DOI: 30.0485/ijearst.10.03.3492 https://ijearst.co.in/

ENGINEERING IN ADVANCED RESEARCH SCIENCE AND TECHNOLOGY

ISSN 2386-7876
Vol.02, Issue.03
June-2026
Pages: 1160-1174

Design and Implementation of a Lightweight MLUT-Based Modular Multiplier Using ASC and OMS for Cryptographic Applications

[1]Y. ARUN KUMAR, [2]K. DEVI PRIYANKA
[1]M. Tech, Dept. of ECE, V S M College of Engineering, Ramachandrapuram, A.P, India.
[2]Assistant Professor, Dept. of ECE, V S M College of Engineering, Ramachandrapuram, A.P, India.

ABSTRACT: Modular multiplication is a fundamental operation in modern cryptographic algorithms such as RSA, ECC, and lattice-based systems, where achieving high throughput and reduced hardware complexity is critical. Traditional multiplier architectures suffer from large area, long critical paths, and significant power consumption, especially for large operand sizes used in security applications. To address these limitations, this project proposes an optimized **Manipulated Lookup Table (MLUT)**–based modular multiplier that exploits precomputed residue values stored in a structured LUT. By reorganizing and manipulating LUT entries based on operand bit patterns, the proposed design minimizes multiplier logic, decreases partial product generation time, and significantly reduces propagation delay. The MLUT architecture supports fast modular reduction and incorporates selective LUT activation to reduce unnecessary memory access, thereby improving energy efficiency. Implementation on FPGA shows reduced area utilization, lower dynamic power, and improved operating frequency compared to conventional Montgomery and Booth-based multipliers. The proposed design is highly suitable for lightweight cryptographic processors, IoT edge devices, and real-time embedded security systems requiring high-performance modular arithmetic. The methodology also provides scalability for different modulus sizes and serves as a foundation for advanced extensions such as pipelined MLUT, reconfigurable modulus support, and side-channel-resistant architectures. This work presents an enhanced Manipulated Lookup Table (MLUT) modular multiplier architecture that integrates Anti-Symmetric Coding (ASC) and Odd-Multiple Storage (OMS) to achieve significant LUT memory reduction. The proposed ASC technique leverages the algebraic anti-symmetry property of precomputed residues to eliminate redundant entries, reconstructing them on-the-fly using simple arithmetic operations. Complementarily, OMS stores only odd multiples of operands while generating even multiples through low-cost shift operations. Together, these optimizations reduce LUT storage requirements by up to 75%, minimize LUT access time, and shorten the critical path.

Keywords: Modular Multiplication, Lookup Table (LUT) Optimization, Manipulated Lookup Table Method, High-Performance Modular Multiplier, High-Speed Computation, Low-Latency Arithmetic, Area-Time Tradeoff, Throughput Optimization, Power-Efficient Design.

INTRODUCTION: Electronic communication is growing exponentially so should be the care for information security issues [10]. Data exchanged over public computer networks must be authenticated, kept confidential and its

integrity protected against alteration. In order to run successfully, electronic businesses require secure payment channels and digital valid signatures. Cryptography provides a solution to all these problems and many others. One of the main objectives of cryptography consists of providing *confidentiality*, which is a service used to keep secret publicly available information from all but those authorized to access it. There exist many ways to providing secrecy. They range from physical protection to mathematical solutions, which render the data unintelligible. The modular exponentiation is a common operation for scrambling and is used by several public-key cryptosystems, such Deffie and Hellman [8], [9] and the Rivest, Shamir and Adleman encryption schemes [4], as encryption/decryption method. RSA cryptosystem consiwsts of a set of three items: a *modulus M* of around 1024 bits and two integers *D* and *E* called *private* and *public* keys that satisfy the property $TDE \equiv T \mod M$. Plain text *T* obeying $0 \leq T < M$. Messages are encrypted using the public key as $C = TE \mod M$ and uniquely decrypted as $T = CD \mod M$. So the same operation is used to perform both processes: encryption and decryption. The modulus *M* is chosen to be the product of two large prime numbers, say *P* and *Q*. The public key *E* is generally small and contains only few bits set (i.e. bits = 1), so that the encryption step is relatively fast. The private key *D* has as many bits as the modulus *M* and is chosen so that $DE = 1 \mod (P-1)(Q-1)$. The system is secure as it is computationally hard toVdiscover *P* and *Q*. It has been proved that it is impossible to break an RSA cryptosystem with a modulus of 1024-bit or more. The modular exponentiation applies modular multiplication repeatedly. So the performance of publickey cryptosystems is primarily determined by the implementation efficiency of the modular multiplication and exponentiation. As the operands (the plaintext or the cipher text or possibly a partially ciphered text) are usually large (i.e. 1024 bits or more), and in order to improve time requirements of the encryption/decryption operations, it is essential to attempt to minimize the number of modular multiplications performed and to reduce the time required by a single modular multiplication. Modular multiplication $A \times B \mod M$ can be performed in two different ways: multiplying, i.e. computing $P = A \times B$; then reducing, i.e. $R = P \mod M$ or interleave the multiplication and the reduction steps. There are various algorithms that implement modular multiplication. The most prominent are Karatsuba-Ofman's [12] and Booth's [3] methods for multiplying, Barrett's [2], [6], [7] method for reducing, and Montgomery's algorithms [18], and Brickell's method [4], [7] for interleaving multiplication and reduction. curve cryptography (HECC) received much attention over the past years due to their small key sizes, which make them particularly suitable for resource constrained embedded devices. Well known schemes such as Diffie-Hellman key exchange or digital signature algorithm were extended to (H)ECC. In numerous works, these schemes were implemented on reconfigurable hardware, such as FPGAs, and optimized towards high-throughput and low-latency [1]– [6]. The performance of such schemes strongly depends on the implementation of the underlying field operations, e.g. modular addition, subtraction, multiplication, squaring, and inversion. In particular, the implementation of prime field multiplications should be optimized thoroughly because it requires a large amount of resources and time. Publickey cryptosystems using Mersenne prime field arithmetic recently gained in importance as Crandall's reduction procedure can be used [7]. Prominent examples are Microsoft's FourQ [8] and Kummer surface based key exchange [9] for elliptic and hyperelliptic curve cryptography respectively.

LITERATURE SURVEY: Savas, E (2005), introduced a new carry free Montgomery Inversion algorithm which is suitable for hardware implementation. It also utilizes a new Redundant Sign Digit (RSD) representation and arithmetic to avoid carry propagation in addition and subtraction. The proposed algorithm is described in such a way that its hardware realization is straightforward. The algorithm enables very fast computation of multiplicative inversion in prime field, which is the most time consuming operation in elliptic and hyper elliptic curve cryptography in the paper named "A Carry- Free Architecture for Montgomery Inversion". It focused on the time (152.0ns) and the number of slices (94 %) while implementing in hardware through CMOS (Complementary Metal Oxide Semiconductor) technology. Michalski et al (2006), improved high radix Montgomery multiplier design based on a limited resource ASIC (Application Specific Integrated Circuit) design, the multiple word Radix-2 Montgomery multiplication. An implementation of a complete RSA core is discussed for two Xilinx FPGA families, and a comparison between the hardware implementations and a public domain software implementation of RSA using Open SSL are discussed. The results show a 1024-bit RSA core on a 100MHz Virtex2 Pro 100 FPGA platform to be 3.13x faster than an equivalent software implementation on a 2.8 GHz Intel Xeon PC workstation in the paper named "Scalable Architecture for RSA Cryptography on Large FPGAs". It focused on the time (9.622ns) and the number of slices (90%) while implementing in hardware through Virtex-II Xilinx technology. Gary et al (2005), introduced a novel processor architecture for pairing-based cryptography applications using large characteristic curves in the article named "A Parameterizable Processor Architecture for Large Characteristic Pairing Based Cryptography". Bilinear Pairing is computationally expensive and inefficiently computed with general purpose processors. The architecture is parameterizable to fields with different bitwidths and different pairing algorithms. It takes advantage of some unique FPGA features such as huge aggregated memory and massively parallel computation logic to achieve high performance and low power efficiency.

EXISTING METHOD:

LOOKUP TABLE MANIPULATION:

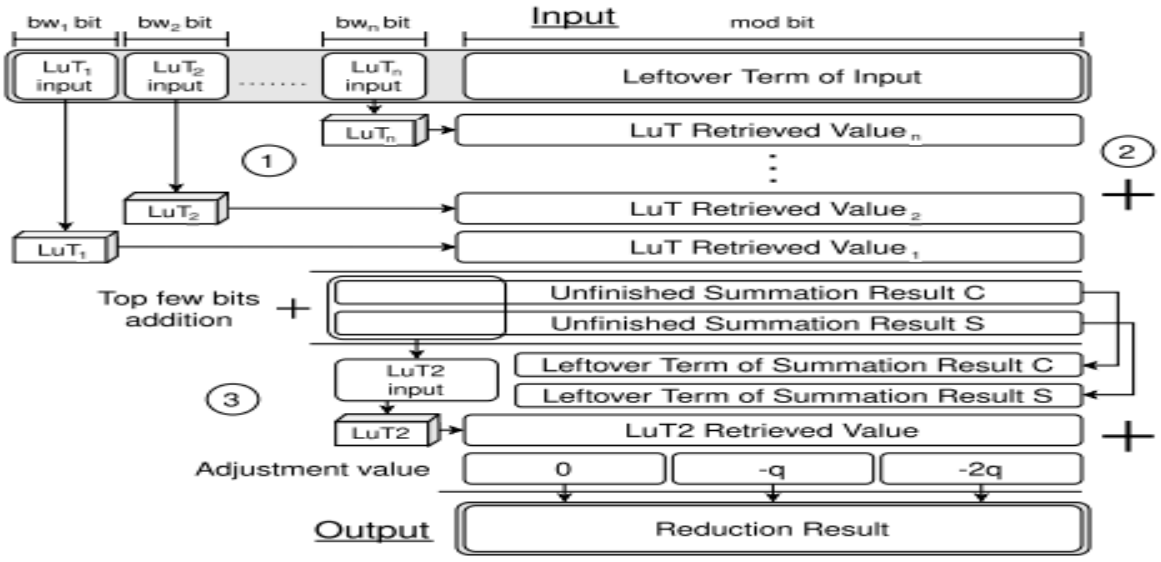

Fig:1 Existing block diagram

To enhance the performance of the LUT-LuT method for modular reduction, we propose to modify the content of LuT in a manner that Ham(LuT) is reduced without altering the reduction result. This manipulation is based on the fact that a constant m can be added to all entries in an LuT, then, the compensation value q − m can be added as an extra term during the summation step, increasing the total sum by q. Since the next step is the value adjustment, the final reduction result modulo q remains unchanged. With a total of L LuTs, a constant mi can be added to all entries in each LuTi , and the compensation value is computed by summing all mi values and subtracting from q, as shown in the following:

$$\text{Compensation value} = (q - \sum_{i=1}^{L} m_i) \mod q. \qquad (2)$$

As discussed earlier, Ham(LuT) directly affects the size and potentially the speed of the summation circuit. Therefore, manipulation values mi should be selected such that it reduces Ham(LuT) when added to the LuT entries

TWO MANIPULATION CASES

Case 1	Description	Continuous binary string of 1_b's
	Manipulation Steps	1. Place 1_b at the start 2. Fill other position with 0_b
Case 2	Description	Multiple continuous binary string of 1_b's separated by 0_b of length 1
	Manipulation Steps	1. Place 1_b at the start and at the location of 0_b 2. Fill other position with 0_b

Here, we propose a heuristic method to determine the manipulation value mi for each LuTi that minimizes the Hamming weight after adding this value to all the entries in the LuT, i.e., Ham(LuTi + mi) ≤ Ham(LuTi), where + denotes an addition to all entries in LuTi . We first explain the case for block width bw = 1, where there are two entries in the LuT, and then expand the concept to cover cases where block width bw > 1

A. bw = 1 LuT Manipulation 1) Basic Manipulation: In the case of bw = 1 (BW1), LuT contains two entries, located at address 0 and address 1. Based on the precomputation using (1), we know that LuT[0] = 0 and LuT[1]≠ 0. To find a manipulation value m that reduces Ham(LuT), we aim to exploit the presence of continuous 1's of the nonzero term of LuT. Suppose there is a sequence of y bits continuous 1's starting at position p. By adding the value 2p to both entries in LuT, we can reduce the number of output bits from y bits to just 2 bits, as shown in manipulation Case 1 of Table I. In this case, a reduction in Ham(LuT) occurs when y > 2. In the case where there are k − 1 chunks of one 0-bit separating k chunks of continuous 1's, each with a length of yj , we can add the value 2p at the start of the first chunk of 1's and at all 0-bit positions to reduce the number of output bits. This reduces the number of output bits from Pk j=1 yj bits to k +1 bits, as shown in manipulation Case 2 of Table I. A reduction in Ham(LuT) occurs when Pk j=1 yj > k + 1. Fig. 2 shows examples of how both manipulation cases reduce the LuT Hamming weight. Notice that the manipulation value m is designed to fill in nonzero values to become a power of two, which has a Hamming weight of one. Therefore, m can also be computed as m = 2 nCHUNK − CHUNK, where CHUNK refers to a segment in LuT[1] and nCHUNK is the length of CHUNK. To apply the two manipulation cases and find the manipulation value, we start from the least significant bit (LSB) and move toward the most significant bit (MSB) of a nonzero value v. We identify the matched manipulation case to construct the manipulation value whenever it is possible to

reduce Ham(LuT). The steps to find mi are formally summarized in Algorithm 1. This algorithm assumes a nonzero input v≠ 0 and is applicable to LuT configurations, where 0 ∈ LuT. In the bw = 1 case, v = LuT[1].

In this section, we describe the implementation of the three steps of LUT-based MLuT Modular Reduction (LUTMLuTMR), as shown in Fig.main block We also explain how this technique is integrated into LUT-MLuTMM. Since the only difference between the LuT and MLuT techniques lies in the content of LuT, the steps remain similar to the LuT method, and all implementation techniques described here are also applicable to the LuT method.

Retrieval Step: The physical LUTs in the FPGA are used for implementing the retrieval step. At bw = 1, there is no retrieval cost, and the reduction computation consists solely of the adder tree, making the MLuT method highly resemble the specialized reduction algorithms. The difference is that, in our case, all operands in the adder tree are guaranteed to be a nonnegative number, arguably simplifying the implementation In our target FPGAs (Virtex7 and Virtex Ultrascle+), a single-physical LUT, called LUT6_2, can accommodate up to six inputs and produce two outputs: one output is derived from a combination logic of all six inputs and another from five inputs (five out of the same six inputs). This means that for bw = 6, one LUT can generate a single-output bit.

Summation Step As the adder tree in the summation step of the LuT method can be quite large in certain parameters, constructing an efficient adder tree is crucial for optimizing the performance of the LuT method. While 3:2 compressors are commonly used to build adder trees in ASIC designs, in FPGA implementations, GPCs are widely regarded as the most efficient approach. Therefore, we use GPC in our implementations and explain the basics of GPC here. 1) GPC Units: A GPC unit produces the sum of the input bits and can be described by its input and output shapes, both of which consist of multiple columns of varying heights. These shapes are written starting from MSB to LSB, with column heights specified for each position.

PROPSOED METHOD: A new approach to LUT design is presented, where only the odd multiples of the fixed coefficient are required to be stored, which is referred to as the OMS. In addition, by the APC approach, the LUT size can also be reduced to half, where the product words are recoded as anti-symmetric pairs.

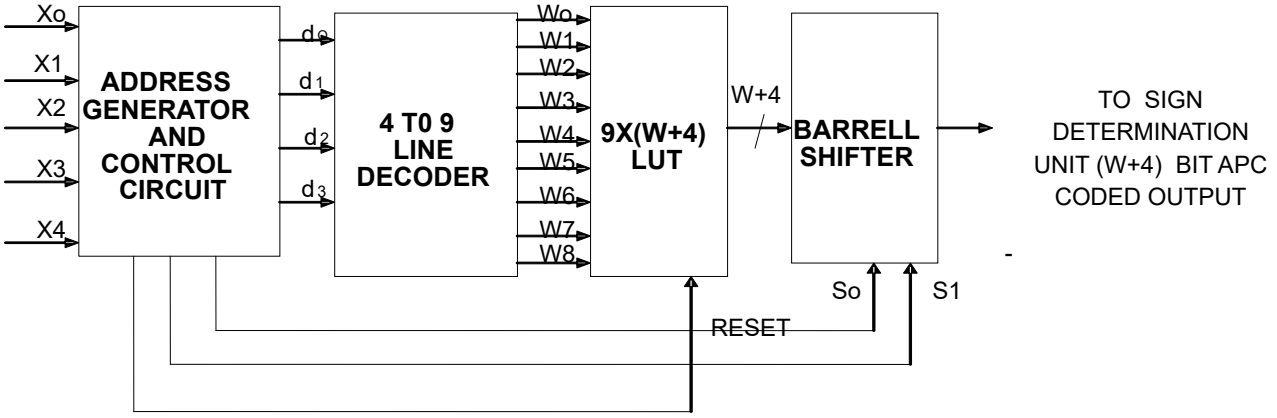

Figure 2 Proposed LUT Multiplier

The APC approach, although providing a reduction in LUT size by a factor of two, incorporates substantial overhead of area and time to perform the two's complement operation of LUT output for sign modification and that of the input operand for input mapping. However, it is found that when the APC approach is combined with the OMS technique the two's complement operations could be very much simplified since the input address and LUT output could always be transformed into odd integers.1 However, the OMS technique cannot be combined with the APC scheme, since the APC words generated according to are odd numbers. Moreover, the OMS scheme in does not provide an efficient implementation when combined with the APC technique. In this brief, a different form of APC is presented combined with a modified form of the OMS scheme for efficient memory based multiplication.

LUT APC Part

The structure and function of the LUT-based multiplier for $L = 5$ using the APC technique is shown in Fig.3.2 It consists of a four-input LUT of 16 words to store the APC values of product words as given in the sixth column of Table I, except on the last row, where $2A$ is stored for input $X = (00000)$ instead of storing a "0" for input $X = (10000)$. Besides, it consists of an address-mapping circuit and an add/subtract circuit. The address-mapping circuit generates the desired address (x3', x2', x1', x0'). A straightforward implementation of address mapping can be done by $X'L$ using $x4$ as the control bit. The address-mapping circuit, however, can be optimized to be realized by three XOR gates, three AND gates, two OR gates, and a NOT gate, as shown in Fig. 3.2 Note that the RESET can be generated by a control circuit (not shown in this figure) .The output of the LUT is added with or subtracted from $16A$, for $x4 = 1$ or 0, respectively, by the add/subtract cell. Hence, $x4$ is used as the control for the add/subtract cell.

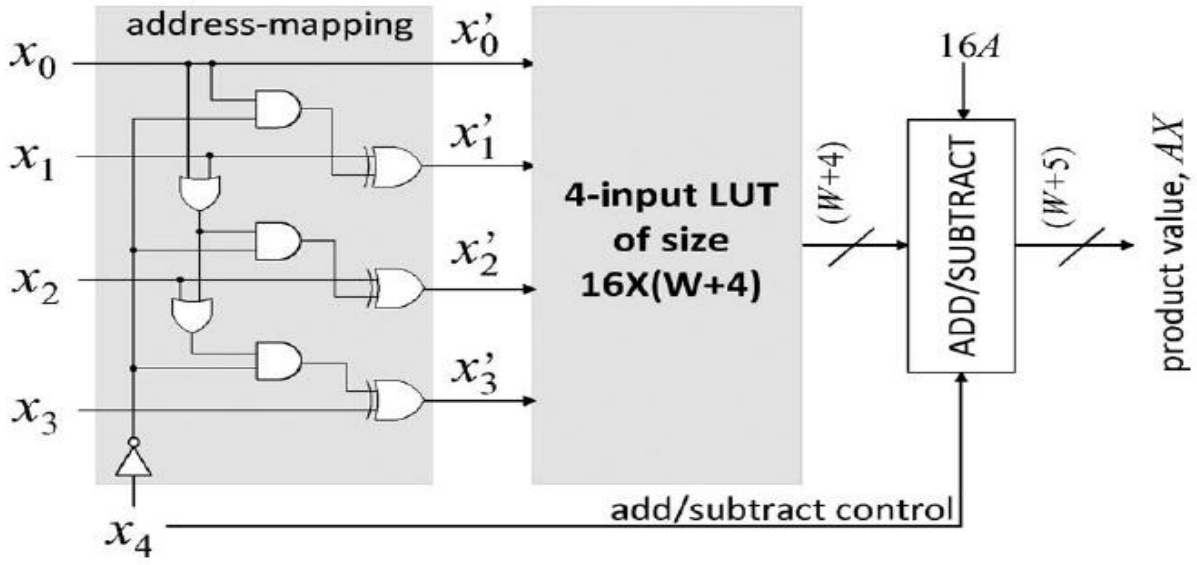

Figure 3 Proposed APC Part

For simplicity of presentation, it is assumed both X and A to be positive integers. The product words for different values of X for L = 5 are shown in Table I. It may be observed in this table that the input word X on the first column of each row is the two's complement of that on the third column of the same row. In addition, the sum of product values corresponding to these two input values on the same row is 32A. LUT based multiplier for L=5 using the APC technique W = Width of A; L = Width of X

Table: Stored APC Words

APC WORDS FOR DIFFERENT INPUT VALUES FOR $L = 5$

Input, X	product values	Input, X	product values	address $x_3'x_2'x_1'x_0'$	APC words
0 0 0 0 1	A	1 1 1 1 1	$31A$	1 1 1 1	$15A$
0 0 0 1 0	$2A$	1 1 1 1 0	$30A$	1 1 1 0	$14A$
0 0 0 1 1	$3A$	1 1 1 0 1	$29A$	1 1 0 1	$13A$
0 0 1 0 0	$4A$	1 1 1 0 0	$28A$	1 1 0 0	$12A$
0 0 1 0 1	$5A$	1 1 0 1 1	$27A$	1 0 1 1	$11A$
0 0 1 1 0	$6A$	1 1 0 1 0	$26A$	1 0 1 0	$10A$
0 0 1 1 1	$7A$	1 1 0 0 1	$25A$	1 0 0 1	$9A$
0 1 0 0 0	$8A$	1 1 0 0 0	$24A$	1 0 0 0	$8A$
0 1 0 0 1	$9A$	1 0 1 1 1	$23A$	0 1 1 1	$7A$
0 1 0 1 0	$10A$	1 0 1 1 0	$22A$	0 1 1 0	$6A$
0 1 0 1 1	$11A$	1 0 1 0 1	$21A$	0 1 0 1	$5A$
0 1 1 0 0	$12A$	1 0 1 0 0	$20A$	0 1 0 0	$4A$
0 1 1 0 1	$13A$	1 0 0 1 1	$19A$	0 0 1 1	$3A$
0 1 1 1 0	$14A$	1 0 0 1 0	$18A$	0 0 1 0	$2A$
0 1 1 1 1	$15A$	1 0 0 0 1	$17A$	0 0 0 1	A
1 0 0 0 0	$16A$	1 0 0 0 0	$16A$	0 0 0 0	0

For $X = (0\ 0\ 0\ 0\ 0)$, the encoded word to be stored is $16A$.

16 x (W+4) → 16 Locations and each location having (W+4) bits.

. Let the product values on the second and fourth columns of a row be u and v, respectively. Since one can write u = [(u + v)/2 − (v − u)/2] and

v = [(u + v)/2 + (v − u)/2], for (u + v) = 32A,

U=16A+ [(V-U)/2] V=16A-[(V-U)/2]

The product values on the second and fourth columns of Table I therefore have a negative mirror symmetry. This behavior of the product words can be used to reduce the LUT size, where, instead of storing u and v, only [(v − u)/2] is stored for a pair of input on a given row. The 4-bit LUT addresses and corresponding coded words are listed on the fifth and sixth columns of the table, respectively. Since the representation of the product is derived from the antisymmetric behavior of the products, we can name it as antisymmetric product code. The 4-bit address X_ = (x_3,x_2,x_1,x_0) of the APC word is given by

$$X' \begin{cases} X_L, \text{if } x4=1 \\ X'_L, \text{if } x4=0 \end{cases}$$

Flow Chart For Proposed APC Part

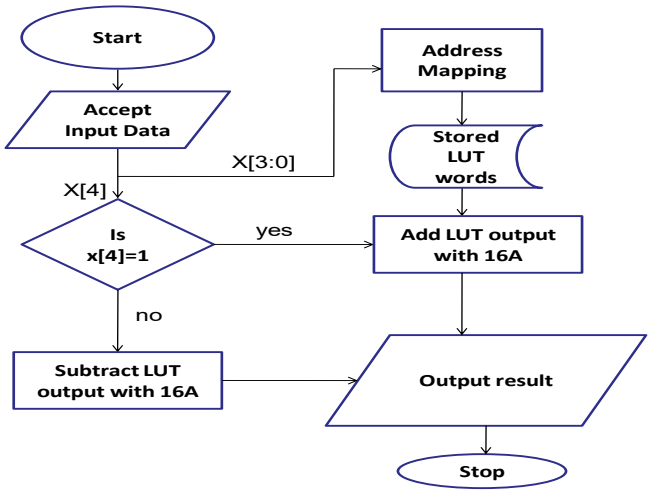

Figure 4 Flow Chart for Proposed LUT

DOI: 30.0485/ijearst.10.03.3492 https://ijearst.co.in/

Proposed APC-OMS Part

For the multiplication of any binary word X of size L, with a fixed coefficient A, instead of storing all the 2L possible values of C = A · X, only (2L/2) words corresponding to the odd multiples of A may be stored in the LUT, while all the even multiples of A could be derived by left-shift operations of one of those odd multiples. Based on the above assumptions, the LUT for the multiplication of an L-bit input with a W-bit coefficient could be designed by the following strategy.

1) A memory unit of [(2L/2) + 1] words of (W + L)-bit width is used to store the product values, where the first (2L/2) words are odd multiples of A, and the last word is zero.

2) A barrel shifter for producing a maximum of (L − 1) left shifts is used to derive all the even multiples of A.

3) The L-bit input word is mapped to the (L − 1)-bit address of the LUT by an address encoder, and control bits for the barrel shifter are derived by a control circuit.

Table 2: Stored APC-OMS Words

input X' $x_3x_2x_1x_0$	product value	# of shifts	shifted input, X''	stored APC word	address $d_3d_2d_1d_0$
0 0 0 1	A	0	0 0 0 1	P0 = A	0 0 0 0
0 0 1 0	2 × A	1			
0 1 0 0	4 × A	2			
1 0 0 0	8 × A	3			
0 0 1 1	3A	0	0 0 1 1	P1 = 3A	0 0 0 1
0 1 1 0	2 × 3A	1			
1 1 0 0	4 × 3A	2			
0 1 0 1	5A	0	0 1 0 1	P2 = 5A	0 0 1 0
1 0 1 0	2 × 5A	1			
0 1 1 1	7A	0	0 1 1 1	P3 = 7A	0 0 1 1
1 1 1 0	2 × 7A	1			
1 0 0 1	9A	0	1 0 0 1	P4 = 9A	0 1 0 0
1 0 1 1	11A	0	1 0 1 1	P5 = 11A	0 1 0 1
1 1 0 1	13A	0	1 1 0 1	P6 = 13A	0 1 1 0
1 1 1 1	15A	0	1 1 1 1	P7 = 15A	0 1 1 1

RESULTS:

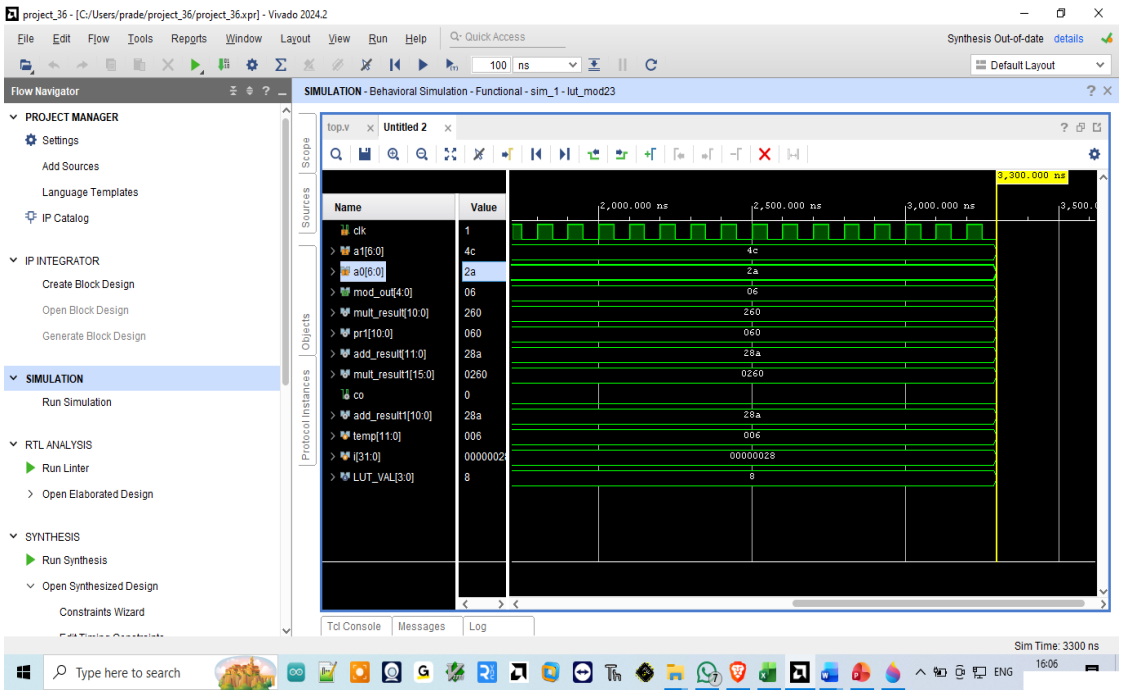

Fig a: Proposed Simulation result

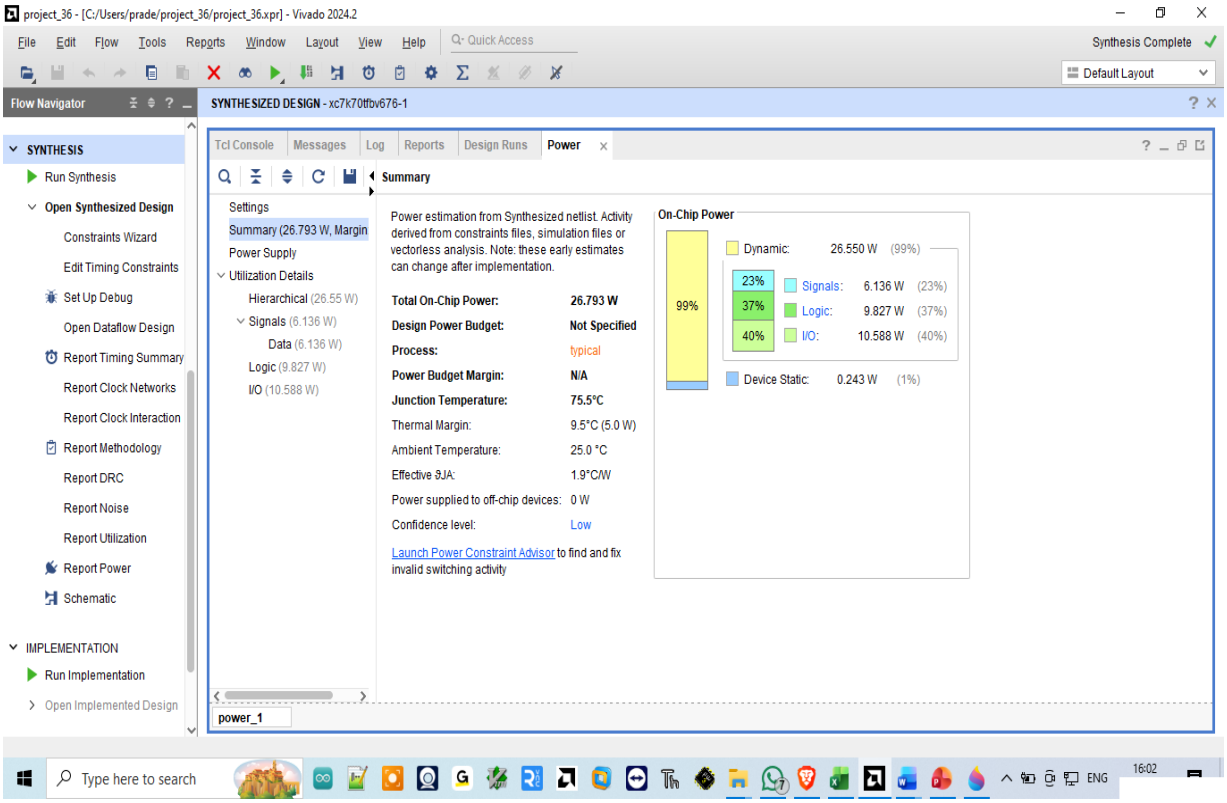

Fig b: Existing Power result

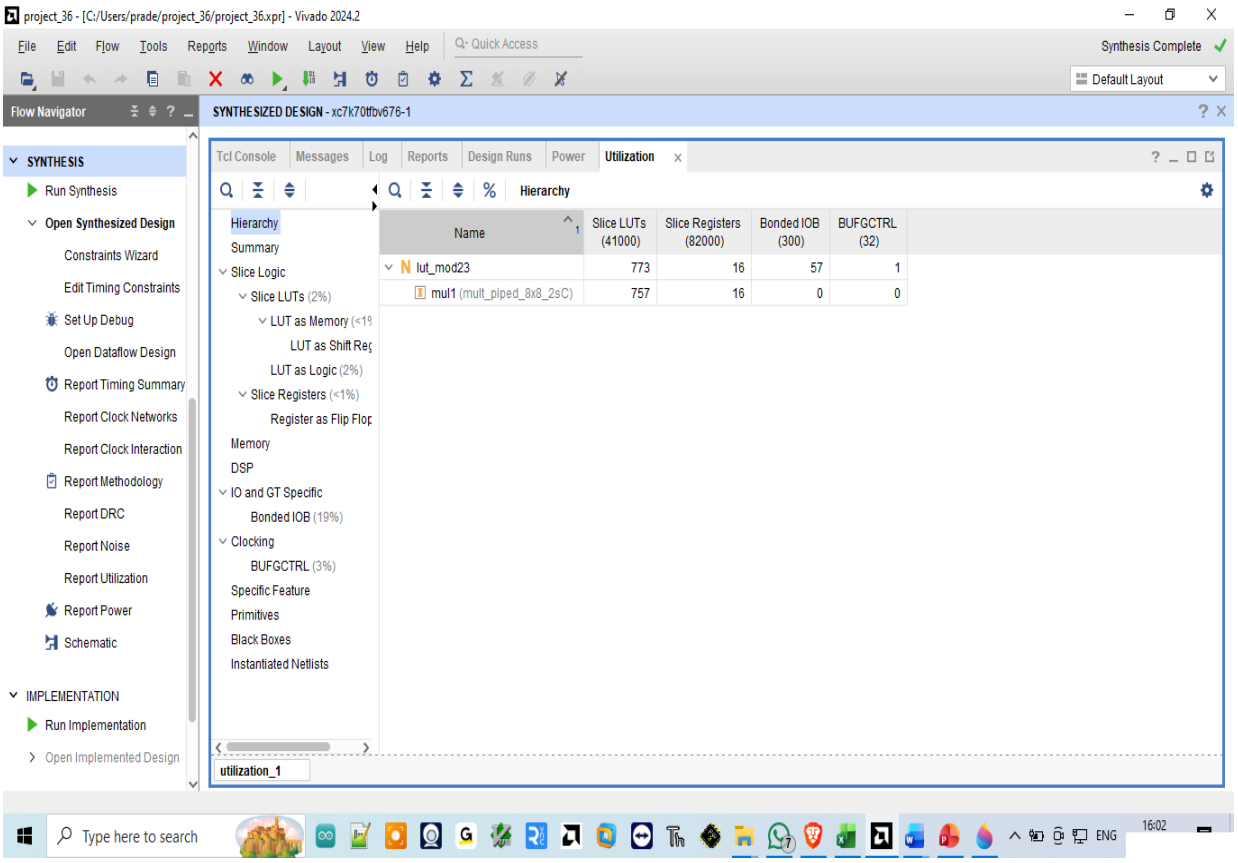

Fig c: Existing Hardware utilization

DOI: 30.0485/ijearst.10.03.3492 https://ijearst.co.in/

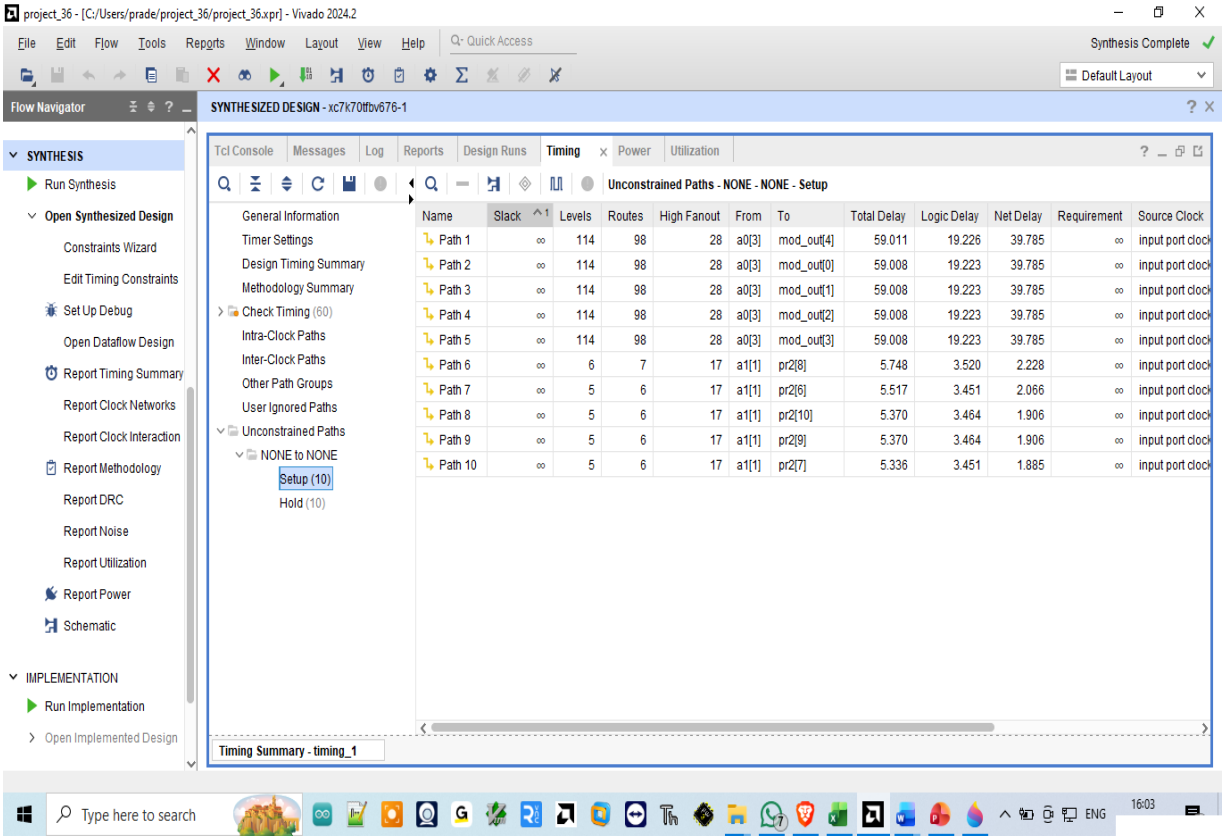

Fig d: Existing Timing Summary

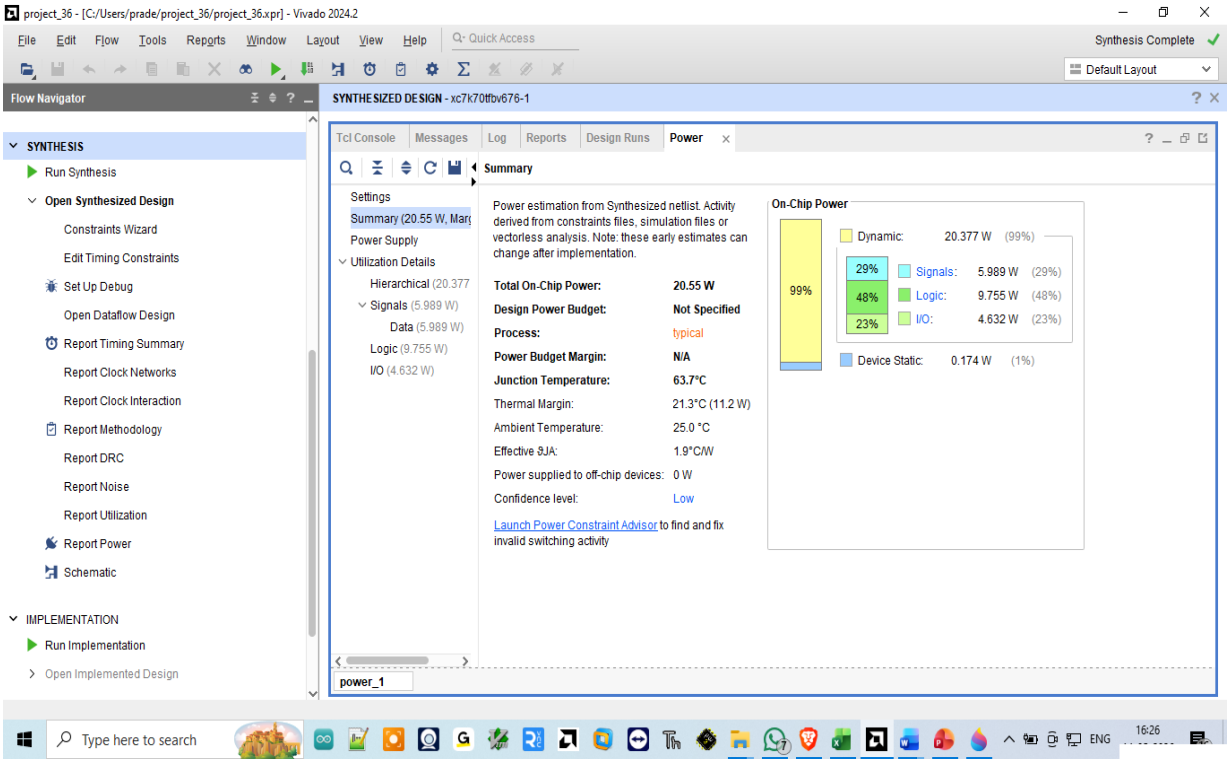

Fig e: Proposed Power report

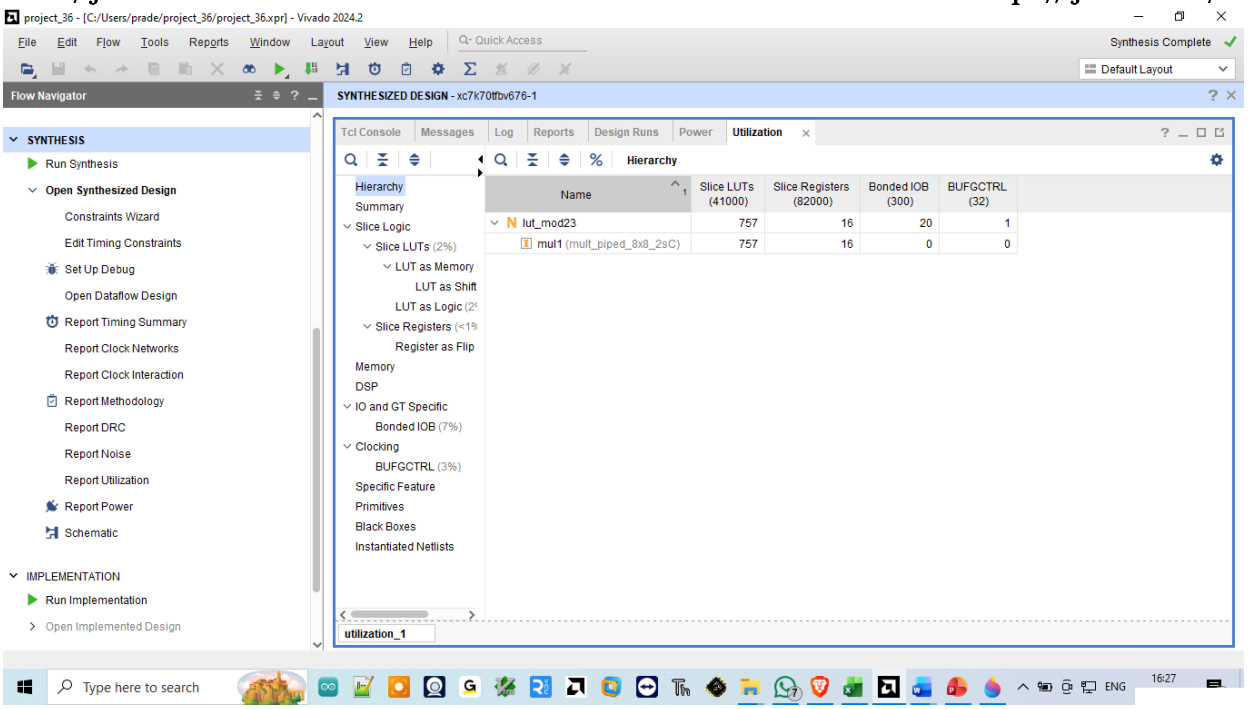

Fig f: Proposed Utilization summary

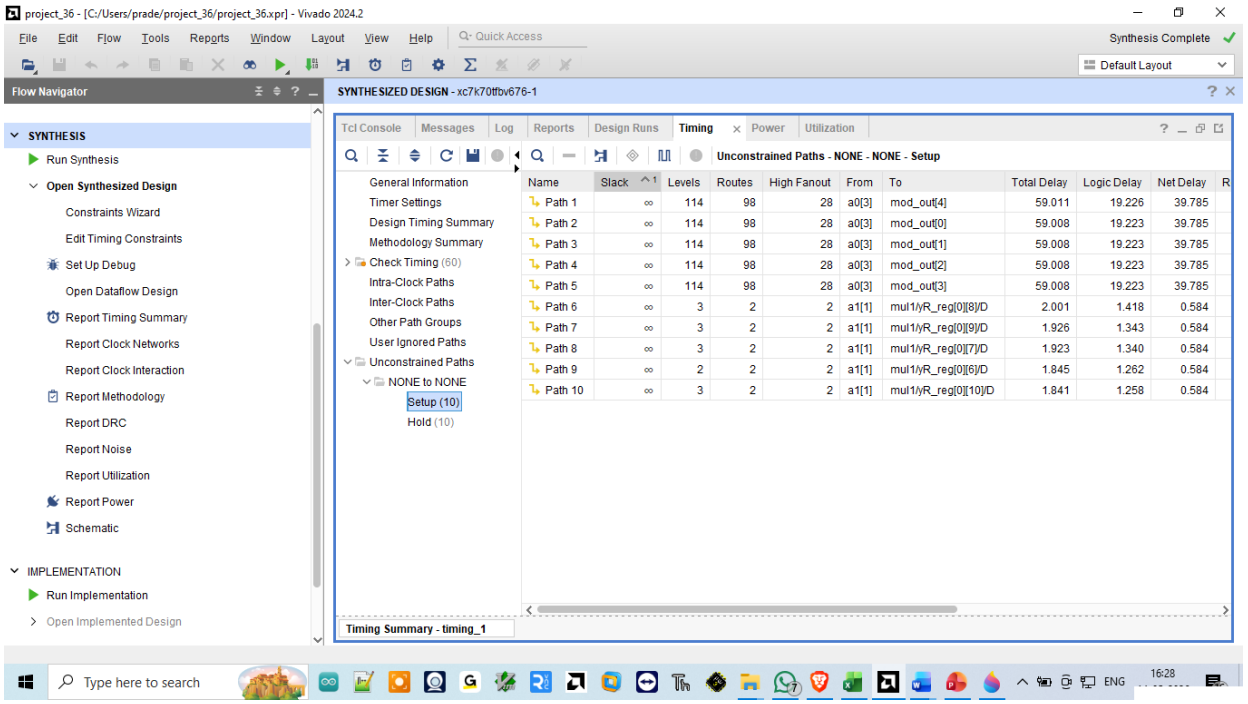

Fig g: Proposed timing summary

Advantages

1. High Speed Operation

   The MLUT method significantly reduces modular multiplication latency by replacing repeated division operations with pre-computed lookup values. This makes it faster than conventional modular reduction techniques.

2. Reduced Computational Complexity

   Traditional modular multiplication uses iterative division or Montgomery reduction. MLUT avoids complex division operations and uses simple table indexing and addition, reducing computational overhead.

3. Low Power Consumption

   Since the number of arithmetic operations is reduced, switching activity in hardware decreases. This leads to lower dynamic power consumption, making the design energy-efficient.

4. Area Efficient Hardware Implementation

   The architecture replaces large arithmetic blocks with optimized memory (lookup tables), reducing gate count and silicon area in FPGA/ASIC implementations.

5. Improved Throughput

   Parallel table access and pipelined architecture enable high throughput suitable for real-time applications.

Applications

1. **Cryptographic Systems**
   - RSA encryption/decryption
   - Elliptic Curve Cryptography (ECC)
   - Diffie–Hellman key exchange

     Modular multiplication is the core operation in public-key cryptography.
2. **Blockchain and Digital Signatures**
   - Cryptocurrency mining
   - Secure transaction verification
   - Digital signature generation (DSA, ECDSA)
3. **Secure Communication Systems**
   - SSL/TLS protocols
   - Secure IoT communication
   - End-to-end encrypted messaging
4. **Digital Signal Processing (DSP)**
   - Residue Number System (RNS) processing
   - Error detection and correction algorithms
   - Fast arithmetic in DSP processors

Conclusion

The Manipulated Lookup Table (MLUT) method provides an efficient solution for high-performance modular multiplication by replacing expensive division operations with optimized pre-computed lookup tables. This approach achieves faster computation, reduced power consumption, and smaller hardware area compared to conventional techniques such as Montgomery and Barrett multiplication. The architecture is highly suitable for modern cryptographic and embedded systems where speed, energy efficiency, and security are critical. Overall, modified

LUT demonstrates strong potential for next-generation hardware accelerators used in secure communication and data protection systems.

Future Scope

1. **Support for Larger Key Sizes**
   Extend the architecture to support 256-bit, 512-bit, and 1024-bit cryptographic key sizes used in modern security standards.
2. **Integration with Cryptographic Processors**
   Implement MLUT as a dedicated hardware accelerator in SoC and RISC-V based secure processors.
3. **FPGA and ASIC Optimization**
   Develop optimized implementations for different FPGA families and ASIC technology nodes to further reduce area and power.
4. **AI-Based Table Optimization**
   Use machine learning techniques to optimize lookup table generation and compression for faster access and reduced memory usage.

REFERENCES:

[1] H. Zhou, C. Liu, L. Yang and F. Yang, "A Fully Pipelined Reconfigurable Montgomery Modular Multiplier Supporting Variable Bit-Widths," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, pp. 4653-4665, 2024.

[2] S. Gribok, M. Langhammer and B. Pasca, "FPGA Modular Multipliers Using Hybrid Reduction Techniques," *Proc. IEEE FPL*, pp. 288-296, 2024.

[3] M. Aqeel Aslam and A. Bilal, "Efficient Implementation of Modular Multiplication Using Carry Look-Ahead Adder," *Int. J. Eng. Res. Technol.*, vol. 2, no. 12, 2013.

[4] K. Masada, R. Nakayama and M. Ikeda, "Lookup Table Modular Reduction: A Low-Latency Modular Reduction for Fast ECC Processor," Proc. IEEE Conf., 2022.

[5] J. Ding and S. Li, "A Low-Latency and Low-Cost Montgomery Modular Multiplier Based on NLP Multiplication," *IEEE Access*, 2019.

[6] P. L. Montgomery, "Modular Multiplication Without Trial Division," *Mathematics of Computation*, vol. 44, no. 170, pp. 519-521, 1985.

[7] C. D. Walter, "Montgomery Exponentiation Needs No Final Subtractions," *Electronics Letters*, vol. 35, no. 21, pp. 1831-1832, 1999.

[8] A. K. Lenstra, "Integer Factoring Using Elliptic Curves," *Annals of Mathematics*, vol. 126, pp. 649-673, 1987.

[9] T. Blum and S. Paar, "High-Radix Montgomery Multipliers," *IEEE Transactions on Computers*, vol. 50, no. 2, pp. 185-189, 2001.

[10] J. Großschädl, "A Bit-Serial Unified Multiplier Architecture for Finite Fields GF(p) and GF(2m)," *IEEE Transactions on Computers*, vol. 57, no. 5, pp. 690-701, 2008.

[11] A. Daly and W. Marnane, "Efficient Architectures for Implementing Montgomery Modular Multiplication and RSA Modular Exponentiation on Reconfigurable Logic," *IEEE Trans. Computers*, vol. 53, no. 9, pp. 1065-1077, 2004.

[12] J. McIvor, M. McLoone and J. McCanny, "Fast Montgomery Modular Multiplication and RSA Cryptographic Processor Architectures," *IEEE Trans. Computers*, vol. 53, no. 11, pp. 1398-1412, 2004.

[13] M. Huang, K. Gaj and T. El-Ghazawi, "New Hardware Architectures for Montgomery Modular Multiplication," *IEEE Trans. Computers*, vol. 60, no. 7, pp. 923-936, 2011.

[14] S. Öztürk and E. Savaş, "Montgomery Modular Multiplication on FPGA," *IEEE Design & Test of Computers*, vol. 23, no. 3, pp. 216-223, 2006.

[15] A. Tenca and C. K. Koç, "A Scalable Architecture for Montgomery Multiplication," *CHES*, LNCS, pp. 94-108, 1999.

[16] C. K. Koç, "High-Speed RSA Implementation," *RSA Laboratories*, 1994.

[17] S. P. Gopal and M. A. Hasan, "Low Complexity Digit-Serial Modular Multiplication for Public Key Cryptography," *IEEE Trans. VLSI Systems*, vol. 14, no. 3, pp. 239-247, 2006.

[18] A. Satoh and S. Morioka, "Unified Hardware Architecture for 128-Bit Block Ciphers AES and Camellia," *CHES*, 2003.

[19] J. Großschädl and G. Gaubatz, "Energy-Efficient Cryptography," *IEEE Security & Privacy*, vol. 4, no. 2, pp. 38-45, 2006.

[20] K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*, Wiley, 1999.

[21] S. Kumar and G. Sathish, "Modular Multiplication Algorithm in Cryptographic Processor: A Review and Future Directions," *IJACEE*, 2017.

[22] J. Ku et al., "ModSRAM: Algorithm-Hardware Co-Design for Large Number Modular Multiplication in SRAM," *arXiv*, 2024.

[23] W. Tan et al., "High-Speed VLSI Architectures for Modular Polynomial Multiplication," *arXiv*, 2021.

[24] A. De et al., "Fast Integer Multiplication Using Modular Arithmetic," *arXiv*, 2008.

[25] H. Orup, E. Svendsen and E. Andreasen, "VLSI Implementation of a Fast Parallel Modular Multiplier," *IEEE Trans. Computers*, vol. 44, no. 12, pp. 1365-1372, 1995.