Rev. Roum. Sci. Techn.– Électrotechn. et Énerg.
Vol. **70**, *4*, pp. 519–524, Bucarest, 2025

REAL-TIME VEDIC MATHEMATICS BASED MEMORYLESS ARITHMETIC CIRCUITS VERIFICATION TECHNIQUE

SINGARAVELU DEVI POONGUZHALI[1], THAMMAMPATTI NATARAJAN PRABAKAR[2],
BALASUBRAMANIAN LAKSHMI[1], SUNDARAM RAMKUMAR[3]

Keywords: Testing; Formal verification; Vedic mathematics; Arithmetic circuits; Test vectors; Very large scale integration (VLSI).

This paper proposes a new method for verifying arithmetic circuit operations based on the Vedic mathematics Sutra (formulae) "Gunita Samuccaya". According to this sutra, our proposed method verifies arithmetic operations, *e.g.*, c = a + b, by checking whether the sum of 'a' and 'b' digits equals the sum of digits of 'c' for correct computation. In contrast to built-in self-test (BIST) schemes, our approach is simpler, eliminating traditional test pattern generators and output analyzers while achieving 100% fault coverage for simple arithmetic operations. Our system, designed in Verilog hardware description language (HDL), is real-time, memoryless, and scalable. This proposed testing method revolutionizes arithmetic circuit verification, guaranteeing the integrity of intricate digital systems where mathematical precision is vital.

1. INTRODUCTION

The importance of arithmetic unit verification has increased significantly due to the widespread use of arithmetic modules in demanding applications like multimedia, signal processing, and encryption within embedded systems. Although certain Electronic Design Automation (EDA) vendors offer tools to ensure arithmetic components are "correct by construction," validating non-standard, bit-optimized embedded arithmetic circuits remains a complex task. To tackle this challenge, a range of verification methods and tools, including formal and simulation methodologies, are utilized. As very large scale integration (VLSI) design complexity increases, field programmable gate arrays (FPGAs) are being used more often to quickly prototype control units and arithmetic circuits. FPGA implementation has the advantage of fast deployment, especially when a significant amount of design time is taken up by debugging logic designs. Despite this, layout principles have a significant impact on circuit performance, even with minor changes to the circuit structure [19]. In [1], a debugging technique for arithmetic circuits that focuses on specific circuit adjustments to accelerate redesign efforts is proposed. Debugging time is reduced by examining the circuit, identifying flaws, and replacing problematic components with suitable circuit parts. The need to start design verification and debugging early in the design process is highlighted by the increasing size and complexity of digital systems. A key hurdle in assessing such complex systems involves managing mathematical data paths and their elements, like multipliers and dividers. While many hardware verification tools rely on bit-level strategies such as satisfiability or binary decision diagrams solvers, these techniques encounter challenges in addressing scalability issues when handling intricate arithmetic circuits [2-3]. In the early twentieth century, Swami Bharati Krishna Tirtha advocated and popularized Vedic Mathematics, based on India's oldest scriptures, the Vedas. This ancient Indian mathematical system includes various techniques and principles to efficiently solve complex mathematical problems. By using Vedic mathematics in this work, we propose a verification method for the basic arithmetic circuits. In contemporary times, Vedic mathematics has undergone a revival in several technological fields, including computer science, data analytics, cryptography, and artificial intelligence. A significant application of Vedic mathematics lies in algorithm optimization and complexity analysis [21].

The techniques of Vedic mathematics can aid in the design of efficient algorithms for solving intricate computational problems [4]. Vedic mathematics, a traditional Indian system of mathematics, contains 16 sutras that facilitate quick problem-solving in most areas of mathematics in contemporary computing environments. One obvious thing is that the suggested square utilizes the Ekadhikena Purvena sutra, meaning "one more than the former." Classically used to square decimal numbers terminating in 5, we extend and modify this sutra for effective squaring of binary numbers, demonstrating its flexibility and possibilities for creative use in digital computation. The Squarer proposed illustrates large benefits, with nearly 50% area savings and a 50% delay reduction, performing better than the duplex squarer in the 32-bit configurations [22]. In this paper, we propose that Vedic mathematics supports the testing of arithmetic circuits by providing efficient methods that aid in verifying their accuracy and functionality, thereby contributing to the overall reliability of arithmetic operations within digital systems.

Our proposed testing method using Vedic mathematics involves providing immediate inputs to the circuit under test and verifying its output simultaneously, rather than storing the reference signatures in a large memory space for comparing them to find faults.

1.1 CONVENTIONAL BIST ARCHITECTURE

The built-in self-test is a conventional technique, as shown in Fig. 1, that allows an integrated Circuit to achieve self-testing, and it includes three main components: output Analyzer (OA), test pattern generator (TPG), and circuit under test (CUT). The TPG uses a linear feedback shift register (LFSR) to generate the required input vectors for the CUT during testing. The OA consists of a memory unit and a comparator. It stores the standard output values (known as golden or reference signatures) from the CUT in memory.

[1] Electronic and Communication Engineering, SASTRA Deemed University, Thanjavur, Tamil Nadu, India.
[2] Faculty of Electronic and Communication Engineering, SASTRA Deemed University, Thanjavur, Tamil Nadu, India.
[3] Department of Electronic and Communication Engineering, Sri Eshwar College of Engineering, Coimbatore, Tamil Nadu, India.
E-mails: dpfi0423011686@sastra.ac.in, prabakar@ece.sastra.edu, lakshmi@cse.sastra.edu, ramkumar.s@sece.ac.in

DOI: 10.59277/RRST-EE.2025.70.4.15

The comparator then checks the output produced by the CUT against the standard output values stored in memory. If the circuit is functioning correctly, the outputs will match. If there are any defects in the circuit, the actual outputs will differ from the expected ones.

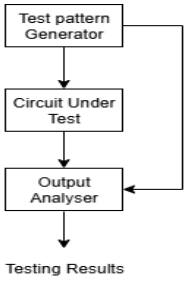

Fig. 1 - Conventional BIST Architecture.

1.2. PROPOSED VEDIC TESTING ARCHITECTURE

The proposed testing strategy (Fig.2.) takes the circuit's inputs and outputs and uses them in the Vedic testing module. This module is based on the Vedic principle of Gunita Samuccayah. This approach aims to improve the accuracy and reliability of our testing process and differs from the traditional BIST method because it checks the circuit under test using immediate inputs and their corresponding outputs, without the need to store reference or golden signatures in a large memory.

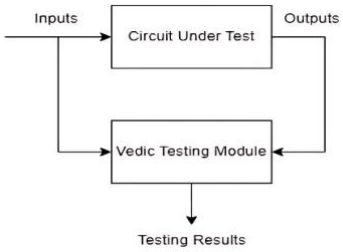

Fig. 2 – Basic block diagram of proposed testing method.

The proposed testing methodology is illustrated in Fig. 3. In this methodology, the inputs provided to and the outputs from the circuit under test are sufficient to evaluate its correctness of operation when it is further processed in the Vedic testing module.

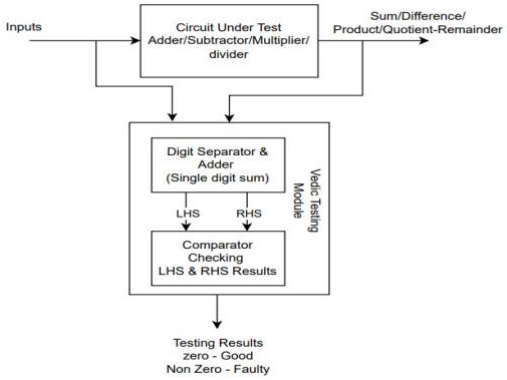

Fig. 3 - Proposed Vedic testing architecture.

We initially tested the Vedic testing module with a set of input values for all arithmetic operations to ensure it was working correctly. Following that, we tested it with different input sizes for addition, multiplication, subtraction, and division operations. The details of the testing process are illustrated in Section 3.

We initially tested the Vedic testing module with a set of input values for all arithmetic operations to ensure it was working correctly. After that, we tested it with different input sizes for addition, multiplication, subtraction, and division. The details of the testing process are illustrated in Section 3.

2. RELATED WORK

Arithmetic circuits are fundamental to digital computing systems, making their accuracy and reliability crucial for optimal performance. There are various techniques available for testing these circuits, each with its own advantages and applications. Verifying arithmetic circuits, especially multipliers, remains a significant challenge despite advancements. The primary approach is to represent the circuit using computer algebra as a set of pseudo-Boolean polynomials. For converting the circuit to a specific mathematical problem, it must be checked whether the circuit polynomials imply the given word-level specification. Solving this problem is essential to confirm that the circuit functions as intended. [5]. Recent techniques for multiplier circuit verification use computer algebra and SAT solving to model the circuit as polynomials and generate a Gröbner basis for verification. Complex final-stage adders are addressed using satisfiability (SAT) solvers, while adder substitution simplifies the verification process [6].

The verification method for arithmetic circuits utilizing output signature (OS) and input signature (IS) polynomials based on symbolic computer algebra (SCA) was presented in [7]. To verify the end polynomial with respect to the IS, the method analyzes the backward step-by-step substitution of the gate polynomials within output polynomials according to the topologically reversed order of the circuit. Further, it discusses the design of arithmetic blocks, such as the Partial product generator (PPG), final sum adder (FSA), and partial product accumulator (PPA), as well as the conversion of binary moment diagrams (BMDs) from binary decision diagrams (BDDs).

In [8], a methodology for verifying arithmetic circuits using Taylor expansion diagram (TED) data structures and a technique that merges an inverter graph (AIG) and an adder tree to verify the correctness of arithmetic functions is proposed. It aims to achieve fewer phases and a reduced area compared to existing methods, using a backward rewriting technique for function extraction. Verifying gate-level divider circuits is challenging due to the need for extensive gate-level verification and the limitations of traditional Boolean approaches. To address this, an extended algebraic model is utilized to prove the accuracy of the division algorithm without requiring a reference design [9]. This technique demonstrates the functionality of gate-level divider circuits, building on successful methods used for other arithmetic circuits. As the complexity of VLSI design increases, FPGAs are increasingly used to quickly prototype control units and arithmetic circuits. The advantage of FPGA implementation is its fast deployment, particularly for arithmetic circuits in embedded systems, which can pose challenges for debugging due to their complexity and non-standard implementations. This approach automates the generation of directed tests by assigning input variables in a particular way, ensuring the remainder is non-zero. Bug localization and correction are facilitated by analyzing remainder patterns and test activations

[10]. In their paper [11], the authors have provided a thorough formalization of polynomial reasoning and introduced a new column-wise verification technique for validating gate-level multipliers without reducing a full word-level specification. This approach demonstrates the precision and comprehensiveness of using exact formalization. The experiments demonstrate that simple multipliers can be easily analyzed using standard computer algebra methods, but more complex and optimized multipliers require more advanced procedures. The procedure presented in [12] extends to mere verification; it can reveal the exact mathematical function performed by the circuit by examining its outputs (*i.e.*, it extracts the arithmetic function).

The research [13] delves into the validation of arithmetic operations performed by a circuit by discovering a distinct mathematical formula, known as a bit-level polynomial function, that is embedded in the circuit's gates. This method identifies the test circuit's main arithmetic function by extrapolating the input signature from the output signature. On the other hand, Function Identification uses the extracted input signature to reveal the circuit's arithmetic operation when its mathematical function is not initially known. The goal of the study [14] was to identify logic errors in a synthesized circuit caused by the incorrect gate (also known as a "gate replacement" error). Initial findings support the effectiveness of the proposed approach in addressing practical issues, with a future focus on optimizing cut generation for improved efficiency. One possible strategy is to sample the circuit with specific cuts and confirm their signatures using a "binary search" technique [20].

To validate carry signals, this method [15] requires a detailed analysis of different Exclusive-Or (XOR) tree topologies, resulting in an exponential runtime cost. Conversely, another verification technique outlined in another reference uses a reverse-engineering approach to quickly synthesize a network of HAs (half-adders) from a gate-level description. The extraction process uses a BLA (Bit-level adder) representation, which is well known for its reliability across different arithmetic circuit topologies. [16]. Performing transformations from primary outputs to primary inputs in reverse topological order is the method suggested in this paper [17]. The method checks for equivalence using canonical data structures such as TED or BMD. In the event of mismatches, SAT/ satisfiability modulo theories (SMT) problems can be solved to identify bugs. The algorithm works on basic Boolean gates but can handle complex gates by writing equations for each internal signal. Once the input signature is computed, it is compared against the expected specification to assess correctness. The method for verifying large arithmetic circuits efficiently is proposed in [18]. This method appears to involve extracting Boolean polynomials from the gate-level implementation, computing a Groebner basis, and reducing the polynomials for verification.

3. PROPOSED METHOD

The method we used for testing arithmetic circuits was inspired by the Vedic sutra Gunita Samuccaya, originally used to validate polynomial factorization. We applied this verification approach to basic arithmetic operations and found that it was effective.

3.1 GUNITA SAMUCCAYAH FOR VERIFYING FACTORIZATION RESULTS

Gunita Samuccayah-Samuccaya Gunitah' is a sub-sutra in Vedic mathematics that is intended to verify the correctness of obtained answers in factorization. It says that: *"The POS (Product of Sum) of the coefficients in the factors is equal to the sum of the coefficients in the polynomial equation".*

Equation (1) represents a third-order polynomial equation and its factorization. Let us verify the factorization result using the principle of Gunita Samuccayah-Samuccaya Gunitah.

$$y^3 + 10\,y^2 + 11\,y - 70 = (y + 5)\,(y + 7)\,(y - 2). \quad (1)$$

By finding the sum of the coefficients of the polynomial equation at the left-hand side (LHS) & POS of the coefficients in the factors at the right-hand side (RHS), we get,

$$1 + 10 + 11 - 70 = (1 + 5)\,(1 + 7)(1 - 2).$$

By simplifying,

$$22 - 70 = 6 \times 8 \times -1,$$
$$-48 = -48 \text{ verified.}$$
$$\text{LHS} = \text{RHS}$$

Therefore, the given factorization is a valid representation of the polynomial equation.

For further understanding, one more example is given below:

$$y^2 + 5\,y + 6 = (y + 3)(y + 2)$$
$$\text{Now, } 1 + 5 + 6 = (1 + 3)(1 + 2)$$
$$12 = 4 \times 3$$
$$12 = 12$$
$$\text{LHS} = \text{RHS}$$

Thus, the given factors are valid, where y is the polynomial variable

We discovered that the Gunita Samuccaya-Samuccaya Gunitah Vedic sutra can also verify the correctness of other mathematical operations, such as addition, division, squaring, cubing, and more. The numerical verification is provided below by summing the digits on both sides.

3.2 GUNITA SAMUCCAYAH FOR VERIFYING OTHER ARITHMETIC OPERATIONS

i. Addition

Let us consider this addition example,

$$98473 + 54672 = 153145$$

By finding the sum of the digits on both sides,

$$(9+8+4+7+3) + (5+4+6+7+2) = 1+5+3+1+4+5$$
$$(3+1) + (2+4) = 1+9$$
$$4 + 6 = 10$$
$$10 = 10$$
$$1+0 = 1+0$$
$$1 = 1$$
$$\text{LHS} = \text{RHS}$$

Thus, the addition result is verified to be correct.

If the single-digit sum of the LHS and RHS is the same, then we can conclude that the adder circuit is working correctly. A similar verification procedure is followed for other arithmetic operations, as in ii, iii, and iv.

ii. Subtraction

Let us consider this subtraction example,

$$98473 - 54672$$

Instead of direct subtraction, 9's complement addition is used below:

98473 + 45327 * = 143800

By finding the sum of the digits on both sides,

(9+8+4+7+3) + (4+5+3+2+7) = 1+4+3+8

(3+1) + (2+1) = 1+6

4 + 3 = 7

7 = 7

LHS = RHS

Thus, the subtraction result is verified to be correct.

*The subtrahend is 54672. Its 9's complement is

99999 - 54672 = 45327.

iii. Multiplication

Let us consider this multiplication example,

98473 × 54672 = 5383715856

By finding the sum of the digits on both sides,

(9+8+4+7+3) × (5+4+6+7+2) = 5+3+8+3+7+1+5+8+5+5+6

(3+1) × (2+4) = 5+1

4 × 6 = 6

(2+4) = 6

6 = 6

LHS = RHS

Thus, the multiplication result is verified to be correct.

iv. Division

For example, the given division problem & the results are:

98473 ÷ 54672 => Quotient = 1; Remainder = 43801

The division rule is verified using the formula:

Dividend = (Divisor × Quotient) + Remainder

Let us verify it using the principle of Gunita Samuccayah-Samuccaya Gunitah.

(9+8+4+7+3) = [(5+4+6+7+2) × 1] + (4+3+8+0+1)

(3+1) = [(2+4) × 1] + (1+6)

4 = 6 + 7

4 = (1+3)

4 = 4

LHS = RHS

Thus, the division result is verified to be correct.

For all verifications, when there are no errors in the circuit or process being tested, the values on the LHS must match those on the RHS.

Table 1 shows the proposed Gunita Samuccayah verification for the different arithmetic operations.

Table 1

Proposed Gunita Samuccayah Verification

Operation	Verification Formula
Addition	(∑a(digits) + ∑b(digits)) =∑sum(digits)
Subtraction	(∑a(digits) - ∑b(digits)) =∑difference(digits)
Multiplication	(∑a(digits) * ∑b(digits)) =∑product(digits)
Division	(∑dividend(digits) = [∑divisor(digits) * ∑quotient(digits)] –∑remainder(digits)
Cubing	∑a(digits)^2 =∑result(digits)
Squaring	∑a(digits)^3 =∑result(digits)

3.3. ALGORITHM OF PROPOSED VERIFICATION METHOD

In the proposed method of testing, to test the correctness of the circuit, the inputs and the results of the circuit under test viz., adder, subtractor, multiplier, divider, squaring circuit, cubing circuit, etc., are given to the testing circuit and undergone the following processes:

Step 1: Separate the digits of the input values (LHS) given to CUT as well as the results (RHS) from them.

Step 2: On both sides, add the separated digits and calculate the sum. If the sum is a single digit, the addition should stop. If not, repeat steps 1 and 2 until the sum becomes a single digit.

Step 3: Perform the required operation (addition/multiplication) on the single-digit at the LHS as per the circuit requirements.

Step 4: Find the difference between the single-digit value of the LHS and the single-digit sum value of the RHS.

If the difference is zero, determine that the circuit under test is error-free. If the difference is non-zero, then the circuit under test is faulty. The above processing steps are shown in the flow diagram in Fig. 4 and apply to all circuits listed in Table 2, except for division. For the division process verification, we need to use the dividend, divisor, quotient, and remainder. We must use the division formula from Table 1 for verification. The process flow for division is shown in Fig. 5.

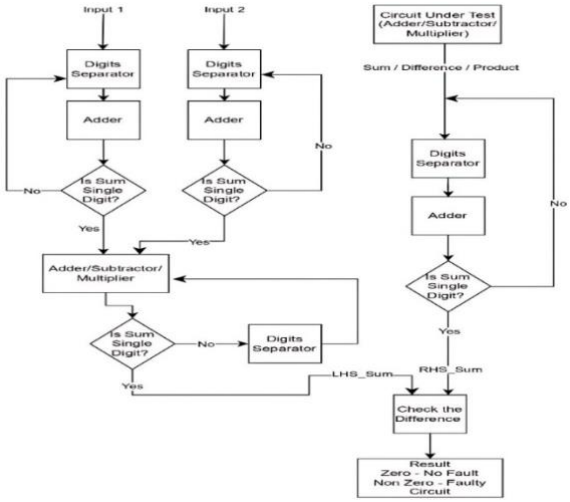

Fig. 4 – Process flow diagram of addition, subtraction, and Multiplication verification.

Table 2

Area and Power of Proposed Testing Technique for 8-bit Inputs

Sl. No.	Arithmetic Circuit	Area (LEs)	Total Power (mW)
1	Addition	124	134.48
2	Subtraction	216	151.52
3	Multiplication	746	137.78
4	Division	543	136.47

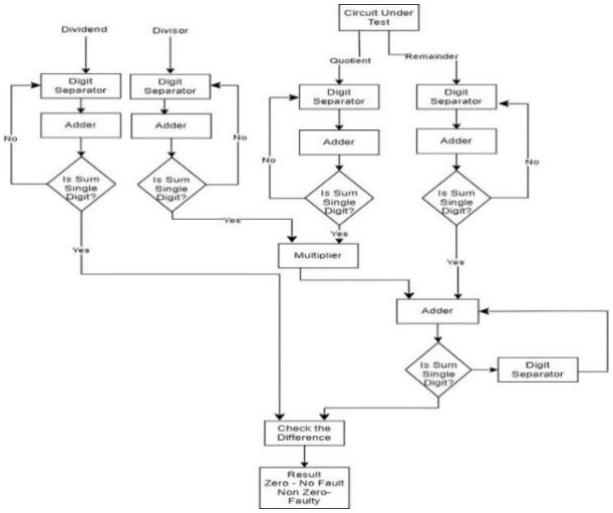

Fig. 5 – Process flow diagram of division verification.

4. RESULTS AND DISCUSSION

In this section, we confer the practical implementation and verification of the Gunita Samuccayah Vedic sutra for

verifying arithmetic circuits. This was accomplished using Verilog HDL programming, through which we determined the area and total power dissipation of the circuits.

4.1 SIMULATION RESULTS

Currently, there is a lack of real-time testing for the Gunita Samuccayah from the perspective of arithmetic circuit verification. Therefore, we present the execution results of our proposed technique below. The software implementation results for 4, 8, 16, 32, and 64-bit arithmetic operations are provided in Tables 2 to 5.

The conventional built-in self-test experiment was conducted for various types of adders, including the ripple carry adder (RCA), carry look-ahead adder (CLA), and carry select adder (CSA). Additionally, different multiplier types were examined, such as the Array multiplier, Booth multiplier, and Urdhva Tiryagbhyam (UT) sutra-based Vedic multiplier. The performance analyses were done for 4, 8, 16, and 32 bits.

Table 3

Area and Power of Proposed Testing Technique for 16-bit Inputs

Sl. No.	Arithmetic Circuit	Area (LEs)	Total Power (mW)
1	Addition	717	141.81
2	Subtraction	769	161.28
3	Multiplication	1226	143.67
4	Division	1016	143.54

Table 4

Area and Power of Proposed Testing Technique for 32-bit Inputs

Sl. No.	Arithmetic Circuit	Area (LEs)	Total power (mW)
1	Addition	2071	149.36
2	Subtraction	2165	168.25
3	Multiplication	2677	151.32
4	Division	2483	151.11

Table 5

Area and Power of Proposed Testing Technique for 64-bit Inputs

Sl. No.	Arithmetic Circuit	Area (LEs)	Total Power (mW)
1	Addition	5043	163.23
2	Subtraction	5237	183.68
3	Multiplication	5786	164.45
4	Division	5534	164.12

Tables 6 and 7 present the BIST testing results for 4-bit and 32-bit configurations. In BIST, as the number of bits increases, the dynamic memory required to store the test vectors also increases proportionally. For BIST-based adder testing, the dynamic memory requirements for storing test vectors are as follows: 160 bits for 4 bits, 4608 bits for 8 bits, 2176 KB for 16 bits, and 264 TB for 32 bits. Similarly, for multipliers, the requirements are 2408 bits for 4 bits, 512 KB for 8 bits, 68 GB for 16 bits, and 553648128 TB for 32 bits.

Table 6

BIST-based adder testing results.

No. of bits	4			32		
Adder Type	RCA	CLA	CSA	RCA	CLA	CSA
Dynamic Power(mW)	12.15	15.18	17.13	62.24	65.85	71.24
Total memory bits	160			264 GB		

However, when the proposed verification method based on Vedic Sutra 'Gunita Samuccaya' is applied to adders and multipliers of the same size and type, there is no requirement to store the test vectors separately. Consequently, the dynamic memory requirement across all cases is effectively zero, which represents the novel contribution of this paper.

Table 7

BIST based multiplier testing results

No. of bits	4			32		
Adder Type	Array	Booth	Vedic	Array	Booth	Vedic
Dynamic Power(mW)	33.12	22.98	21.12	33.12	22.98	21.12
Total memory bits	2048			553648128 TB		

5. CONCLUSION AND FUTURE WORK

The proposed novel test methodology for arithmetic circuit verification provides a revolutionary solution to real-time verification of sophisticated functional blocks. The eschewal of large memory demands, e.g., 264 GB for testing the 32-bit adder and 553648128 TB for testing the 32-bit multiplier, is a point well-taken regarding the important benefit of our method. Our novel method of testing using Vedic mathematics promises to transform computation and mathematical applications, speeding up, improving efficiency, and enhancing reliability in testing arithmetic circuits. Furthermore, this testing is an optimized approach for periodic testing, typically performed to detect and resolve the potential problems that can develop over time, e.g., performance degradation, aging effects, or manufacturing flaws. Therefore, our research makes a significant contribution to the field, opening the door to future innovation in digital system design and testing.

CREDIT AUTHORSHIP CONTRIBUTION STATEMENT

Singaravelu Devi Poonguzhali: methodology, software, writing – original draft.
Thammampatti Natarajan Prabakar: methodology, software, supervision.
Balasubramanian Lakshmi: conceptualization, investigation, visualization.
Sundaram Ramkumar: supervision, investigation, writing – review & editing.

REFERENCES

1. M. Kubo and M. Fujita, *Debug methodology for arithmetic circuits on FPGAs*, In IEEE International Conference on Field-Programmable Technology (FPT) Proceedings, Hong Kong, China, IEEE, pp. 236–242 (2002).
2. R. Kaivola, R. Ghughal, N. Narasimhan, A. Telfer, J. Whittemore, S. Pandav, A. Slobodova, C. Taylor, V. Frolov, E. Reeber, and A. Naik, *Replacing testing with formal verification in Intel® CoreTM i7 processor execution engine validation*, In A. Bouajjani and O. Maler (eds.), Computer Aided Verification, Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, vol. 5643 (2009).
3. M.D. Aagard, R.B. Jones, R. Kaivola, K.R. Kohatsu, and C.H. Seger, *Formal verification of iterative algorithms in microprocessors*, Proceedings of the 37th ACM/IEEE Design Automation Conference (DAC 2000), ACM Press, Los Angeles, pp. 201–206 (2000).
4. C.M. Kalaiselvi and R.S. Sabeenian, *Design of area-speed efficient Anurupyena Vedic multiplier for deep learning applications*, Analog Integr Circ Sig Process, **119**, pp. 521–533 (2004).
5. A. Biere, M. Kauers, and D. Ritirc, *Challenges in verifying arithmetic circuits using computer algebra*, In 19th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), Timisoara, Romania, IEEE, pp. 9–15 (2017).
6. D. Kaufmann and A. Biere, *Improving AMULET2 for verifying multiplier circuits using SAT solving and computer algebra*, Int J Softw Tools Technol Transfer, **25**, pp. 133–144 (2023).
7. M. Barhoush, A. Mahzoon, and R. Drechsler, *Polynomial word-level verification of arithmetic circuits*, 19th ACM-IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE), Beijing, China, IEEE, pp. 1–9 (2021).
8. S. Abed, M. AlMehteb, W. Mansoor, and A. Gawanmeh, *Verification*

of non-linear arithmetic circuits using functionally reduced and-inverter-graph (FRAIG), Global Congress on Electrical Engineering (GC-ElecEng), Valencia, Spain, IEEE, pp. 118–123 (2020).
9. A. Yasin, T. Su, S. Pillement, and M. Ciesielski, *Functional verification of hardware dividers using algebraic model*, IFIP/IEEE 27th International Conference on Very Large-Scale Integration (VLSI-SoC), Cuzco, Peru, IEEE, pp. 257–262 (2019).
10. F. Farahmandi and P. Mishra, *Automated Test generation for debugging multiple bugs in arithmetic circuits*, IEEE Transactions on Computers, **68**, *2*, pp. 182–197 (2019).
11. D. Ritirc, A. Biere, and M. Kauers, *Column-wise verification of multipliers using computer algebra*, Formal Methods in Computer Aided Design (FMCAD), Vienna, Austria, IEEE, pp. 23–30 (2017).
12. B. Yu and M. Ciesielski, *Formal verification using don't-care and vanishing polynomials*, IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Pittsburgh, PA, USA, IEEE, pp. 284–289 (2016).
13. B. Yu, W. Brown, D. Liu, A. Rossi, and M. Ciesielski, *Formal Verification of Arithmetic Circuits by Function Extraction*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, **35**, *12*, pp. 2131–2142 (2016).
14. S. Ghandali, C. Yu, D. Liu, W. Brown, and M. Ciesielski, *Logic Debugging of Arithmetic Circuits*, In IEEE Computer Society Annual Symposium on VLSI, Montpellier, France, IEEE, pp. 113–118 (2015).
15. O. Sarbishei, B. Alizadeh, and M. Fujita, *Arithmetic circuits verification without looking for internal equivalences*, 6th ACM/IEEE International Conference on Formal Methods and Models for Co-Design, Anaheim, CA, USA, IEEE, pp. 7–16 (2008).
16. O. Sarbishei, M. Tabandeh, B. Alizadeh, and M. Fujita, *A formal approach for debugging arithmetic circuits*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, **28**, *5*, pp. 742–754 (2009).
17. M. Ciesielski, C. Yu, W. Brown, D. Liu, and A. Rossi, *Verification of gate-level arithmetic circuits by function extraction*, In 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), San Francisco, CA, USA, IEEE, pp. 1–6 (2015).
18. F. Farahmandi, B. Alizadeh, and Z. Navabi, *Effective combination of algebraic techniques and decision diagrams to formally verify large arithmetic circuits*, IEEE Computer Society Annual Symposium on VLSI, Tampa, FL, USA, IEEE, pp. 338–343 (2014).
19. M.R. Kumar and R. Sundaram, *Effective feature extraction method for unconstrained environment: local binary pattern or local ternary pattern*, Rev. Roum. Sci. Techn. – Électrotechn. et Énerg., pp. 449–454 (2024).
20. P. Nagarajan, T. Kavitha, N.A. Kumar, Al.S. Edward, *Power energy and power area product simulation analysis of master-slave flip-Flop*, Rev. Roum. Sci. Techn. – Électrotechn. et Énerg., pp. 325–330 (2023).
21. A. Ramaiah, P.D. Balasubramanian, A. Appathurai, and N.A. Muthukumaran, *Detection of Parkinson's disease via Clifford gradient-based recurrent neural network using multi-dimensional Data*, Rev. Roum. Sci. Techn. – Électrotechn. et Énerg., **69**, *1*, pp. 103–108 (2024).
22. L. Sriraman, K.S. Kumar, and T.N. Prabakar, *Design and FPGA implementation of binary squarer using Vedic mathematics*, Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT), pp. 1–5 (2013).