction, determining which operands are in memory, retrieving the operands in memory then assigning a command to a processor to execute the instruction. This is done with the help of control unit which generates the timing signals that controls the various processing elements which involves in execution of the instruction.

The MAR is also called as address buffer; the address in the program counter is applied to memory so after the increment in PC to the next address the current instruction is stored in the Memory location. The MAR is completely loaded with Binary words which point the location of the word in RAM. This location stores the instruction in it.

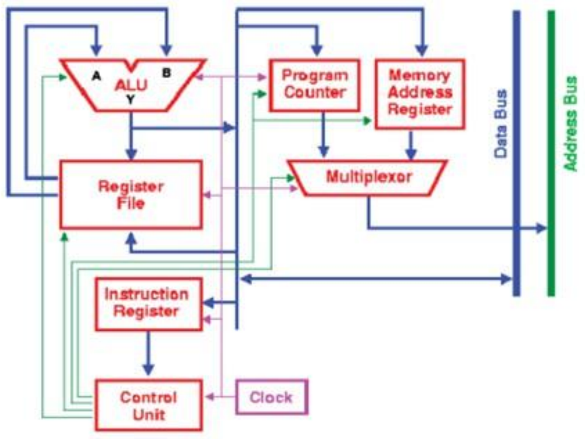


Fig.1: block diagram of processor

Program Counter points to the next instruction to be executed. In the complete instruction cycle, the instruction is loaded into the instruction register after the processor fetches it from the memory location which is pointed by the program counter. Control Unit is an essential part of any kind of computer or systems because this circuits generates the timing and control signals for the operations which is performed by the CPU. Here, the communication is between the ALU and the main memory as it controls the transmission of signals between the processor, memory and various buses.

RESULTS

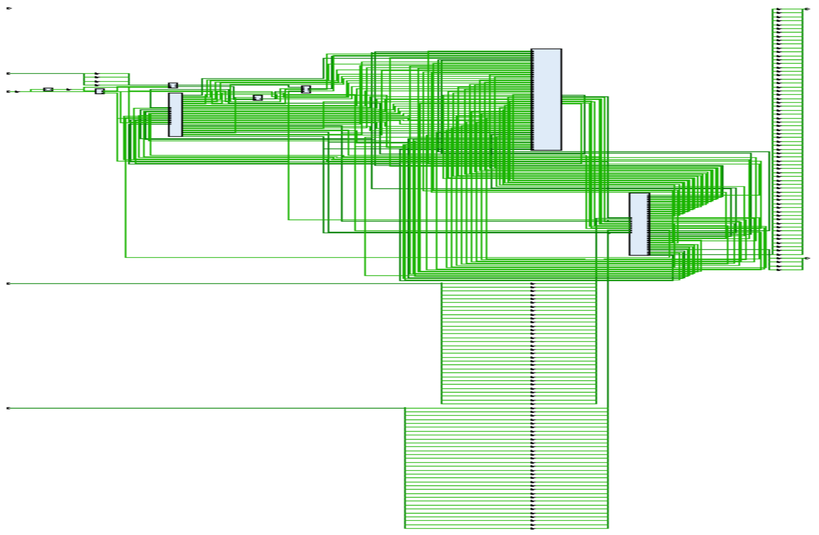
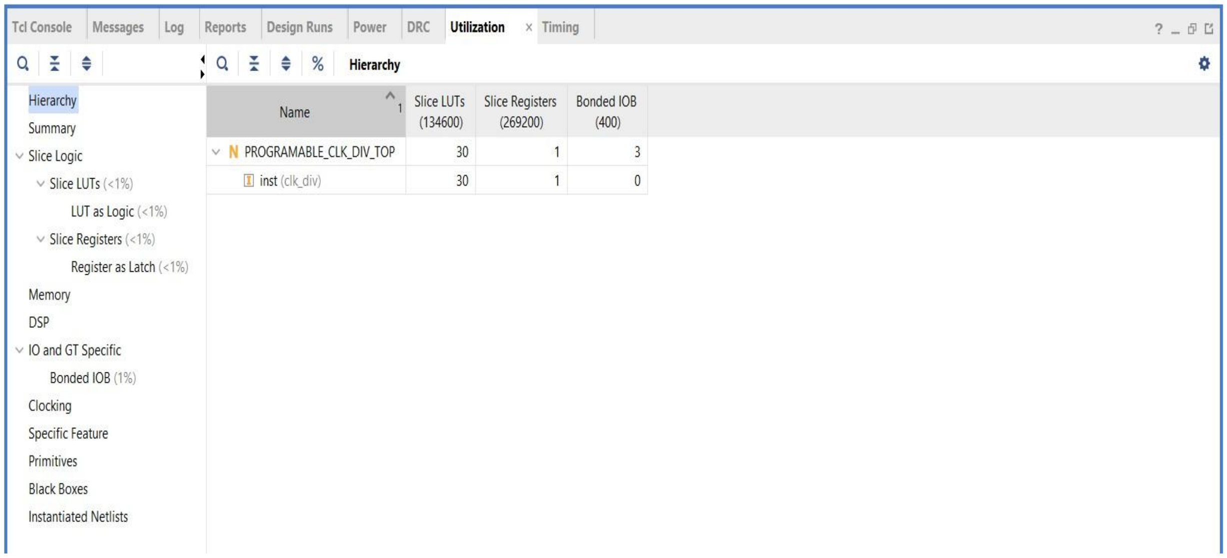


FIG: **RTL schematic:**



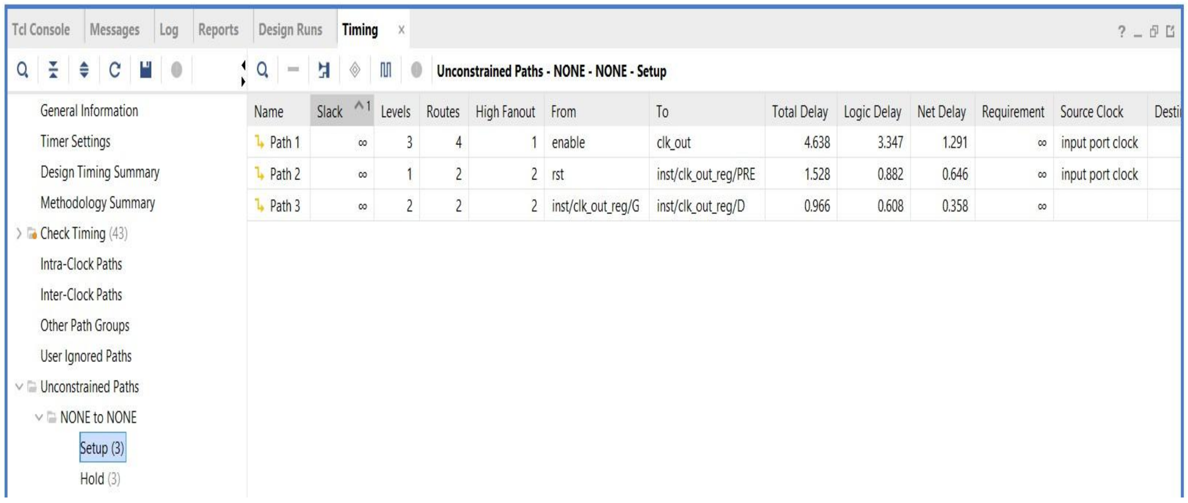
Tcl Console | Messages | Log | Reports | Design Runs | Power | DRC | Utilization | Timing

Hierarchy

- Hierarchy
- Summary
- Slice Logic
  - Slice LUTs (<1%)
    - LUT as Logic (<1%)
  - Slice Registers (<1%)
    - Register as Latch (<1%)
- Memory
- DSP
- IO and GT Specific
  - Bonded IOB (1%)
- Clocking
- Specific Feature
- Primitives
- Black Boxes
- Instantiated Netlists

Name	Slice LUTs (134600)	Slice Registers (269200)	Bonded IOB (400)
PROGRAMABLE_CLK_DIV_TOP	30	1	3
inst (clk_div)	30	1	0

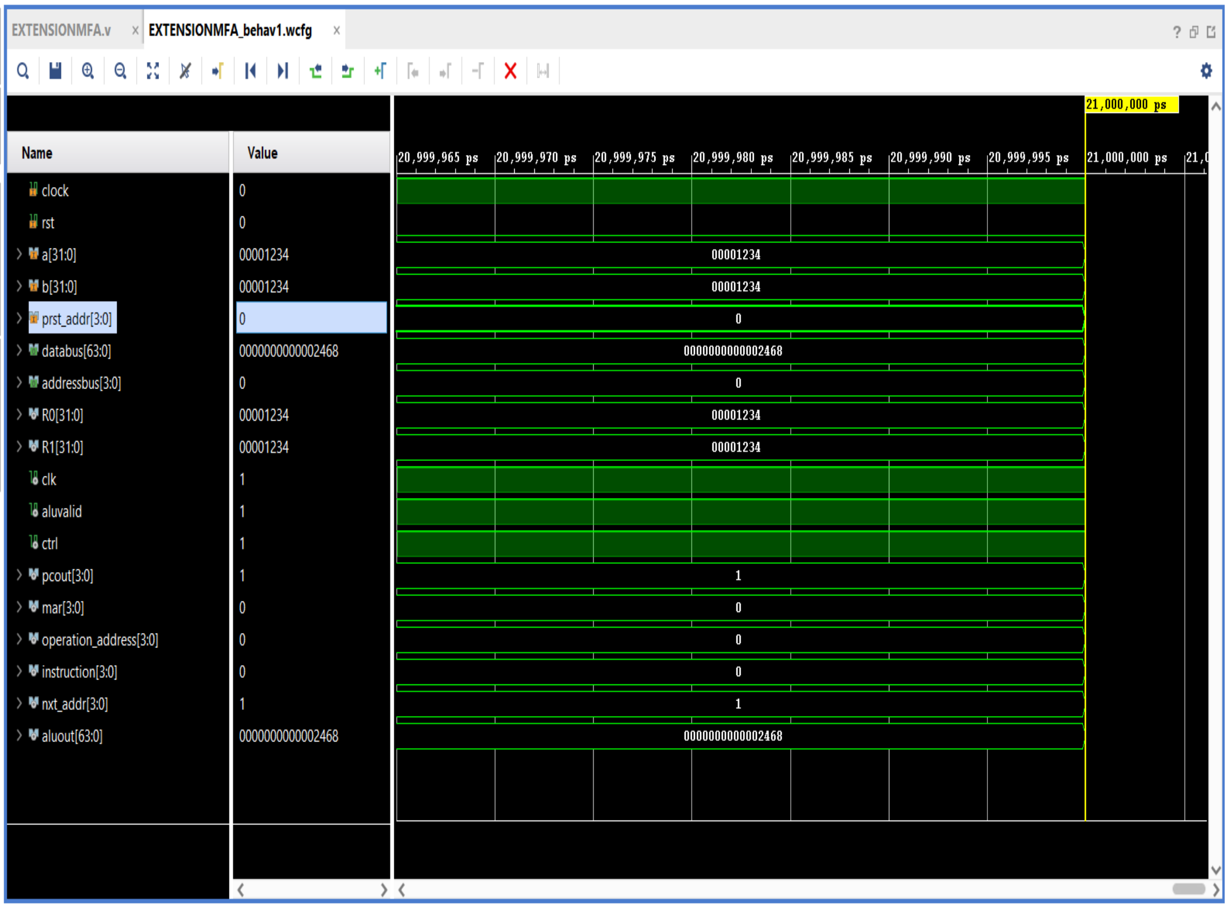
FIG-2:Utilization



General Information
Timer Settings
Design Timing Summary
Methodology Summary
Check Timing (43)
Intra-Clock Paths
Inter-Clock Paths
Other Path Groups
User Ignored Paths
Unconstrained Paths
NONE to NONE
Setup (3)
Hold (3)

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Desti
Path 1	∞	3	4	1	enable	clk_out	4.638	3.347	1.291	∞	input port clock	
Path 2	∞	1	2	2	rst	inst/clk_out_reg/PRE	1.528	0.882	0.646	∞	input port clock	
Path 3	∞	2	2	2	inst/clk_out_reg/G	inst/clk_out_reg/D	0.966	0.608	0.358	∞		

FIG-3: DEALY



Name	Value
clock	0
rst	0
a[31:0]	00001234
b[31:0]	00001234
prst_addr[3:0]	0
databus[63:0]	0000000000002468
addressbus[3:0]	0
R0[31:0]	00001234
R1[31:0]	00001234
clk	1
aluvalid	1
ctrl	1
pcout[3:0]	1
mar[3:0]	0
operation_address[3:0]	0
instruction[3:0]	0
nxt_addr[3:0]	1
aluout[63:0]	0000000000002468

FIG-4: SIMULATED OUTPUT

CONCLUSION

The 32-bit RISC processor designed in this project utilizes **minimal functional units** while adopting the **Harvard architecture** for separate instruction and data memory access. The design was implemented using **Xilinx Vivado 2023.1**, with synthesis results indicating a **minimum clock period of 2 ns**, ensuring efficient timing performance. Functional verification through simulation confirmed the accuracy of the processor's operations, validating its arithmetic, logical, and control functionalities

REFERENCES

[1] S. Lad and V. S. Bendre, "Design and Comparison of Multiplier using Vedic Sutras," 2019 5th International Conference On Computing, Communication, Control And Automation (ICCUBEA), Pune, India, 2019, pp. 1-5

[2] Balpande Vishwas V, Abhishek B. Pande, Meeta J. Walke, Bhavna D. Choudhari and Kiran R. Bagade. "Design and Implementation of 16 Bit Processor on FPGA." (2015).

[3] Seung Pyo Jung, Jingzhe Xu, Donghoon Lee, Ju Sung Park, Kang-joo Kim and Koon- shik Cho, "Design & verification of 16 bit RISC processor," 2008 International SoC Design Conference, Busan, 2008, pp. III-13-III-14, doi: 10.1109/SOCDC.2008.4815726.

[4] F. Adamec and T. Fryza, "Design — Time configurable processor basic structure," 13th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems, Vienna, 2010, pp. 119-120, doi: 10.1109/DDECS.2010.5491804.

[5] A. Bisoyi, M. Baral and M. K. Senapati, "Comparison of a 32-bit Vedic multiplier with a conventional binary multiplier," 2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies, Ramanathapuram, 2014, pp. 1757- 1760, doi: 10.1109/ICACCCT.2014.7019410.

[6] Mr. Nishant G. Deshpande, Prof. Rashmi Mahajan, "Ancient Indian Vedic Mathematics based Multiplier Design for High Speed and Low Power Processor", IJAREEIE, Pune, 2014

[7] Priyanka jain, Dr. G. S. Virdi, : Multiplier-Accumulator (MAC) Unit: International Journal of Digital Application & Contemporary Research ,Volume 5,

Issue 3, October 2016

[8] P. S. Mane, I. Gupta and M. K. Vasantha, "Implementation of RISC Processor on FPGA," 2006 IEEE International Conference on Industrial Technology, Mumbai, 2006, pp. 2096-2100, doi: 10.1109/ICIT.2006.372448.

[9] Ram, G. & Lakshmanna, Y. & Rani, D. & Kandula, Bala. (2016). Area efficient modified vedic multiplier. 1-5. 10.1109/ICCPCT.2016.7530294.

[10] Yogesh M. Motey, Tejaswini G. Panse, “Traditional and Truncation Schemes for Different Multipliers”, International Journal of Electronics and Computer Science Engineering, vol.2, no.2, pp.627-633, May 2013.

AUTHOR’S

K.G.VENKATA KRISHNA
Assistant Professor (C)
Department of Electronics and Communication Engineering
Krishna University College of Engineering and Technology
Krishna University
Rudravaram, Machilipatnam.
Ph.: 9966963399
EId: kgvk.aca.ece@kru.ac.in

ANUSHA BATHALA
UG Student
Department of Electronics and Communication Engineering
Krishna University College of Engineering and Technology
Krishna University
Rudravaram, Machilipatnam.
Ph:7893114097
Eid : anushabathala2005@gmail.com

ASIRI NAIDU KOYYANA
UG Student
Department of Electronics and Communication Engineering
Krishna University College of Engineering and Technology
Krishna University
Rudravaram, Machilipatnam.
Ph:6305873908
Eid : koyyanaasirinaidu6305@gmail.com

VAMSI KRISHNA REDDY.S
UG Student
Department of Electronics and Communication Engineering
Krishna University College of Engineering and Technology
Krishna University
Rudravaram, Machilipatnam.
Ph:8688287706
Eid: sunkarivamsikrishnareddy@gmail.com