Developing a Multi-Threaded RISC-V Processor for Real-Time Processing Applications

GAJA CHANDANA SRI
Department of ECE Institute of Aeronautical Engineering
*Telangana, India 21951A0438@iare.ac.in 0009-0006-2797-3517*MANDULA SRAVANA SREE
Department of ECE Institute of Aeronautical Engineering
Telangana, India 21951A0438@iare.ac.in 0009-0004-0832-8155

VAKKAPATLA VAMSI
Department of ECE Institute of Aeronautical Engineering
Telangana, India 21951A04P8@iare.ac.in 0009-0008-9480-1736

MRS. M. SREEVANI
Assistant Professor Department of ECE
Institute of Aeronautical Engineering Telangana, India m.sreevani@iare.ac.in

Abstract:

The popularity of RISC-V has been growing despite its intrinsic inability to support multithreaded mode. Hence, there have been research efforts toward the development and implementation of multithreaded RISC-V processors. This paper describes a multithreaded Reduced Instruction Set Computer (RISC) processor for improved performance and increased operation speed. It can execute a larger number of instructions with a simple design and lower critical path delay. RISC architecture allows dynamic control of threads with create, execution, halt, delete, etc., operations to reflect the operation of a real-time operating system task queue. We add specific instructions on the top of RISC-V to support these thread control operations. That leads us to the implementation of a multithreaded RISC-V processor with prioritized execution of programs across eight threads and concurrent execution of instructions execution beyond the number of its cores. This capability enables the processor to execute the number of threads greater than its number of cores in parallel. In order to be flexible, scalable, and also low-cost, this design leverages the open-source RISC-V \instruction set architecture (ISA) for a wide range of hard-real-time applications, from industrial automation to \autonomous systems. The anticipated real-time performance along with the function of the proposed multithreaded RISC-V processor is, showing significant improvements in throughput as well as latency, making it a sturdy solution for the dynamic demands of modern real-time systems.

Keywords: RISC, ISA, Inclusive Creation, Context Switching, Strict Timing.INTRODUCTION

1.1 Overview of RISC Processor

In conventional approach the system consumes too much of power. The power reductions in conventional RISC processors are done at fabrication step itself, but which is too complex process. Here utilization of chip area is more and the system consumes more

power which leads to increased latency. RISC architecture is designed with less number of gates. Low power consumption helps to reduce the heat dissipation, lengthen battery life and increase device reliability. This technology strongly affects battery, electronic packaging of ICs, heat dissipation. Low power embedded processors are used in a wide variety of applications including cars, mobile phones, digital cameras, printers emerged in today's electronics.

The need for low power has caused a major paradigm shift where power dissipation has become an important consideration as performance and area. RISC is termed as Reduced Instruction Set Computer. Low power design of 32-bit data width RISC processor and also a high speed floating point double precision addition, subtraction, multiplication and division operations are described, which are implemented using pipelined architecture. Through this, one can improve the speed of the operation as well as overall performance. In this design, the pipelining technique consists of four stages. They are Fetch, Decode, Execute and Memory Read/Write.

The architecture doesn't need any control hazards, as auto branch prediction is happening in the Fetch stage. Without branch prediction, the processor has to wait until the conditional jump has passed the execute cycle before the next instruction can enter the fetch stage in instruction pipeline. The branch prediction part to be the most likely is then fetched and speculatively executed. This will increase flow in instruction pipeline and achieve high effective performance. During the design process various low power techniques in architectural level are included. It has a complete instruction set, program and data memories, general purpose registers and a simple Arithmetical Logical Unit (ALU) including Floating Point operations. In this design, most instructions are of uniform length and similar structure.

1.2 Architecture of RISC Processor

The architecture of the proposed low power pipelined 32-bit RISC processor with FPU is a single cycle pipelined processor. It has small instruction set, load/store architecture, fixed length coding and hardware decoding and large register set. This is a general-purpose 32-bit RISC processor with pipelining architecture. It gets instructions on a regular basis using dedicated buses to its memory, executes all its native instruction in stages with pipelining. In the low power RISC design, all the arithmetic, branch, logical and floatingpoint arithmetic (add, sub, mul and div) operations are performed and the resultant value is stored in the memory/register and retrieved back from memory, when required. In the design, power reduction is done in front end process so that low power RISC processor is designed without any complexity.

The system architecture of a low power pipelined 32-bit RISC processor with FPU is shown in Fig. l. The architecture comprises of Modified Harvard Architecture, low power unit and floating unit. The Modified Harvard architecture consists of four stage pipelining: Instruction Fetch, Instruction Decode, Execution Unit and Memory Read/Write. Pipelining technique allows for simultaneous execution of parts or stages of instructions more efficiently. With a RISC processor, one instruction is executed while the next is being decoded and its operands are being loaded while the following instruction is being fetched at the same time. Pipelining would not flush when branch instruction occurs as it is implemented using dynamic branch prediction. The branch prediction attempts to avoid the waste of time whether the conditional jump is most likely to be taken or not taken. In the low power RISC design, all the arithmetic, branch, logical and floating point arithmetic (add, sub, mul and div) operations are performed and the resultant value is stored in the memory/register and retrieved back from memory, when required.

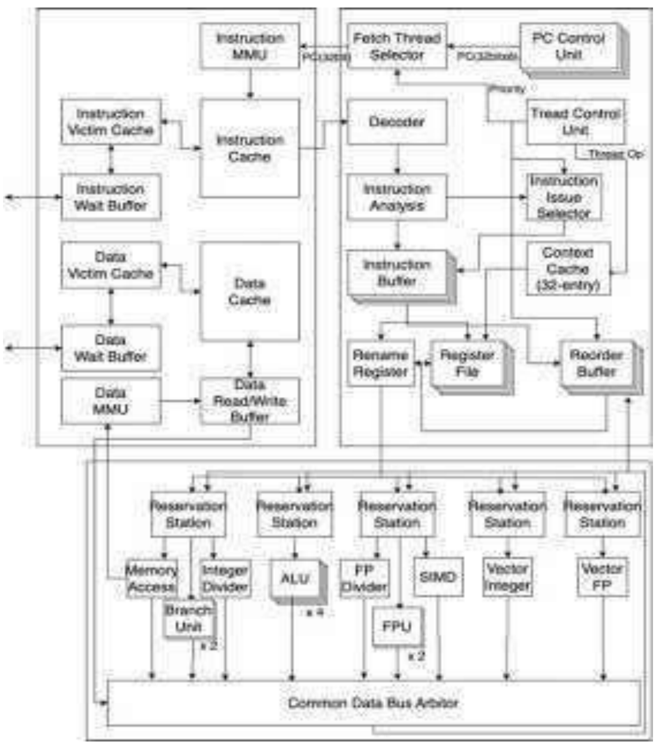


Fig. 1 Architecture of RISC Processor

I. LITERATURE SURVEY

Reduced Instruction Set Computing (RISC) is based on the assumption that the simplification of the instruction set leads to the performance gain through a reduction of the processor cycles per instruction. A pre- concept of the RISC technology was invented already in the 1960s and 70s, but it gained popularity in the 1980 through projects at Stanford and Berkeley, which developed successful architectures like MIPS and SPARC. RISC designs, of course, ARM and PowerPC, were everywhere in Unix workstations, embedded devices, and the more recent mobile systems: smartphones and tablets.

The ancestry of RISC traces back to the 1960s with Seymour Cray's CDC 6600, which used a load/store architecture and greatly simplified addressing modes. However, the term "RISC" was actually originated by David Patterson during the Berkeley RISC project. Others credit the first actual RISC system to IBM's 801 design, led by John Cocke in 1975; this was not a commercial success. In 1980, the Berkeley RISC project, which was funded by DARPA's VLSI Program, notably took RISC technology significantly forward with features such as pipelining and register windowing.

2.1 CISC vs RISC

Design of a processor's instruction set has played a significant role in computer architecture and thereby in the way machine language programs are constructed. The early computers have very simple instruction sets because the hardware did not allow for a sophisticated instruction set, but with the advent of integrated circuits, which became cheap, the size of the instruction set began to grow, and their complexity, as well. Some of today's computers have more than 200 instructions in their instruction set. They can employ several data types and addressing modes. This design trend towards increasing complexity was known as a ComplexInstruction Set Computer (CISC), which was supposed to emulate more application-specific functions at the hardware level. In the 1980s, however, designers proposed something called Reduced Instruction Set Computers (RISC), which utilized fewer, simpler instructions to execute more quickly inside the CPU and reduced dependence on memory.

2.1.1 CISC

The design of an instruction set for a computer must take into account, besides machine language constructs, the requirements imposed on the use of high-level programming languages. Translation from high-level to machine language programs is carried out by means of a compiler program. The job of a compiler is to produce a sequence of machine instructions for each high-level statement. The job is simplified if there exists a corresponding machine instruction for each statement. A CISC architecture basically seeks to attempt to provide a machine instruction for every high-level language statement.

Some operations involving register operands can be represented in a two-byte instruction, whereas memory-addressed instructions could demand up to five bytes. This introduces variability with special decoding circuits that need to decode instructions of various lengths using a fixed memory word size. Direct memory manipulation in CISC processors allows instructions such as ADD to reference memory for the operands and the result, resulting in several memory accesses at the time of executing the instruction. While this additional flexibility makes high-level language compilation easier, the added complexity of addressing modes and instruction-sets translates to additional logic hardware that slows down computations because the system has to manage these operations.

2.1.2 RISC

The small set of instructions of a typical RISC processor consists mostly of register-to- register operations, with only simple load and store operations for memory access. Thus each operand is brought into a processor register with a load instruction. All computations are done among the data stored in processor registers. Results are transferred to memory by means of store instructions. This architectural

feature simplifies the instruction set and encourages the optimization of register manipulation. Other addressing modes may be included, such as immediate operands and relative mode. By using a relatively simple instruction format, the instruction length can be fixed and aligned on word boundaries. An important aspect of RISC instruction format is that it is easy to decode. Thus the operation code and register fields of the instruction code can be accessed simultaneously by the control. By simplifying the instructions and their format, it is possible to simplify the control logic. For faster operations, a hardwired control is preferable over a microprogrammed control.

A characteristic of RISC processors is their ability to execute one instruction per clock cycle. A load or store instruction may require two clock cycles because access to memory takes more time than register operations. Efficient pipelining, as well as a few other characteristics, are sometimes attributed to RISC, although they may exist in non-RISC architectures as well. Other characteristics attributed to RISC architecture are: A relatively large number of registers in the processor unit Use of overlapped register windows to speed-up procedure call and return efficient instruction pipeline. A large number of registers is useful for storing intermediate results and for optimizing operand references. Thus register-to-memory operations can be minimized by keeping the most frequent accessed operands in registers. For this reason a large number of registers in the processing unit are sometimes associated with RISC processors.

II. METHODOLOGY

3.1 INTRODUCTION

Real-time systems are a critical component of any industrial applications such as -automotive aerospace industrial automation technology and tele communication and health care. These applications require predictable and bounded real-time behavior, because delays even for very small periods can result in system failure or unsafe conditions. General computing applications is a highly efficient application that truly falls behind in real-time applications due to their inability to efficiently execution of multiple concurrent processes. With the increase in complexity and demands of real- time systems, there arises a need for advanced processor architectures that can satisfy strict requirements.

The open-source instruction set architecture (ISA) RISC-V is extremely flexible and customizable and thus acts as an ideal platform in the development of specialized processors. Its open-source nature has led to increased interest and adoption across academia and industrial research communities, with new perspectives on processor design opening up. The modularity of the RISC-V ISA makes it feasible to include only those functionalities that optimize the processor for high performance, low power, and low area-these three parameters are critically important while working in real-time systems.

Multithreading is a very powerful technique, as it allows a single processor core to execute more than one thread at any instant of time. It distributes workload in multiple threads, a multithreaded processor could deliver improved latency and higher throughput, thus a more dependable enforcer of real-time constraints. Recently, research in design and implementation of RISC-V multithreaded processors for a real-time system has picked up speed due to various hurdles presented to it by a single-threaded processor that violates the requirements of modern applications. It is also noteworthy that research into the development of a multi-threaded RISC-V processor also has huge cost-saving and scalability. Open- source license removes licensing fees, making it highly appealing to cost-sensitive applications. Furthermore, since the architecture of RISC-V is scalable, designers can implement processors ranging from relatively simple embedded cores to more complex high-performance systems. The scalability is a need of real-time systems due to the fact that requirements are vastly different based on an application domain. The multithreaded processor RISC-V can be configured for it has the advantage of meeting specific real-time requirements by offering a versatile, low-cost solution.

3.2 RISC Blocks and Working

In the present work, the RISC processor consists of blocks namely, Instruction Fetch (PC), Control Unit, Register File, Arithmetic & Logical Unit (ALU), Floating Point Unit and Memory Unit.

3.2.1 Instruction Fetch

The current processor architecture incorporates a Program Counter (PC) and dynamic branch prediction to enhance instruction fetching efficiency. The PC performs two main operations: incrementing to point to the next instruction and loading a new address when a branch instruction is executed. It typically increments by one each clock cycle, but the increment is modified by the branch offset during a branch instruction.

In the Instruction Fetch stage, dynamic branch prediction minimizes branch penalties by using a branch prediction buffer indexed by the lower-order bits of the branch address. This buffer indicates whether a branch is taken or not and facilitates quick access to the branch target address through a Branch Target Cache (BTC), which stores the target address along with an address tag. If the predicted branch is taken, the BTC allows the processor to initiate the next instruction access promptly, ensuring efficient operation.

3.2.2 *Control Unit*

The control unit generates all the control signals needed to control the coordination among the entire component of the processor. This unit generates signals that control all the read and write operation of the register file and the data memory. It is also responsible for generating signals that decide when to use the multiplier and when to use the ALU. It generates appropriate branch flags that are used by the Branch Decide unit.

3.2.2 *Register File*

This is a two-port register file which can perform two simultaneous read and write operations. It contains four 64-bit general purpose registers. These register files are utilized during the arithmetic, data instructions and floating- point operations. It can be addressed as both source and destination using a 2-bit identifier. The registers are named as RO through R3. The load instruction is used to load the values into the registers and store instruction is used to hold the address of the corresponding memory locations. When the Reg_Write signal is high a write operation is performed to the register.

3.2.3 *Arithmetic Logic Unit*

The ALU is responsible for arithmetic and logic operations that take place within the processor. These operations can have one operand or two, these values coming from either the register file or from the immediate value from the instruction directly. The operations supported by the ALU include add, sub, compare, increment, AND, OR, NOT, NAND and NOR.

The output of the ALU goes either to the data memory or through a multiplexer back to the register file. The multiplier is designed to execute in a single cycle instruction. All operations will be done according to the control signal coming from ALU control unit. Control unit is responsible for providing signals to the ALU that indicates the operation that the ALU will perform.

3.2.4 *Memory Unit*

The load and store instructions are used to access this module. Finally, the memory access stage is where, if necessary, system memory is accessed for data. Also, if a write to the data memory is required by the instruction it is done in this stage. In order to avoid additional complications, it is assumed that a single read or write is accomplished within a single CPU clock cycle.

3.2.5 *Instruction Set*

The instruction set used in this architecture consists of arithmetic, logical, memory and branch instructions. It will have short (8-bit) and long (16- bit) instructions, which are shown in Table 1. For all arithmetic & logical operations, 8- bit instructions are used. For all memory transactions and jump instructions, 16-bit instructions are used. It will have special instructions to access external ports.For all the jump instruction, the processor architecture will automatically flush the data in the pipeline, so as to avoid any misbehavior.

INSTRUCTION SET TABLE:

OPCODE	SOURCE	DESINATION
1010	10	11

Table: 1 Short Instruction Format

OPCODE	SOURCE	DESTINATION
0011	00	-
ADDRESS		
0101	11	01

Table: 2 Long Instruction Format

3.2.6 *Low Power Technique*

There are several different RTL and gate-level design strategies for reducing power. In the present work, Clock Gating design is used for reducing dynamic power. In this method, clock is applied to only the modules that are working at that instant. Clock gating is a dynamic power reduction method in which the clock signals are stopped for selected registers banks during the time when the stored logic values are not changing. The clock pulse for low power technique is shown in Fig. 2. The input to low power unit is global clock and its output is gated clock, since the module will block the main clock in the following conditions. When instruction is halt. When there is a continuous Nop operation. When program counter fails to increment.

.

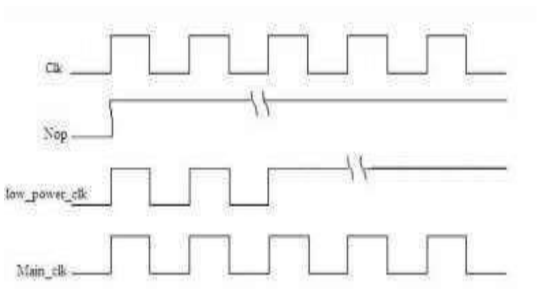


Fig: 2 Clock Pulses of Low Power Unit

3.2.7 *Floating Point Unit*

A floating point (FPU), also known as a math co-processor or numeric processor is a specialized co-processor that manipulates numbers more quickly than the basic microprocessor circuitry. Floating point computational logic has long been a mandatory component of high performance computer systems as well as embedded systems and mobile applications. The performance of many modern applications which give a high frequency of floating point operations is often limited by the speed of the floating point hardware. The advantage of floating point representation over fixed point and integer representation is that it can support a much wider range of values.

FP _Add:

In the module FP _Add, the inputs operands are separated into their mantissa and exponent components. Then the exponents are compared to check which variable is larger. The larger variable goes into "mantissaJarge" and exponent_large". Similarly the smaller variable goes into "mantissa_small" and "exponent_small".

FP _Sub:

The input variables are separated into two components namely mantissa and exponent. Subtraction is similar to that of addition such that the mantissa of the smaller exponent is shifted to the right before performing the subtraction.

FP _ Mul:

Multiplying all 53 bits of varl by 53 bits of var2 would result in a 106-bit product. 53 bit by 53 bit multipliers are not available in the Altera FPGAs, so the multiply would be broken down into smaller multiplies and the results would be added together to give the final 106-bit product. The module (FP _ Mul) breaks up the multiply which can perform 24-bit by 17-bit.

FP_Div:
Division is performed in FP _ Div. The exponent is obtained by adding 1023 with the exponent of var1 and then by subtracting the exponent of var2 from the sum.

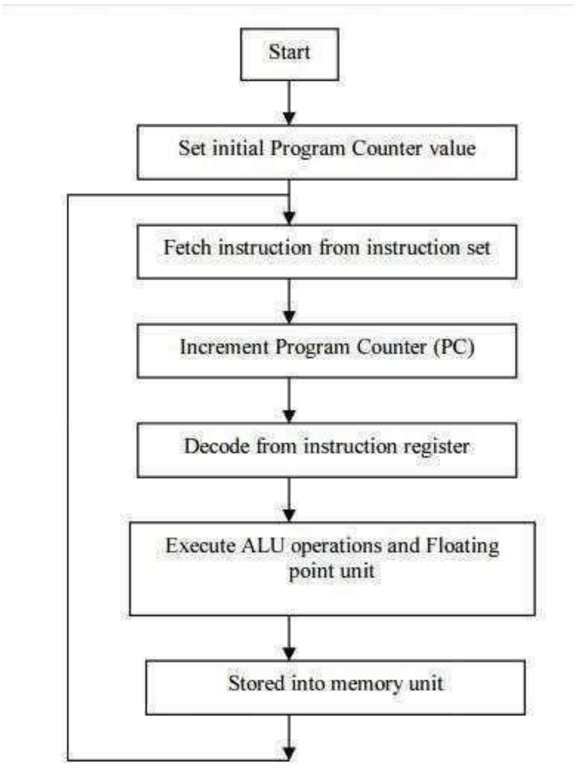

Start
Set initial Program Counter value
Fetch instruction from instruction set
Increment Program Counter (PC)
Decode from instruction register
Execute ALU operations and Floating point unit
Stored into memory unit

Fig: 3 Flow Chart of Processor

3.3 INTRODUCTION TO XILINX

3.3.1 To Migrate a Project

- From the ISE 12 Project Navigator, select File > Open Project.
- In the Open Project dialog box, select the .xise file to migrate.

Note: You may need to change the extension in the Files of type field to display

.npl (ISE 5 and ISE 6 software) or

.ise (ISE 7 through ISE 10 software) project files.

• In the dialog that opens, select Backup and Migrate or Migrate Only. The ISE software automatically converts your project to an ISE 12 project.

Note: If you selected Backup and Migrate, a backup of the original project is saved at project_name_ise12migration.zip.

3.3.2 Run the design with the new software.*Using the Project Browser:*

To Copy a Project:

1. Select File > Copy Project.

2. In the Copy Project dialog box, enter the Name for the copy.

Note that a name for the copy can be equal to the name of project, while making it in other location.

3. Specify a directory Location where the copied project will be stored

4. Optionally, you enter a Working directory.

This is by default empty, and it defaults to the same as the project directory. You can specify a working directory so that you keep your ISE project file somewhere - separated from your working area.

5. OPTIONAL: Provide a Description for the copy. The description can help you identify dominant characteristics of the project to reference later.

6. Source options area:

Choose one of the following:

-leave the design source files in their current location. Leave sources in place

If you select this option, the copied project references the files at their current location. If you make changes to the files in the copied project, those changes are also apparent in the original project because the source files reside in both projects.

opy sources to the new location C

To create a copy of all design source files and store them in the specified Location directory.

If you edit the files in the copied project, the changes do not appear in the original project because source files are not shared by two projects. Optionally select Copy files from Macro Search Path directories to copy files from the directories that you specify in the Macro Search Path property in the Translate Properties dialog box.

Copy all the files from the referenced folders, not just files in use by the design. Note: If you added a net list source file directly to the project as described in Working with Net list-Based IP, the file is automatically copied as part of Copy Project because it is a project source file. Preferred way to add netlist source files to your design is through the adding of netlist modules since the files will automatically be dealt with by Project Navigator.

3.3.3 Project Archive

A project archive is a single, compressed ZIP file with a .zip extension that includes all your project files, sources, and files generated, unless otherwise specified.

These include:

- User-added sources and associated files
- Remote sources
- Verilog `include files
- Files in the macro search path
- Files generated

3.3.4 To Archive a Project:

- Project > Archive
- In the Project Archive dialog box, specify a file name and directory for the ZIP file.
- Check Exclude generated files from the archive to eliminate generated files and non- project files from the archive.
- OK.

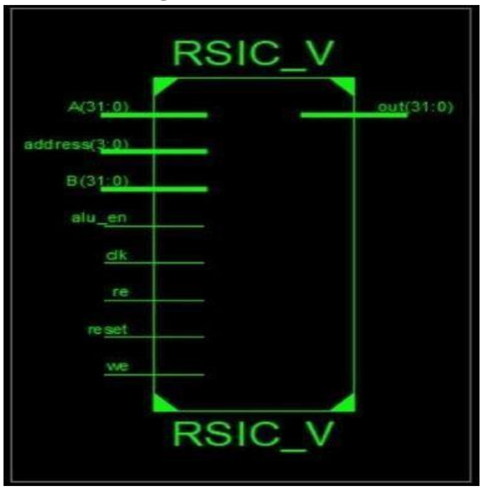
A ZIP file is created in the chosen directory. In order to unpack the archived project, the ZIP file needs to be first unpacked, and then the project needs to be unpacked. Note: Sources, that are not located inside the project directory, are copied into a remote_sources subdirectory within the archive of the project. By following the above procedure one can archive a project. The archived project must be stored in a zip file which is unpacked.*Properties*

Xilinx FPGAs feature extremely high performance, supporting a high rate of data processing with complex computations.

- Support for high-speed I/O standards such as PCIe, Ethernet, and memory interfaces like DDR, QDR
- They supply their products from low-cost, low-power devices to high-end, high- performance devices targeted at a variety of application needs
- Power optimization techniques and options are integrated in the device as Xilinx devices indeed are power sensitive
- Security is integrated into the Xilinx devices with the help of bit-stream encryption.

III. RESULTS

The design and implementation of a multithreaded RISC-V processor for real-time systems have yielded promising results, demonstrating significant improvements in performance and resource utilization. By leveraging the modularity and flexibility of the RISC-V architecture, the processor effectively manages multiple concurrent threads, allowing for efficient handling of high-priority tasks and real-time constraints. The multithreading capabilities enable seamless context switching and optimized thread synchronization, resulting in reduced latency and improved throughput for time-sensitive applications. The open-source nature of RISC-V has facilitated cost-effective development, making the processor an attractive solution for various industries, including automotive, aerospace, and healthcare. Testing under real-world scenarios has confirmed that the multithreaded design can meet stringent timing requirements while ensuring deterministic execution, thereby enhancing the reliability and safety of real-time systems. Overall, the successful implementation of this multithreaded RISC-V processor underscores its potential to revolutionize real-time computing applications, providing a scalable and efficient platform tailored to meet evolving demands.



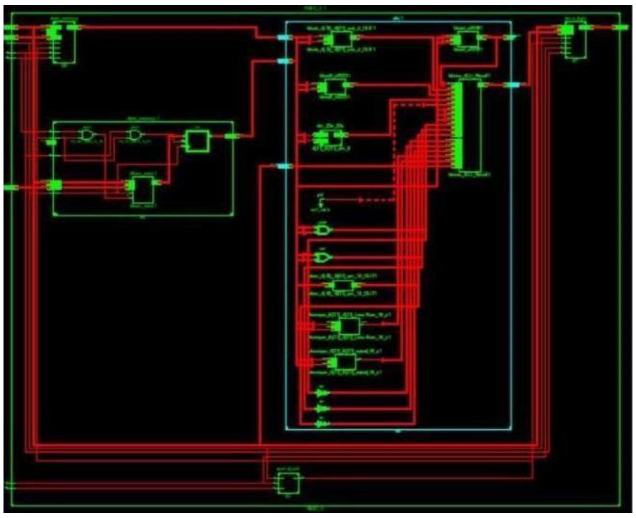
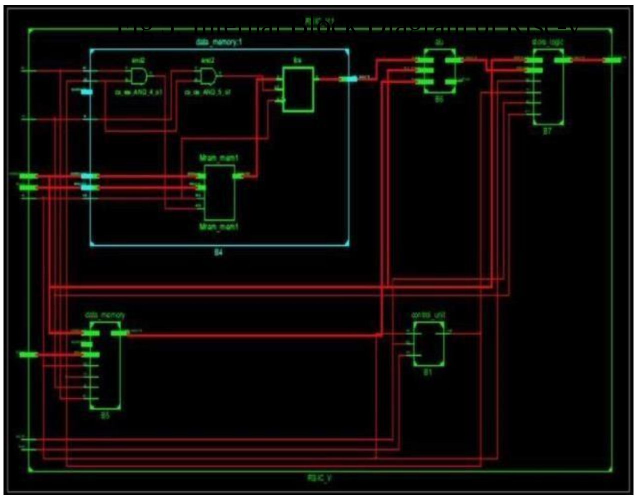


Fig 4: Block Diagram of RISC-V



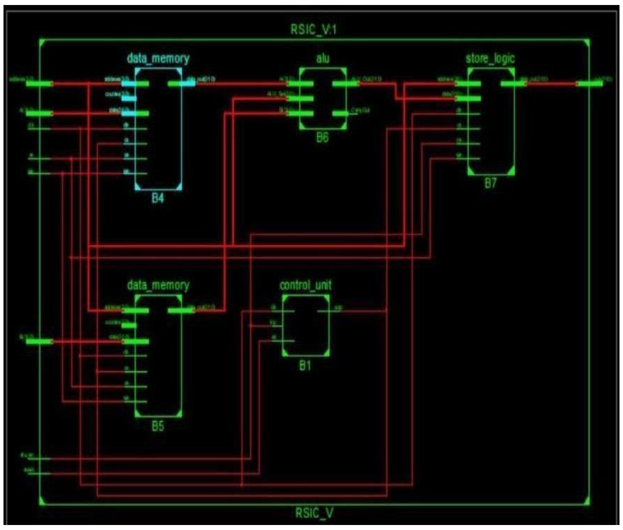


Fig 6: Internal Block Diagram of Data Memory

Fig 7: Internal Block Diagram of ALU

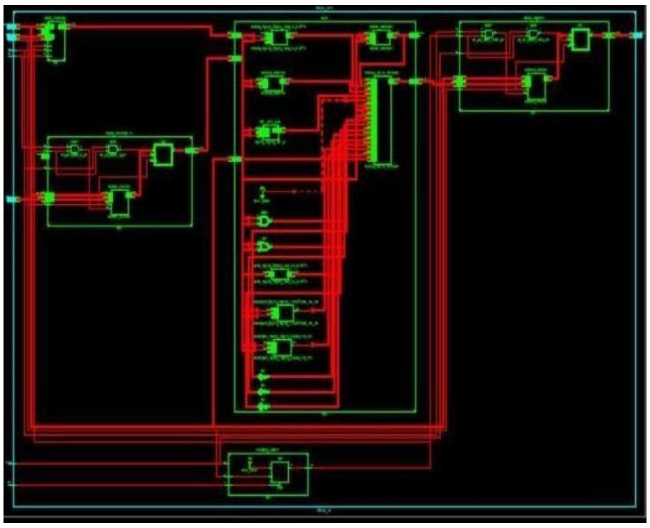


Fig 8: Internal Block Diagram of Store Logic

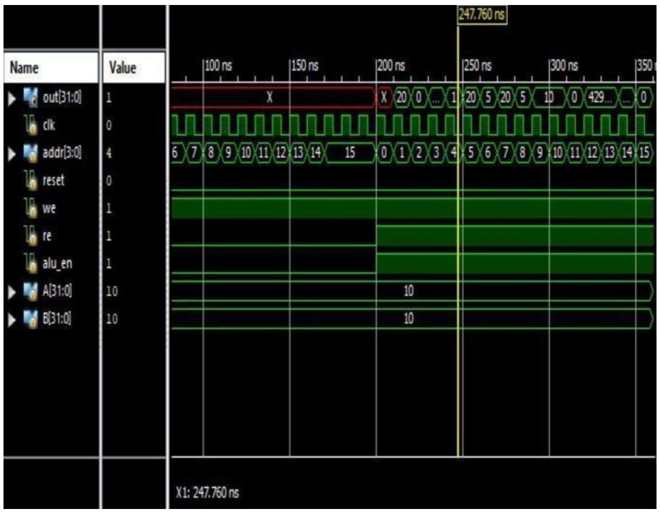


Fig 9: Output of Multithreaded RISC-V Processor

I. CONCLUSION

Low power pipelined 32-bit RISC processor with Double Precision Floating Point is designed. The design and implementation of a multithreaded RISC-V processor for real-time systems show great promise in enhancing performance and efficiency. By allowing multiple tasks to run simultaneously, this processor reduces delays and makes better use of resources, which is essential for meeting strict timing requirements. Modalism is used to verify the simulation results. The design is implemented on Altera DE2 FPGA on which Arithmetic, Branch operations and Logical functions are verified. Pipelining would not flush when branch instruction occurs as it is implemented using dynamic branch prediction. Branch predictions will increase flow in instruction pipeline and achieve high effective performance.

The proposed architecture is able to prevent pipeline to multiple executions with a single instruction. Whenever the processor enters in sleep mode, then it disables the clock enable signal so this saves some power by using low power technique. The proposed design can access more data processing for data intensive applications like packet processing. This 32-bit RISC processor consumes only 1 instruction, whereas 32-bit RISC processor needs more than 1 instruction. This processor with floating point operations is used in many applications like Signal processing, Graphics and Medical equipment's.
The flexible architecture of RISC-V allows for cost-effective solutions that are specific to the different applications available in various industries. To put simply, this multithreaded processor presents a reliable and responsive platform that can significantly enhance real-time computing, thus becoming of great value in the development of future technology. Taken together, the inherent simplicity and scalability of the RISC-V architecture with robust multithreading capabilities, the processor effectively deals with concurrent tasks in a manner that supports stringent.

In the meantime, this implementation benefits performance and power efficiency. but is flexible enough for future upgrades and adaptation. Integration of real-time features like precise interrupt handling and reliable mechanisms for controlling time ensures the processor can support a wide range of applications in various real- time embedded systems.

II. FUTURE SCOPE

The development of a multi-threaded RISC-V processor for real-time systems has vast potential across various technology fields. In IoT, edge computing, and AI**,** such processors enable highly efficient multi-tasking, allowing devices to process data in real time with minimal latency. This is crucial for applications like smart devices **and** machine learning at the edge, where quick response times are essential. The open-source nature of RISC-V allows it to be tailored for optimized power consumption and performance, making it ideal for IoT and AI deployments. With these advantages, RISC-V can effectively support real-time applications in devices that operate on limited power but require fast data processing capabilities.

In sectors like **automotive** and **aerospace**, multi-threaded RISC-V processors show promising benefits in safety-critical applications such as Advanced Driver Assistance Systems (ADAS) and radar processing. By providing faster response times and enabling reliable concurrent processing, these processors help improve both safety and operational performance in autonomous systems. This is especially relevant in the automotive industry, where real-time decision-making can support vehicle navigation, obstacle detection, and sensor fusion. Similarly, in aerospace, RISC-V's customizable architecture and multi-threading capabilities can power systems that require secure and rapid data handling, such as navigation and radar.

The **telecommunications** industry, particularly with the rollout of 5G, stands to benefit from multi- threaded RISC-V processors as well. In 5G networks,

real-time processing is critical for managing multiple, simultaneous data streams and ensuring seamless data throughput. RISC-V's adaptability also positions it well for **healthcare** applications, where real-time monitoring and data analysis are essential in medical devices and wearables for patient health tracking and critical alert systems. In **cloud computing and data centers**, these processors can support high-throughput and workload optimization, particularly in data- intensive tasks. The modular and open-source design of RISC-V makes it versatile, allowing these processors to be customized and optimized across industries to meet specific real-time demands.

III. REFERENCES

1. https://ieeexplore.ieee.org/document/104098
49 “Design of a Multithreaded RISC-V Processor for Real-Time Systems” - 2023 Eleventh International Symposium on Computing and Networking Workshops (CANDARW)

2. Preetam Bhosle, Hari Krishna Moorthy,"FPGA Implementation of Low Power Pipelined 32 bit RlSC Processor", Proceedings of International Journal of Innovative Technology and Exploring Engineering (IJITEE), ISSN: 2278 3075, Vol- I, Issue-3, August 2012.

3. Galani Tina G,Riya Saini and R.D.Daruwala,"Design and Implementation of 32-bit RlSC Processor using Xilinx",lnternational Journal of Emerging Trends in Electrical and Electronics(IJETEE),ISNN:2320-9569,Vol 5,lssue I ,July-20 13.

4. http://elearning.vtu.ac.in/12/enotes/Adv_Com
_ArchlPipeline/Unit2- KGM.pdf

5. http://en.wikipedia.org/wiki/Classic_RISC pipeline

6. IEEE Transactions on Circuits I: Regular Papers, vol. 69, no. 2, pp. 735-745, Feb. 2022

1. Imran Mohammad, Ramananjaneyulu, "FPGA Implementation of a 32-bit RlSC Processor Using VHDL", Proceedings of International Journal of Reconfigurable (IJRES),ISSN:2089-4864, Vol-l, No.2, July 2012.

2. Aboobacker Sidheeq.V.M,"Four Stage Pipelined 16 bit RlSC on Xilinx Sparatn 3AN FPGA", Proceedings of International Journal of Computer Applications, ISNN: 0975 888,Vol-48, June 2012.

3. http://en.wikipedia.org/wikilBranch redictor

4. http://en.wikipedia.org/wiki/Double-precision _ floating- point_ format.

5. Tashfia.Afreen, Minhaz. Uddin Md Ikram, Aqib. AI Azad, and Iqbalur Rahman Rokon," Efficient FPGA Implementation of Double Precision Floating Point Unit Using Verilog HDL", International Conference on Innovations (ICIEE'20 12),October 20 12,Dubai (UAE).

6. Addanki Purna Ramesh, Ch. Pradeep, "FPGA Based Implementation of Double Precision Floating point Adderl Subtarctor Using Verilog", International Journal Proceedings of of Emerging Technology ISSN-2250-2459,Vol-2,lssue 7,July 2012.

7. J. Ravindra, T. Anuradha, "Design of Low Power RlSC Processor by Applying Clock gating Technique", International Journal of Engineering Research and Applications, ISSN2248 9622, Vol-2, Issue-3, May-Jun 2012.

8. N. Yamasaki, “Responsive Multithreaded Processor for distributed real time control,” The 8th IEEE International Workshop on Advanced Motion Control, 2004. AMC ’04., Kawasaki, Japan, 2004, pp. 457-462.

9. I. Tanaka, T. Tanaka, R. Higashi, T. Sekibe, S. Takada, and H. Nakajo, “Implementation of a RISC-V SMT Core in an AI processor,” 11th In ternational Symposium on Information and Communication Technology (SoICT ’22). Association for Computing Machinery, New York, NY, USA, 2022, pp. 15–22.

10. Y. Eni, S. Greenberg and Y. Ben-Shimol, “Efficient Hint- Based Event (EHE) issue scheduling for hardware multithreaded RISC-V pipeline,” in