# Implementation of Application Specific Instruction set Processor for Approximate Computing

AKSHAY DEOLE, VANITA AGARWAL, VAISHALI INGALE
Electronics and Telecommunication Department,
COEP Technological University,
Pune, Maharashtra,
INDIA

*Abstract:* - The performance, energy efficiency, power dissipation, and cost of the processors used in Internet of Things edge devices can be improved by introducing approximate computing. In this paper, the design and synthesis of an application-specific instruction set processor for approximate computing is proposed. Approximate computing can also be defined by using Artificial Intelligence-based technique like the Fractal theory and Inconsistent Information System. Application Specific Instruction Set Processor is a new processor design area and is used to design a new instruction set using existing processor configuration available on the EDA tool. From the results, it is observed that significant improvement is achieved in processor performance using the TIE application as compared to without the TIE application which is applied to user-designed instruction.

*Key Words*: - Approximate Computing, ASIP, FPGA, Internet of Things (IoT), Smart City, TIE Language, and Tensilica Tool.

Received: November 19, 2023. Revised: June 21, 2024. Accepted: July 16, 2024. Published: August 14, 2024.

## 1 Introduction

The development of electronic commodities usually uses the path: of accomplishing more functions, do them faster, and diminishing the cost. It creates huge pressure on the design teams. The performance of the product is improved by two factors, one is computational complexity which is an important parameter for hardware design and the second is data rates for data transfer, [1]. Due to this, energy consumption is an increasing concern in many computer systems and other electronic products. Much of the focused concern with reducing energy consumption has been on low-power architectures, performance/power tradeoffs, and resource management. Presently, almost all processors use exact computing as the core and promote Instruction sets for the same. However, in many applications especially for IoT edge devices, for simple operation, this consumes a lot of silicon area and energy. This results in increased costs for the unnecessary functions/accessories which will not be used in the lifetime of IoT devices. So to solve this problem we are exploring the design of Application Specific Instruction Set Processor (ASIP) for approximate computing applications. Many emerging applications do not require an exact answer but rather acceptable ones. Similarly, the utilization of a complete instruction set for the IoT edge devices is also a matter of concern especially when battery life is restricted and most of the instructions of ARM and similar processors are never used in the lifetime of the IoT cycle. Approximate computing deliberately introduces significant “acceptable errors” into the computing process and promises significant energy efficiency gains, [2]. The approximate computing along with ASIP can offer a potential solution to the problem of silicon area, energy, and increased cost of hardware. Synopsys Processor Designer (PD), Cadence Tensilica is a tool-based solution for the design and implementation of ASIPs. The Language for Instruction Set Architecture (LISA), a processor design platform (LPDP) based on machine descriptions in the LISA language provides one common environment for design phases present in ASIP, [1], [3].

General-purpose processorsdonot perform all the operations related to mathematics such as exponential, sine/cosine, etc. To use this function we have to include a math header file in the program. Sometimes there is no need to work faster in simple operations but the processor works faster which is not good. Approximate computing can be done at different levels such as approximate software, approximate architecture, and approximate circuit, [2]. We can improve energy efficiency, and speed, decrease power dissipation, and silicon area using

approximate computing, and result in a decreased cost of hardware.

This paper is organized as follows. Section 2 focuses on the ASIP approach and its advantage over other processor design methodologies. Section 3 discusses the type of methods available for approximate computing and gives an idea about a fractal method for approximate computing. Section 4 provides information about xtensa architecture and TIE languages in brief. Section 5 provides information on the project methodology flow for design instructions. Results are calculated on a software basis for the proposed method. Lastly, the paper is concluded and future scope is discussed.

# 2 Application-Specific Instruction Set Processor

The term application specific is not only associated with application software but is also applicable to the functionof the processor with a different perspective such as system-specific context, a specific function with a unique design objective, [1], [4], [5]. It is important to meet design constraints related to ASIP through hardware implementation and design point of view. Also, it is observed that the instruction format for ASIP is slightly different from the traditional instruction which contains mnemonic and operands that use register/memory to store or load the data into it, [1]. ASIP interfaces are used for outer communication whereas inner communication between functional units is defined by instruction operand. Currently, ASIP design consists of four phases Architecture Exploration, Architecture Implementation, Software Application Design, and System Integration, [6], [7].The basic information about these four phases is given as follows:

## 2.1 Architecture Exploration

In this phase, the application is implemented on a processor with a repetitive process to obtain a perfect fit between architecture and application, [1], [6]. In addition to this, hotspot plays an important role in securing great performance improvement in the application. Hotspots are software functions that consume the majority of processor cycles in the application. It is important to detect hotspots present in the application if any and should be minimized. Nowadays, various tools are available to identify hotspots in an application with a profile option.

## 2.2 Architecture Implementation

The user-defined processor must be synthesized using Hardware Description Languages (HDL) and this can be done using different HDLs like VHDL and Verilog. The synthesized codes are used to implement on PLDs like Field Programmable Gate Array (FPGA), Programmable Array Logic (PAL), and Programmable Logic Array (PLA), [7], [8].

## 2.3 Software Application Design

Systematic application design for specific purposes is a challenge to software designers as they require the collection of various software development tools for standard production [6]. But still, a requirement of software designers and hardware designers play a different role in software development tools.

## 2.4 System Integration and Verification

Usually, cycle-accurate processor instruction sets are released for verify the development activities of a particular processor. TI's Code Composer Studio promotes this extensively. Also, the BDTI 2000 benchmark helps in the evaluation of the processor family for its performance. Unlike this, the Tensilica Xtensa processor modules are not working in simulation mode when we use them in Xtensa Xplorer software.

Due to all above mention features, ASIP is preferred over General Purpose Processors (GPP) and Application Specific Integrated Circuit (ASIC) because of some issues. Table 1 gives a comparison on a different processor with respect to the various parameters.

Table 1. Comparison between different SoC, [1]

| Specification | GPP | ASIP | ASIC |
|---|---|---|---|
| **Performance** | Low | High | Very High |
| **Power** | Large | Medium | Small |
| **Software Design** | Large | None | Large |
| **Hardware Design** | Small | Very Large | Large |
| **Flexibility** | Excellent | Good | Poor |
| **Cost** | Mainly on Software | Volume-Sensitive | Large |
| **Reuse** | Excellent | Good | Poor |
| **Market** | Very Large | Small | Large |

Referring to Table 1, ASIP is the solution to the tradeoff between performance and flexibility associated with GPP and ASIC respectively. Due to this reason, we move towards ASIP as compared to GPP and ASIC.

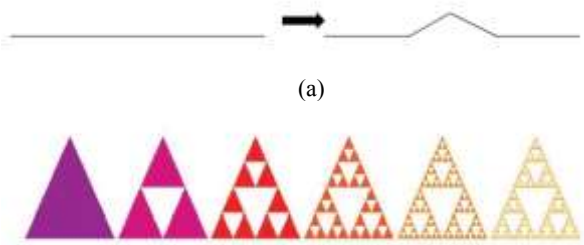
# 3 Approximate Computing

Approximate computing plays an important role in smart cities wherein we think about low power

consumption in hardware, especially through an IoT perspective. There is lot of work done by Purdue University [2]. The main focus of this paper is on a new method of approximate computing using the fractal method.

### 3.1 Fractals

Fractal is a complex and never-ending pattern that repeats itself on a different scale and follows self-similarity property, [9], [10]. They are made by repeating a simple process. Some physical systems are complex, repetitive, and productive and are represented by fractals. Generally, fractals does not stick to regular mathematical dimensions such as 1-D, 2-D, 3-D, etc. so it is obvious that fractals are distinguished as a real dimension e.g. fractal curve and fractal surfaces, [9]. The fractal curve has a dimension between 1-D and 2-D whereas the fractal surface lies between 2-D and 3-D, [9]. Fractals can be observed in nature, algebra, geometry, circuits, and city. Fractal uses replacement rule for the development of irregular shape. For example, a straight line with dimensions1-D can be divided into three equal parts and the middle part is replaced by the shape as shown in Figure 1(a) and it has the same length as an original middle portion and it goes on.



(a)

(b)

Fig. 1: Replacement Rule (a) Fractal in a straight line, [6] and (b) Fractal in a triangle, [9]

As shown in Figure 1(b), the replacement rule can be applied to a 2-D object like a triangle, star, etc. There are different types of fractals like Mandelbrot Set fractal, Julia fractal, etc. This paper mainly focuses on Mandelbrot Set fractals.

### 3.2 Mandelbrot Set Fractal

The Mandelbrot Set is a function that uses a complex number such as $z = x + iy$ in the fractal. The absolute value of a complex number is given and denoted by $\sqrt{x^2 + y^2}$ and $|z|$ respectively, [10].

*Definition 1:* The Mandelbrot set is complex quadratic iterative equation and assemblage of the value of c in the complex plane for which succession of the quadratic plot remains bounded, [10].

$$Z_{n+1} = Z_n^2 + C \quad (1)$$

In equation (1) above, C is any value (complex/real/integer) lies in a complex plane $z_n$ and $z_{n+1}$ is the iterative value in the complex plane for a finite number of points (n).

*Definition 2:* A point is said to be in Mandelbrot Set if it is satisfies, [9]:

$$|Z_{n+1}| < 2; for\ all\ n > 0 \quad (2)$$

In another word, if the absolute value of $z_{n+1}$ is more than 2 then Mandelbrot sequence will go to infinity. Refer to (1), suppose initially $c = 1$ and $z_0 = 0$, then function $z_{n+1}$ will have value 0,1,2,5,26,…..and it goes on infinity and referring to (2), we can figure out that selected point does not belong to Mandelbrot Set whereas if $c = -1$ and $z_0 = 0$.Function $z_{n+1}$ contains value 0,-1,0,-1…goes to infinity. By observing function $z_{n+1}$ it is found that second point belongs to the Mandelbrot set fractal. Equation (2) gives us an idea about the convergence of the Mandelbrot set in complex plane Z.

## 4 XTENSA Architecture And TIE Language

Xtensa processor is based on simple RISC architecture with a single core, [11], [12], [13], [14]. It has a separate instruction bus and data bus which allows load/fetch of data concurrently and consists of 5 or 7-stage pipeline structure which is configurable. It uses a 24-bit instruction set for xtensa core and instructions are defined using TIE language. According to the requirement, it supports optional 16-bit "density" instructions which can be used frequently in an assembly program. Pipeline structure of xtensa consists of different stages like Instruction Fetch (I), Register Read (R), Execute (E), Memory Access (M), and Write Back (W).

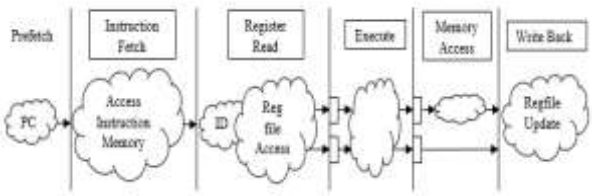



Fig. 2: 5-stage abstracted pipeline view for instructions

Figure 2 shows a simple and basic representation of the 5-stage pipeline for xtensa core. The

combinational logic in the core is represented by clouds. Generally, every instruction spends a minimum of one clock cycle in each stage and it may vary depending on other situations like interrupt and exceptions which occur during program execution. The working of each stage present in Figure 2 is given below:

***Prefetch:*** It is used to send the address to instruction memory.

***Instruction Fetch:*** The main purpose of this stage is to fetch an instruction from memory & align the instruction.

***Register Read:*** This stage is used to decode the instruction and read the value from the register file during the execution of instructions.

***Execute:*** It is the most important stage where all computation and address calculations for different types of instructions are performed.

***Memory Access:*** This stage loads or reads data from memory or cache and loads it in ALU result.

***Write Back:*** This is the 5th stage of the pipeline and writes computed results obtained from the executing stage via the memory access stage to the corresponding register file or memory.

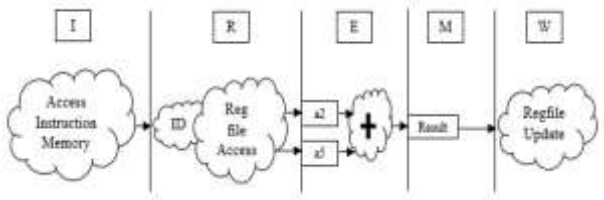



Fig. 3: 5-stage pipeline representation of ADD instruction

Figure 3 represents a 5-stage pipeline of ADD instruction. In the execute stage, the addition of a2 and a5 takes place and then it is stored in memory and finally, the result gets stored in a4, which is part of the register file.

As xtensa supports 7- a stage pipeline, so the remaining two stages for xtensa core are H stage used for instruction fetch along with the I stage and L stage extended with M stage to access local memory between instruction and data memory. According to application requirements, user-defined instruction can use the xtensa platform as it provides various constraints present in TIE language like operation, semantic, scheduler, functions, register file, proto, ctype declaration etc. These constraints can be used to add instructions for various DSP processor and specific applications. We have used TIE language due to its immense features which are available to end users and through which instruction sets can be created. The instructions requiredto support Mandelbrot Set Fractals are divided into different part parts like arithmetic instructions, data transfer instructions, etc.

Table 2. Proposed Instruction Set for Mandelbrot Set fractal

| Instruction Type | Instructions | Data Length |
|---|---|---|
| **Arithmetic** | CADD, CSUB | 32-bit |
| | CMUL, CSQU | |
| **Data Transfer** | LDA16, STA16 | 16-bit |
| | LDA32, STA32, MOV | 32-bit |
| | LDA64, STA64, MOVE | 64-bit |

Table 2 shows the classification of the instruction set. The instruction syntax in assembly language is shown below:

ADD ar, as, at AR[r] ← AR[s] + AR[t]

Where ADD is mnemonic with source and target are two operands and denoted by *as* and *at* respectively, whereas *ar* is the destination operand. AR is a predefined register which is used in the application.

| | | | | |
|---|---|---|---|---|
| 1000 0000 | R | S | T | 0000 |

Fig. 4: Bit representation for Opcode and operand in Assembly Language

Figure 4 shows the bit representation format of ADD instruction for opcode and operand part present in assembly language. The lower 4 bits are reserved for the register address. With 4 bits, 16 locations can be accessed. The high order 8 bits represent the opcode for that instruction.

## 4.1 Program Code

The actual definition for Mandelbrot Set user-defined instruction in assembly and C/C++ language is given below:

**(A) Assembly Language Representation of Design TIE Instruction:**

```
CADD c4,    c3,    c1;    //c4 = c3 + c1
```

**(B) C/C++ Function Prototype for Design TIE Instruction:**

```
// C or C++ header file declaration
#include<xtensa/tie/add.h>
int main ()
{
        ....
        ....
        CR32 X, Y, Z;
```

```
        ....
        Z = CADD (X, Y);
        ....
        Return 0;
}
```

**Code1:** Code snippet for Instruction calling definition of Mandelbrot set in assembly and C/C++ language

From code-1, it is observed that instruction has a specific direction for input and output argument. Due to this, the instruction calling process will be different in C/C++ language and mathematical representation in the assembly language will be changed.

# 5 Project Methodology

There are various steps that are followed to obtain the best possible optimization in hardware through a software application in the cadence Tensilica tool. Figure 5 gives a simple and basic idea about the steps present in the instruction design process. The first step is to create an xtensa C/C++ project followed by C/C++ application code and selection of language depending on our expertise. The next step is to look for an active set like a project which we have created, the configuration on which we will write source code, and finally the debug configuration option which is used to debug code on the FPGA platform. Once successfully built and run, themost important part is benchmark perspective which is nothing but profiling of source code. It gives information about hotspots present in the code. After the analysis performed in the profile section, we will use predefined TIE instruction with new code and compare its result with the source code. If it is observed that both results are the same then instructions are designed using TIE language to reduce hot spots in design and instructions are verified in C/C++ source code with some modification in the source code. Again repeat the same process up to the profile configuration. Now compare the profile section with and without TIE for optimization purposes.

Once the optimized source code is finalized then the next part is to generate synthesizable RTL code for the application-specific processor and the last step is to implement it on FPGA hardware using vivadotoolchain provided by Xilinx where we can check for the area, power, and operating frequency as it gives us exact information about newly design processor. The results obtained in the TIE report are approximate and hence to get an accurate result this hardware implementation is a necessary step.

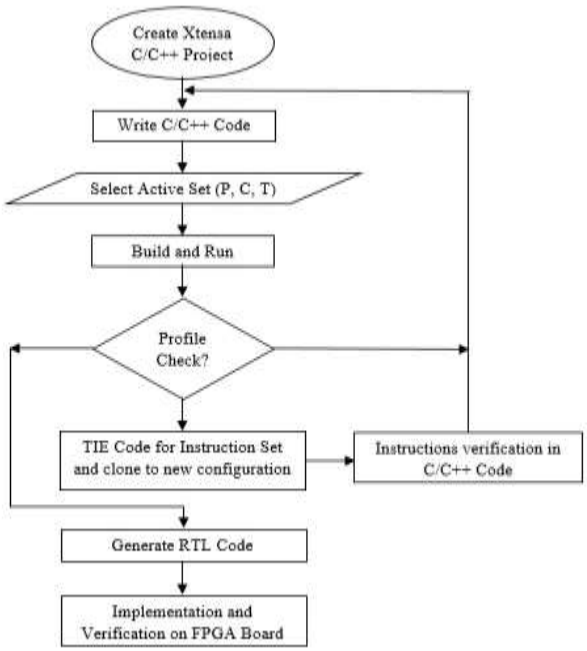



Fig. 5: Proposed Flowchart of adding Instructions for Processor Design using TIE Language

## 5.1 Design Instruction

This section gives information about designed instruction for the Mandelbrot set fractal along with its prototype. There are two types of instruction such as - arithmetic and data transfer instruction. Table 2 shows the proposed instructions set for Mandelbrot Fractal. The **"operation"** TIE construct is a primary requirement for the design of any instruction. TIE has another construct to design instruction like "iclass". Also, we require a register file to store the value of the input argument, and can be done using the **"regfile"** TIE construct. Along with this, we can use immediate data in the instruction whenever we require instant data for application and it is just like the 8085 microprocessor or 8051 microcontroller with different instruction widths.

# 6 Experimental Results

The instruction to support the Mandelbrot Set is synthesized using ASIP and approximate computing. The different parameters such as performance, area, the frequency are analyzed on the Tensilica EDA tool. The exact value of the parameter is available after synthesis.

Figure 6 shows the instruction cycles comparison graph for with and without TIE for user-defined instructions**.** So by using TIE language, cycles are

reduced. Along with this, by using different TIE construct, the hardware area is reduced. TIE construct like semantics is used to design single data paths for different instructions.

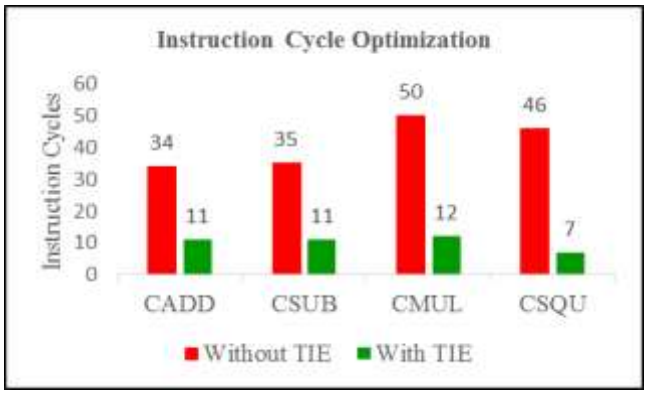



Fig. 6: Instruction cycle optimization for Mandelbrot Set

Table 3. Profile information for Mandelbrot set instruction

| Parameters | Without TIE | With TIE |
|---|---|---|
| **Total Number of Cycles** | 104946 | 74554 |
| **Total Instructions** | 68198 | 47409 |
| **Application Size (bytes)** | 1012 | 566 |

Table 3 gives information about the architecture performance of the xtensa processor. A superior architecture will result in better performance. We don't explore other processor configurations which is available in the xtensa tool. The period of a selected processor configuration for a single clock cycle is 1.26 ns. Similarly, the same architecture but on different fabrication technologies like 45nm, 90nm, and 22nm are not explored in this paper. We specifically focus on TIE.

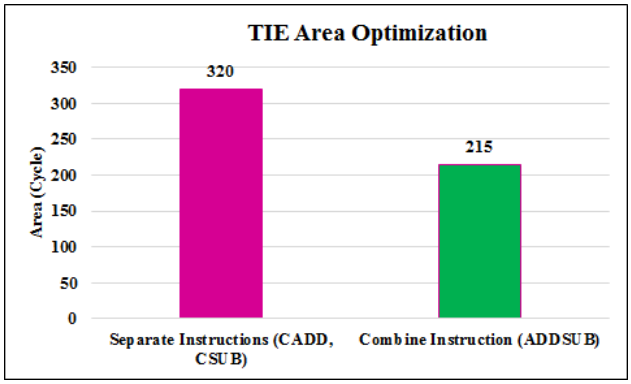



Fig. 7: TIE Area optimization for Mandelbrot Set

Figure 7 shows the comparison between combined and separate instruction which is designed for the Mandelbrot set fractal. From Figure 7, it is observed that the area for designed instruction is reduced.

Table 4. TIE Area Overview

| Parameters | Area(Gate) | Percentage (%) |
|---|---|---|
| **TIE Instruction** | 8217 | 31.85 |
| **Register File** | 11826 | 45.84 |
| **Functions** | 1380 | 5.34 |
| **Decoder, Mux, etc.** | 3195 | 12.38 |

Table 4 provides an idea about the area in terms of gate count related to TIE instruction. Different architectures such as CISC, RISC, and NISC are implemented on FPGA and compared with TIE. NISC is used to convert C program applications to hardware language with no instruction, fixed design architecture, and no flexibility to explore design space whereas in TIE it allows us to explore design space for the instruction set. Hence though Table 4 gives gate count for instructions, register files, functions, decode, and multiplexer it is not treatedsimilarly to FPGA gate count. The TIE report is useful in designing iterative-based optimization i.e. changing the register file number, and register word length. For the same processor configuration, one configuration design used an $N_1$ register number for each of the $W_1$ word lengths out of these designs compared with another configuration with $N_2$ as the register number and $W_2$ as word length of each register where $N_2 < N_1$ but $W_2 > W_1$. This report also helps in overcoming design constraints like register files, instruction sets, and clock speed. The experimental result with a change in clock speed is not the focus of this paper.

```
Slot 0: CADD(8965_4523,3578_7412)=12543_11935
Slot 0: CSUB(8965_4523,3578_7412)=5387_-2889
Slot 0: CMUL(8965_4523,3578_7412)=-1447706_82631874
Slot 0: CSQU(8965_4523)=59913696_81097390
```

Fig. 8: Simulation result for design TIE instruction

Figure 8 shows the instructions that have been simulated for the Mandelbrot set fractal and these simulation results are in agreement with manual calculation. The Analysis of power constraint using approximate computing is not calculated.

## 7 Conclusions

This paper explains two emerging areas such as ASIP and approximate computing in the semiconductor field. ASIP is used to overcome the problem of GPP and ASIC. It is observed that the overall performance of processor architecture is improved by 40.76% concerning TIE. The improvement observed in the number of instruction cycles ranges from 20.9% - 55.7% for the designed instruction. TIE construct like semantics is used to design single data paths for different instructions. Using semantic construct, for combined addition and

subtraction instruction, the area is reduced by 1.48 times compared to separate addition and subtraction instruction. Moreover, this process of synthesis of Application-specific instruction set processor is faster.

*Acknowledgement:*
Authors would like to thank Cadence Design System, Pune for their immense inputs and timely suggestions which helped us in timely completion of our proposed work. We also acknowledge the guidance of late Prof. A B Patki for his continuous guidance and inspiration in successful completion of this work.

*References:*

[1] Nohl, Achim et al. "Application specific processor design: Architectures, design methods and tools." *2010 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)* (2010), San Jose, CA, pp. 349-352.

[2] Q. Xu, T. Mytkowicz and N. S. Kim, "Approximate Computing: A Survey," in *IEEE Design & Test*, vol. 33, no. 1, pp. 8-22, Feb. 2016, doi: 10.1109/MDAT.2015.2505723.

[3] P. V.Bhandarkar, S.S.Limaye, "Modeling with LISA- A Power Full Machine Description Language", *IJCSET*, Vol-3, Issue 1, 2013, pp. 1-7.

[4] Laura Micconi, Jan Madsen, Paul Pop, "System-level synthesis of multi-ASIP platforms using an uncertainty model", *Integration, the VLSI Journal*, Volume 51, 2015, pp. 118-138, ISSN: 0167-9260.

[5] S. Ramanathan, V. Visvanathan, S.K. Nandy, "Synthesis of ASIPs for DSP algorithms", *Integration*, Vol. 28, Issue 1, 1999, pp. 13-32, ISSN 0167-9260.

[6] M. Hohenauer, R. Leupers, "ASIP Design Methodology. In: C Compilers for ASIPs Automatic Compiler Generation with LISA", *Springer Science and Business Media, LLC*, 2010, pp.7-13.

[7] Meyer-Bäse, U., Vera, A., Rao, S., Lenk, K., and Pattichis, M., "FPGA wavelet processor design using language for instruction-set architectures (LISA)", in Independent Component Analyses, Wavelets, Unsupervised Nano-Biomimetic Sensors, and Neural Networks, 2007, vol. 6576: doi: 10.1117/12.719020.

[8] J. Stepaniuk, M. Kopczynski, T. Grzes, "The First Step Toward Processor for Rough Set Methods", *Fundamenta Informaticae*, vol. 127, Issue 1-4, Jan. 2013, pp. 429-443.

[9] D. Ashlock, "Evolutionary Exploration of the Mandelbrot Set," *2006 IEEE International Conference on Evolutionary Computation*, Vancouver, BC, Canada, 2006, pp. 2079-2086, doi: 10.1109/CEC.2006.1688563

[10] L. Lazareck, G. Verch, J. F. Peters, "Fractal in Circuits", *Canadian Conference on Electrical and Computer Engineering*, Canada, 2001, pp. 589-594.

[11] Cadence Design Systems, Inc. (2019). Tensilica Vision DSP Family [White Paper]. Cadence Design Systems, Inc.

[12] Cadence Design Systems, Inc. (2014). Xtensa Processor Developer's Toolkit [White Paper]. Cadence Design Systems, Inc., [Online]. https://ip.cadence.com/uploads/102/HWdev-pdf (Accessed Date: June 17, 2024).

[13] Cadence Design Systems, Inc. (2014). Tensilica Software Development Toolkit (SDK) [White Paper]. Cadence Design Systems, Inc, [Online]. https://ip.cadence.com/uploads/103/SWdev-pdf (Accessed Date: June 17, 2024).

[14] Ninad Patil, Vanita Agarwal, "Performance Simulation of a Traffic Sign Recognition based Neural Network on Cadence's Tensilica Vision P6 DSP using Xtensa Xplorer IDE", WSEAS Transactions On Computer Research, Vol. 10, 2022, pp. 35-42, https://doi.org/10.37394/232018.2022.10.5.

**Contribution of Individual Authors to the Creation of a Scientific Article (Ghostwriting Policy)**

- Akshay Deole carried out the simulation work.
- Vanita Agarwal and Vaishali Ingale were responsible for ideating, formulating, guiding, organizing and execution of Research carried.

**Sources of Funding for Research Presented in a Scientific Article or Scientific Article Itself**
No funding was received for conducting this study.

**Conflict of Interest**
The authors have no conflicts of interest to declare.