SAUCIS

Sakarya University Journal of Computer and Information Sciences

http://saucis.sakarya.edu.tr/

SAKARYA UNIVERSITY

DOI: 10.35377/saucis...1229353

RESEARCH ARTICLE

High-Capacity Data Processing with FPGA-Based Multiplication Algorithms and the Design of a High-Speed LUT Multiplier

Kenan Baysal [1], **Deniz Taşkın** [2]

[1] Information Management, Hayrabolu Vocational High School, Tekirdağ Namık Kemal University Tekirdağ/Türkiye

[2] Computer Engineering, Engineering Faculty, Trakya University Edirne/Türkiye

Corresponding author:
Kenan Baysal, Tekirdağ Namik Kemal Unniversity, Hayrabolu Vocational High School, Tekirdağ/Türkiye
E-mail address:
kbaysal@nku.edu.tr

Submitted: 04 January 2023
Revision Requested: 16 October 2023
Last Revision Received: 04 November 2023
Accepted: 07 November 2023
Published Online: 07 November 2023

Citation: Baysal K. and Taşkın D. (2023). High-Capacity Data Processing with FPGA-Based Multiplication Algorithms and the Design of a High-Speed LUT Multiplier *Sakarya University Journal of Computer and Information Sciences*. 6 (3)
https://doi.org/10.35377/saucis...1229353

ABSTRACT

Encryption algorithms work with very large key values to provide higher security. To process high-capacity data in real time, we need advanced hardware structures. Today, compared to previous design methods, hardware solutions can be designed more easily using Field-Programmable Gate Arrays (FPGAs). Over the past decade, FPGA speeds, capacities, and design tools have been improving. Thus, the hardware that can process high-capacity data can be designed and produced with lower costs. This study aims to create the components of a high-speed arithmetic unit that can process high-capacity data and be used for FPGA encoding algorithms.
In this study, multiplication algorithms were analyzed. High-capacity adders that constitute high-speed multiplier and look-up tables were designed using Very High-Speed Integrated Circuit Hardware Description Language (VHDL). The designed circuit/multiplier was synthesized with ISE Design Suite 14.7 software. Simulation results were obtained using the ModelSIM and ISIM programs.

Keywords: FPGA, VHDL, look up table, multiplication, adder

1. Introduction

As the computers reach the physical limits of the power of arithmetic operations, the command structures of the processors and how they process these commands have become significant [1]. The operations performed by the processors are based on arithmetic and logic commands [2]. The speed at which arithmetic operations, such as addition and multiplication, directly affect the processor's data processing capacity.

Since FPGA has been used for electronic circuit designs created using transistors, complex circuit designs have become easily and quickly achievable. Since companies like Altera, Xilinx, Actel, etc., started FPGA production and developed more easy-to-use design and simulation tools and equipment in the mid-1980s, the lower-level designers began to program their own FPGA hardware on tighter budgets.

In this study, we analyzed the high-speed multiplication methods, which have a significant role in the data processing capacity and speed of the computers, to create sub-units of the high-speed arithmetic unit. A high-speed expandable adder was designed. A high-speed LUT multiplier, expandable through look-up tables, was designed using a high-speed adder. A 32x32-bit long LUT multiplier was modeled with VHDL and applied to FPGA hardware.

The study aims to design efficient and cost-effective hardware solutions for processing high-capacity data in real time, particularly in encryption algorithms requiring large key values. The primary objective of the study is to create the components of a high-speed arithmetic unit that can process high-capacity data, which are designed to be used in FPGA-based encryption algorithms. The study focuses on analyzing multiplication algorithms and designing high-capacity adders for building high-speed multipliers. Additionally, look-up tables are designed using Very High-Speed Integrated Circuit Hardware Description Language (VHDL). The entire circuit and multiplier design is synthesized using ISE Design Suite 14.7 software, and simulation results are obtained through ModelSIM and ISIM programs. The study aims to advance the field of hardware design for encryption algorithms, offering efficient and cost-effective solutions for processing large key values in real time. The study's findings can contribute to developing more secure and efficient encryption algorithms, essential for protecting sensitive data in various healthcare, finance, and government applications.

2. Literature Review

Abd-Elkader et al. developed a design model to improve the performance of the hardware structure of the Montgomery Modular Multiplier. They used VHDL to code their proposed model and found that it consumed fewer resources on the FPGA. Their study showed that the proposed model could operate more efficiently than the existing hardware structure [3].

Behl et al. developed a multiplier circuit that reduces carry propagation time and performs faster calculations using the Redundant Binary Signed Digit number system. They encoded this circuit in VHDL and applied it on an FPGA. Using the VIVADO multiplier in their design, they observed a significant reduction in the number of Look-up tables used. This approach can potentially improve the efficiency of digital electronics, such as binary multipliers, and could be useful in various applications. [4].

Sakali et al. have introduced a new error analysis approach to reduce hardware redundancy in existing fault-tolerant techniques. This approach is particularly useful for reducing the additional hardware resources that Triple Modular Redundancy (TMR) requires. They applied the proposed approach to a multiplier circuit and found a 48% reduction in hardware resource usage. [5].

Malathi et al. have explored a new approach to enhancing image quality and resolution on FPGA using deep learning and Fast Fourier Transform (FFT) techniques. They tackled this approach in three main stages: noise reduction, segmentation, and resolution enhancement. This method achieved low power consumption, minimal latency, and high efficiency. This approach can potentially improve the quality of images in various applications, including medical imaging and surveillance. Deep learning and FFT techniques allow for greater flexibility and customization in image processing, making it a valuable tool for image enhancement. Applying this approach to FPGA allows for efficient and fast processing of images, making it a promising technology for real-time image processing. [6].

Bianchi et al. have proposed an architecture for a new Vedic multiplier that employs the 'Urdhava-tiryakbhyam' method and implemented it on an FPGA. They evaluated this approach regarding hardware performance, LUT (Look-Up Table) sizes, and propagation delay. The results have shown performance equal to or better than existing approaches in the literature. [7].

Özcan et al. have proposed a fast Montgomery multiplier design for modern FPGAs. Their designs implemented on Virtex-7 have provided competitive performance and significant savings in FPGA logic resources.[8].

Morales-Sandoval et al. discussed using Field-Programmable Gate Arrays (FPGAs) to implement Montgomery Multiplication in public-key encryption algorithms like RSA and Elliptic Curve Cryptography (ECC). Their study focused on area-performance trade-offs, tested different architectures, and compared their efficiencies with previous FPGA Montgomery multipliers. [9].

3. Multiplication Algorithms

All arithmetic operations made on a computer are based on addition. Multiplication, the basis of many scientific practices, such as encoding, decoding, signal processing, etc., is also realized based on shifting and addition. The structure of multiplication algorithms is the most significant factor that affects the computer's performance, especially when we work on high-capacity data in scientific programs. Approximately 9% of this scientific program consists of multiplication [10]. Thus, a wide array of multiplication algorithms has been developed in the years following computer technology developments. The multiplication algorithm, which will be selected according to the size of the data and suitability of the algorithm, can increase the speed of the process without enhancing the performance of the computer's processor.

For an n bit-long number of X and m bit-long number Y in an ordinary multiplication, as seen in Equation 1, the equation can indicate multiplication [5].

$$P(m+n) = X(n)Y(m) = \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} x(i)y(j)2^{i+j} \tag{1}$$

3.1. Sequential Shift and Add Multiplication

In this method, we start with the lowest-weight bit rate of the multiplier and obtain results by shifting the multiplicand to the left and summing up at each clock stroke, depending on whether the bit rate is either 1 or 0. Although this method is based on a very simple logic, the performance is adversely affected if the data size is too large.

3.2. Booth Algorithm

In the booth algorithm, the numbers in both marked bases are multiplied based on the add-shift principle by comparing adjacent bits through a reciprocating operation of 2. The advantage of this method, when compared to other multiplication algorithms, is that there are fewer additions and multiplications. [11].

3.3. Wallace Tree

Wallace tree multiplication is an effective method that may be preferred at hardware-level multiplication of two unmarked integer numbers. This method was developed by computer scientist Chris Wallace in 1964 [12].

In the Wallace Tree method, the partial fractions are added up as a tree until they reach the last two partial fraction rows that will be added at the final stage. The speed of this algorithm is inversely proportional to the number of bits to be processed. The waste of unused space and its complex structure are some of the primary problems of this algorithm [13]. Its structure is not suitable for rapid transit logic in hardware structure. It is also not as fast as the transit structure within modern FPGAs [14].

3.4. Array Multiplication

It is a common multiplication algorithm with a parallel index structure based on the add-shift principle. Partial result rates are obtained by multiplying the respective 1-bit digits of the multiplier by the multiplicand and adding up by shifting per the bit order. Although its structure is slow as an algorithm, this method is often preferred for its parallel-moving index structure is systematic and easy to position on the hardware [15].

3.5. Karatsuba Multiplication Algorithm

This is an effective multiplication method for multiplying two large numbers. The numbers to be multiplied are divided into sub-groups. The results are obtained by adding the results obtained by multiplying these sub-groups. This method provides a great advantage in multiplying large numbers [16]. While the length of the operation is $O(n^2)$ in traditional methods, the length becomes $O(n^{\log(2)3})$ with the Karatsuba method [17].

4. Material and Methods

4.1. Look-Up Table Multiplication

This efficient and fast multiplication method is suitable for hardware structures with strong memory units. Look-up table multipliers are generated using block memory units, which store multiplication results corresponding to all input values. The results are obtained in a shorter period since no real multiplications are performed through this method. However, the biggest disadvantage of this method is that the look-up table size increases incrementally as the number increases. Since the memory unit includes all multiplication possibilities, some data may take up space in memory even though they are never used. However, in cases where continuous and unstable signal routing, such as encoding, decoding, image processing etc., is present, the real-time multiplication performance is very high.

If the input data length of the look-up table is k-bit in multiplication, the potential number of results contained in the look-up table would be 2^{2k}. However, when at least one input is zero, $2^{2k}/(2.2^k - 1)$ number of results would be zero. This enables us to send a zero value directly to the output without taking up space in the look-up table. Thus, depending on the state of the input bit length from the memory unit, it would be possible to save one $2^{2k}/(2.2^k - 1)$ of space.

Multiplying directly with the look-up table in large data sizes may not be practical due to the space the look-up table takes up in the memory unit. In this case, the duration of the standard multiplication may be shortened by using a partial look-up table.

The number is divided into factions by the bit length of the look-up table, and partial results are obtained according to the result of the look-up table. K-bit left-shift is performed according to the bit length and the number of steps in the look-up table when the partial results are obtained. While the multiplication is performed in the standard shift-add form, the result can be seen directly in the look-up table instead of obtaining results through performing the multiplication physically. With this method, the k-bit length of the operational cycle would be saved.

4.2. n Bit High-Speed LUT Multiplier Design

To multiply n bit-long numbers A and B, the numbers are divided into k bit-long $\frac{n}{k}$ fractions. Being i ≤ $\frac{n}{k}$ and j ≤ $\frac{n}{k}$, partial results are obtained by using A_j and B_i number fractions look-up table at each t time. The partial results are summed up, and the multiplication result is found. The adder used at the multiplier is obtained by increasing and expanding a 1-bit adder accordingly. It is now possible to perform a 2n bit-long addition at a single clock stroke with the advantage it provides.

In a multiplication performed with a lookup table by using an expandable adder, the relationship between the multiplication by the number of operational steps (t) and the bit length (n) of the multiplicands and fraction bit-length (k) can be expressed by the Equation 2.

$$t = 2^{2[\log_2(n) - \log_2(k)]} \tag{2}$$

The look-up table size should be calculated by considering the capacity of the FPGA chip where the multiplier will be applied. The number of possibilities of the look-up table can be expressed as in Equation 3.

$$LUT\ possibility\ number\ = 2^{2k} \tag{3}$$

Because the output will be zero if the input is zero regardless of the state of the other input, the number of possibilities may be reduced as seen in Equation 4.

$$LUT\ possibility\ number\ = 2^{2k} - 2.2k - 1 \tag{4}$$

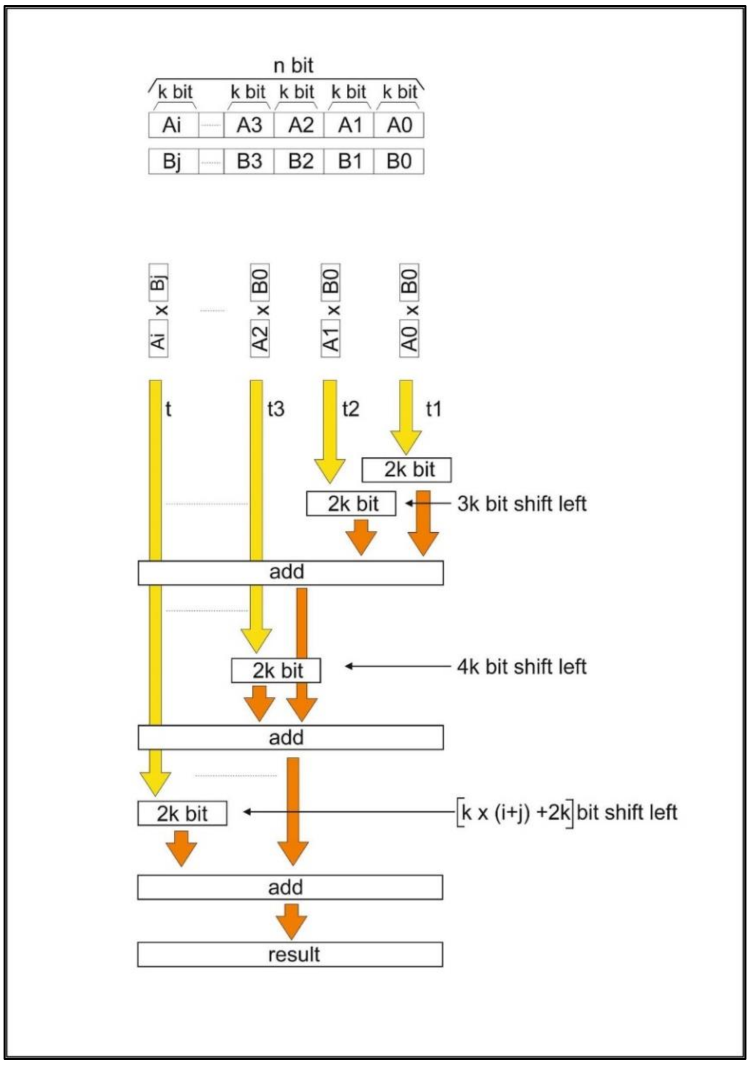


Figure 1 Shifting and adding.

Figure 1 shows how many bits of the Aj and Bi fractions were shifted to the left according to the current condition before the addition.

Being $t_a \leq t$, according to ta current condition, the relationship between fraction j of k bit-long n-bit A multiplicand data and i fraction of n-bit B multiplier data at LUT inputs can be expressed as in the Equation 5 and the Equation 6.

$$j = t_a\left(mod\ \frac{n}{k}\right) \tag{5}$$

$$i = \left[t_a - t_a\left(mod\ \frac{n}{k}\right)\right]\left(mod\ (\frac{n}{k} - 1)\right) \tag{6}$$

Thus, in the case of ta, the multiplicand fractions of A and B data can be expressed as in Equation 7 and Equation 8.

$$St_a \rightarrow\ A_j\ x\ B_i \tag{7}$$

$$St_a\ \rightarrow\ A\left[t_a\left(mod\ \frac{n}{k}\right)\right]\ \ x\ B\left[t_a - t_a\left(mod\ \frac{n}{k}\right)\left(mod\ (\frac{n}{k} - 1)\right)\right] \tag{8}$$

Table 1 Look-up table and number of steps in different fraction lengths for 1024 bits of data input

k	8	32	128	512
LUT	65536	1.8×10^{19}	1.15×10^{77}	1.79×10^{308}
Clock Cycle	16384	1024	64	4

4.3. Performing 32-Bit High-Speed LUT Multiplier with Vhdl

As seen in Figure 2, the LUT multiplier that will be created through VHDL has three main parts.

1. A 64-bit adder that has been created with the expansion of 1-bit adders,
2. ROM that consists of a look-up table,
3. A basic circuit that divides the input data into fractions reads the results of the multiplication of relevant fractions on the look-up table and sends them to the adder.

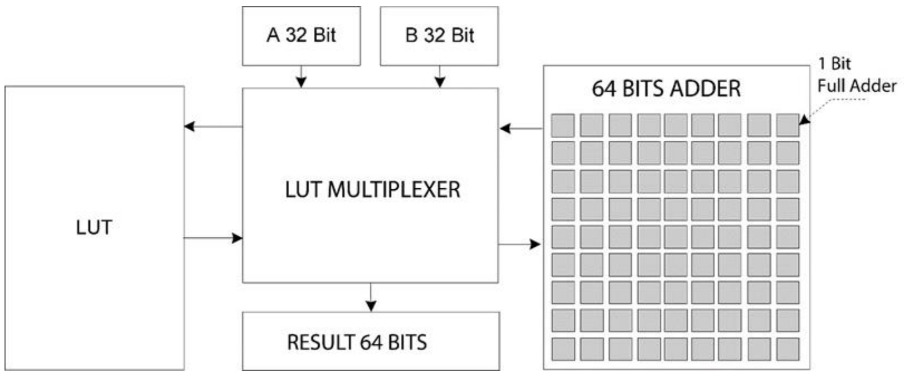


Figure 2 Basic block diagram of 32-bit multiplier created with VHDL.

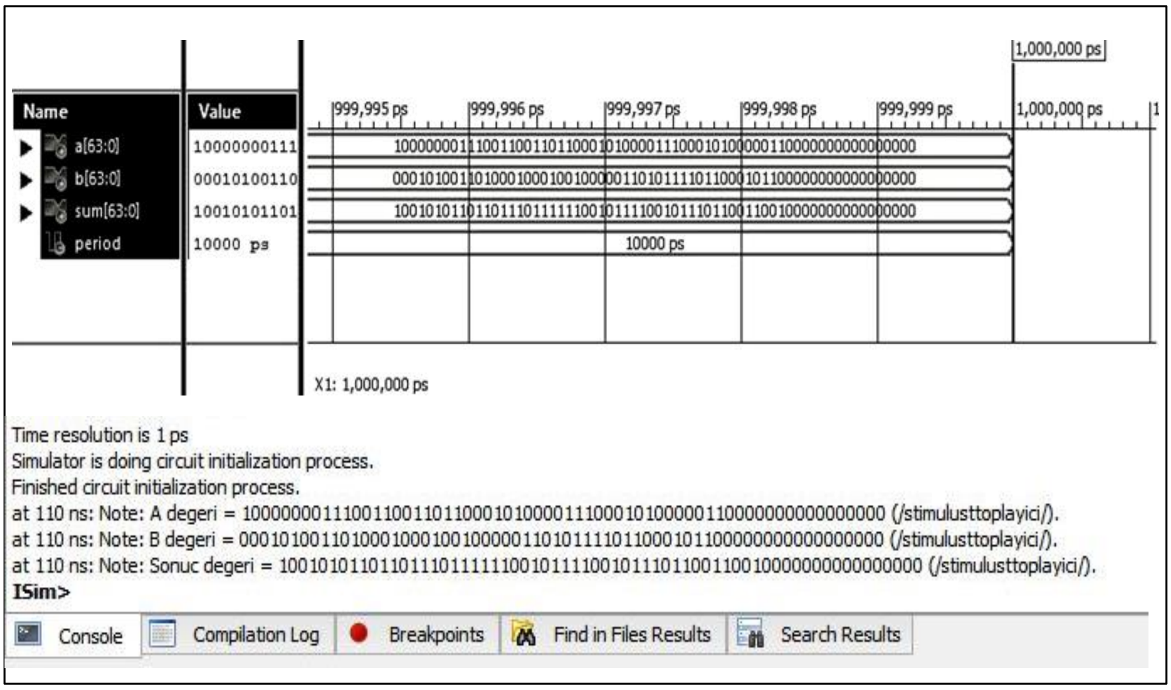
4.4. 64-Bit High-Speed Adder Design

Since multiple addition is the basis of multiplication, the adder's speed is critical in designing a multiplier. In a standard addition, the duration of the operation increases as the number of digits increases. In this multiplier, designed to multiply high-capacity numbers quickly, a standard adder cannot provide desired results in larger data sizes. To solve this problem that we encounter in adding large data, an adder was designed by properly expanding 1-bit full adders by the input data size.

The biggest advantage of this adder is that it can add input and output at a single clock stroke using 1-bit full adders connected in parallel. Another advantage of this circuit is that its size can be expanded to the desired capacity. While its capacity can be expanded to the desired size depending on the size of the FPGA hardware, the addition can be performed within the same period, i.e., within a single clock stroke.

Since the amount of data that can be processed within the same processing time can be expanded, the size of the multiplier can be increased to the desired level.

Figure 3 shows simulation results of a 1024-bit adder created with the same method. It is observed that the result is obtained within a single clock pulse. As long as the FPGA hardware has sufficient capacity, the input sizes can be increased to a desired amount without changing the time spent for addition. In this case, the sizes of designed multipliers can be expanded to desired sizes.



```
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
at 110 ns: Note: A degeri = 1000000011100110011011000101000011100010100000110000000000000000 (/stimulusttoplayici/).
at 110 ns: Note: B degeri = 0001010011010001000100100000110101111011000101100000000000000000 (/stimulusttoplayici/).
at 110 ns: Note: Sonuc degeri = 1001010110110111011111100101111001011101100110010000000000000000 (/stimulusttoplayici/).
ISim>
```

Figure 3 Results of simulation obtained through ISIM program of 64-bit adder.

4.5. Look-Up Table Design

The look-up table stores all potential results of previously calculated multiplier and multiplicand numbers. In this multiplier, this circuit takes up the most space on FPGA. Thus, the capacity of the FPGA hardware to be used when designing the look-up table should be considered.

If we want to design a 32-bit input-long LUT to multiply two 32-bit binary numbers as an integral, the LUT possibility number will be 264.

In compliance with the capacity of Xilinx Virtex 5 XC5VLX50T FPGA hardware, 32 bit-long inputs may be divided into k=8 bit-long fractions. In this case, the LUT possibility number will be 216.

In this case, the time equation required for obtaining the result of the multiplication can be found from Equation 9 as;

$$t = 2^{2[\log_2(32)-\log_2(8)]} = 2^{2[5-3]} = 2^4 = 16 \text{ steps.} \tag{9}$$

The entire 2^{2k} possibility that may be implemented on k bit-long inputs of the circuit has been tested in 2^{2k} of time. According to the results of the simulation, the random input values of the circuit showed that it gave results within a single clock pulse.

Since no physical multiplication was performed with the LUT circuit, the results can be obtained without additional addressing circuits according to the values applied to their inputs.

4.6. High-Speed LUT Multiplier Design

In a multiplier designed to multiply two 32-bit numbers, an 8-bit-input look-up table and a 64-bit adder that will add the values from the look-up table at each step were used.

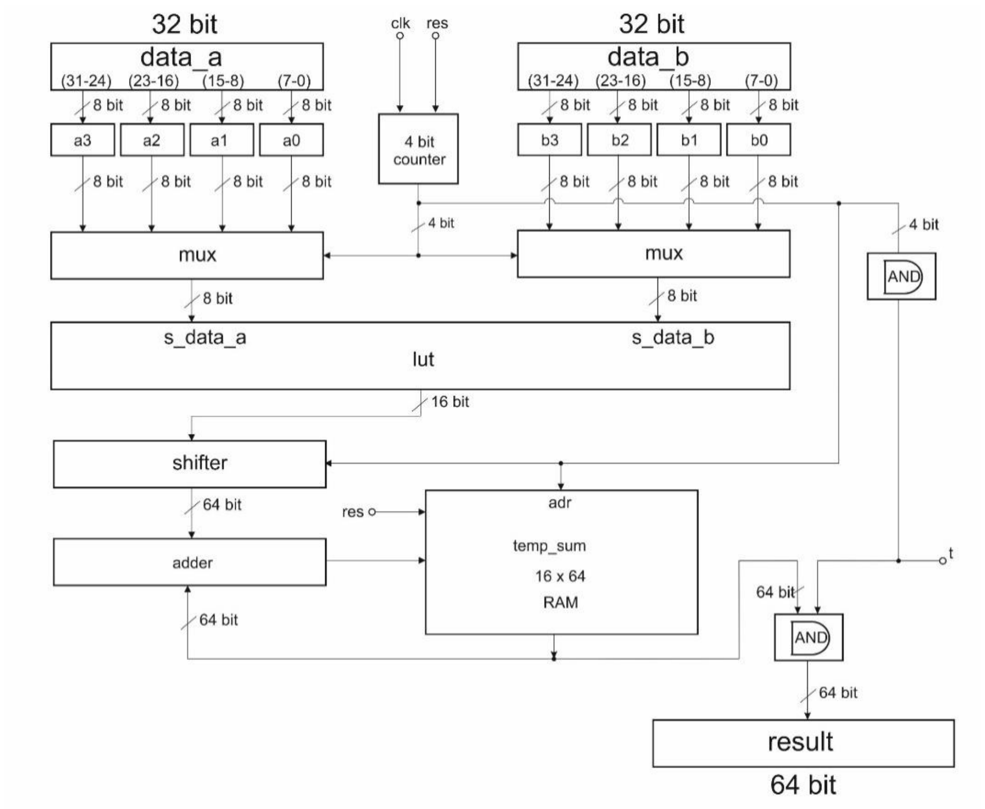


Figure 4 Block diagram of 32-bit LUT Multiplier

Algorithm 1 Expandable multiplication circuit structure with LUT

```
case c_state is
when sr =>
  result <=(others=>'0');
  t<='0';
  b_signal <= (others=>'0');
  a_signal <=  (others=>'0');
  s_data_a <= data_a(k  downto 0);
  s_data_b <=   data_b(k  downto 0);
  n_state  <=  S0;
when  Sta =>
  s_data_a<=data_a  [$k.t_a\left(mod\ \frac{n}{k}\right)+(k-1)$   downto   $k.t_a\left(mod\ \frac{n}{k}\right)$];
  s_data_b<=data_b  [$k.\left[t_a-t_a\left(mod\ \frac{n}{k}\right)\right]\left(mod\ \left(\frac{n}{k}-1\right)\right)+(k-1)$ downto $k.\left[t_a-t_a\left(mod\ \frac{n}{k}\right)\right]\left(mod\ (\frac{n}{k}-1)\right)$]  ;
  a_signal[ $k(i+j)+2k$  downto   $k(i+j)$ ] <= lut_result;
  b_signal <= total_signal;
  t='0';
  n_state<= Sta + 1;
when  Sta(max) =>
  result <= total_signal;
  t<='1';
  b_signal <= (others=>'0');
  a_signal <=  (others=>'0');
  s_data_a <=  (others=>'0');
  s_data_b <=  (others=>'0');
  n_state  <= sr;
when others =>
  b_signal <= (others=>'0');
  a_signal <=  (others=>'0');
  s_data_a <=  (others=>'0');
  s_data_b <=  (others=>'0');
  result   <=  (others=>'0');
  t<='0';
end case;
```

As Equation 9 states, two numbers with 32-bit input and 8-bit fraction data length are multiplied in 16 steps. A single processing time step passes at each clock pulse. Thus, the number of steps can be expressed with a 4-bit up counter that increases with every clock pulse. According to the number of steps from the counter, it is decided to implement which fractions of A and B data will be applied to the lut input and which bit sequence will be added.

The state timing of the designed circuit is shown in Figure 6. "c_state" refers to the current condition of the circuit. The signals referred to as "s_data_a" and "s_data_b" in Figure 6 are the inputs of the look-up circuit. Beginning from the "sr" state, the fractions of “data_a" and "data_b” that will be applied to the input of the look-up circuit in any case are shown. The signal connections referred to as "a_signal" and "b_signal" are the input connections of the adder. Beginning from the "s0" state, the output of the look-up circuit is transferred to the input of the "a_signal" adder. The "total_signal" is the output signal connection of the adder. Beginning from the "s0" state, the previous result of the adder at each state until "s16” determines the input value of "b_signal." Thus, the change in "b_signal" input occurs at "s1" state.

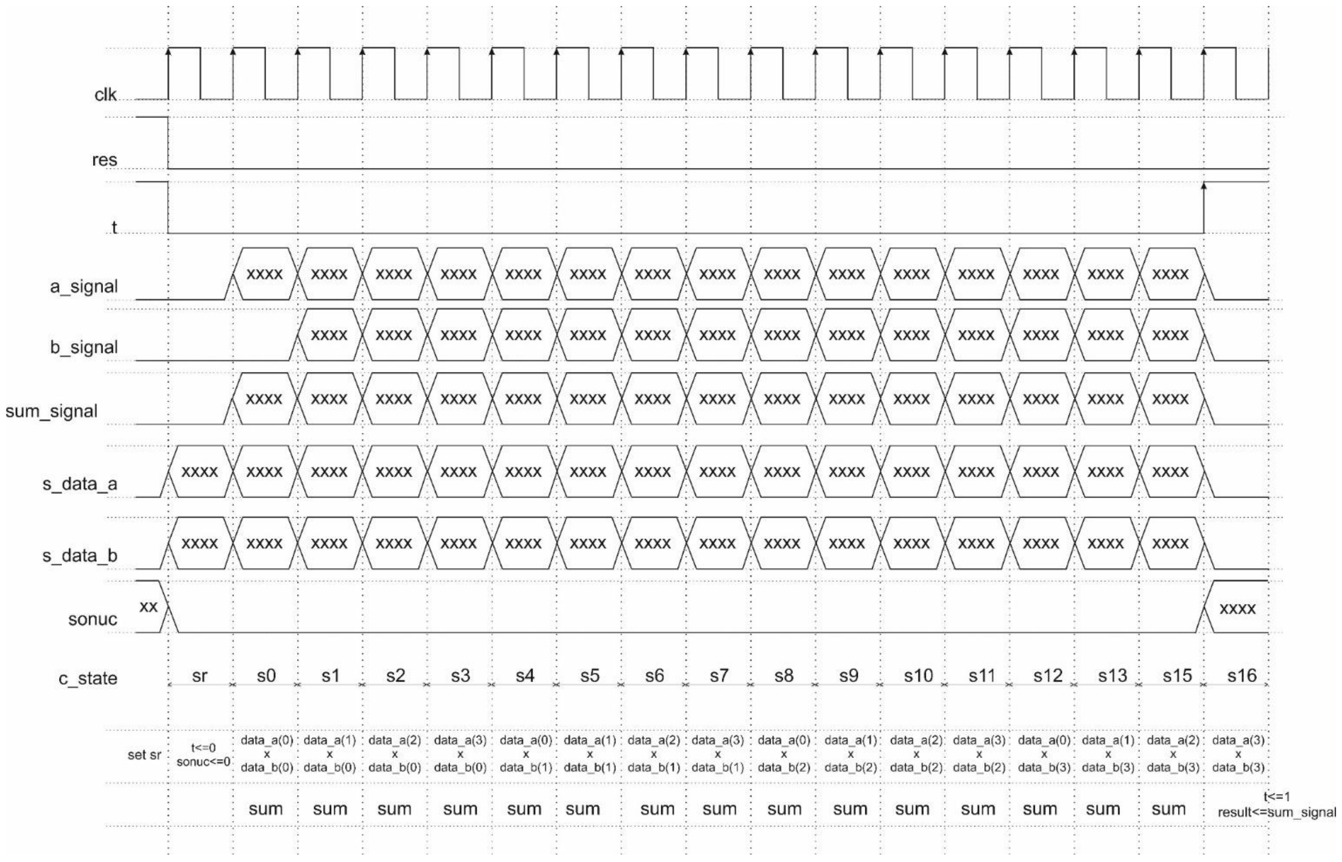
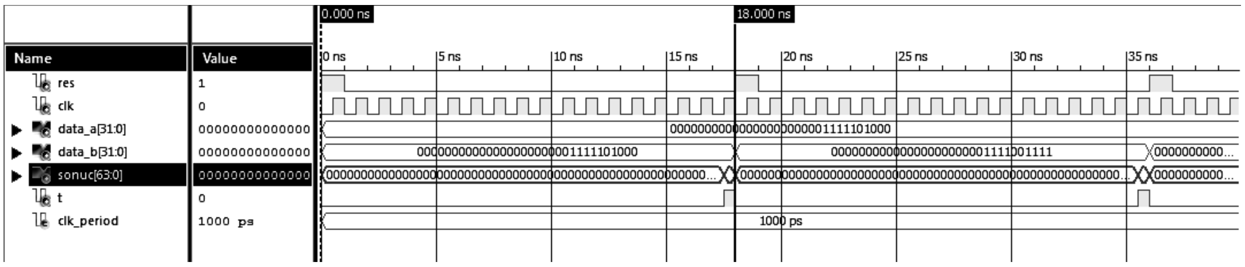


Figure 5 Input-output and signal connection states under current circumstances

4.7. FPGA Application of 32-Bit High-Speed LUT Multiplier

To try the structure of the circuit on different hardware structures during synthesizing and simulating multiplier and to obtain simulation results in different environments, Quartus II v9.0 - v14.1 Web Edition, ISE Design Suite v.14.7 and Vivado v2014.14 programs were used.

Random input values were assigned to the data_a and data_b inputs of the circuit, and the results were observed. Figure 6 shows that after the reset input becomes logic 0, the t output remains logic 0 for 16 cycles, and the results are obtained. In other cases, t output was logic 0, and all bits of resulting output were logic 0 zero.



Name	Value
res	1
clk	0
data_a[31:0]	00000000000000
data_b[31:0]	00000000000000
sonuc[63:0]	00000000000000
t	0
clk_period	1000 ps

```
data_a[31:0]   00000000000000000000001111101000
data_b[31:0]   00000000000000000000001111101000   00000000000000000000000111001111   0000000000...
sonuc[63:0]    000000000000000000000000000000000000000000000000000000000000...   000000000000000000000000000000000000000000000000000000000000000...   0000000000...
clk_period     1000 ps
```

Figure 6 Results of simulation obtained through ISIM program of 32-bit LUT multiplier.

The FPGA chip and the location of the circuit on the FPGA chip determine the circuit's performance at high speeds. When the transistors within FPGA delay data transmission, this has a negative impact on the performance of the circuit at high speeds. Accordingly, the maximum frequency of the circuit is determined by the longest distance the data between input and output will follow. The register was attached to the input and outputs of the circuit when analyzing timing. The maximum frequency was determined based on the time it takes for data to travel between two registers.

Table 2 shows synthesizing processes performed on various FPGA chips with ISE, Quartus and Vivado software and the maximum frequency values.

Table 2 The timing analysis chart of the circuit between two recorders

Program	ISE 14.7			Quartus II 14.1			Vivado v2014.4	
FPGA	Virtex 5	Virtex 6	Kintex 7	Cyclone IV	Cyclone V	Cyclone V	Kintex 7	Artix 7
	xc5vlx50t-2ff1136	xc6vlx75t-2ff784	xc7k70t-fbg676	ep4cgx150df31I7ad	5cgxfc7d7f27c8	5cgxfc7d7f31c8	xc7k160tffg676-2l	xc7a200tffg1156-3
RR (ns)	1.498	1.171	0.995	1.975	2.763	1.640	1.409	1.592
Fmax (Mhz)	667.557	853.97	1005.03	506.33	361.93	609.76	709.72	628.14

5. Discussion and Conclusion

In this study, a high-speed multiplier was designed by using look-up tables. The look-up circuit stored all previously calculated multiplication results for two 8-bit numbers, and the logic circuit that multiplied two 32-bit long numbers was synthesized on Virtex 5 xc5vlx50t FPGA hardware. In the results of the simulation performed on ModelSIM and ISIM, it was observed that the multiplication of two 32-bit numbers gave results in 16 cycles through multiplication by division into partial fractions. Since the high-speed expandable adder, designed to add partial results, could add the partial results in a single cycle, it allows the circuit to perform arithmetic operations quickly. The maximum frequency value of the circuit was calculated as 667.557 Mhz in time analyses performed via the ISE program.

This circuit can be used as a sub-unit of an arithmetic unit or as a circuit sub-unit in encoding applications. It can provide high-speed processing for larger capacities in real-time signal processing. In addition, since the circuit uses look-up tables, it consumes less power as it does not perform physical multiplication.

Table 3 Comparison of Time Complexity

	Time Complexity	Number of steps for two 32-bit numbers
Standard Multiplication	O(n^2)	1024
Karatsuba	O($n^{1,585}$)	243
Shift/Add	O(n)	32
High-Speed LUT	O($2^{2(log_2(n)-log_2(k))}$)	16

Although the algorithm built for disintegration numbers in this circuit is similar to the Karatsuba algorithm, they differ in the multiplication of fractions and addition of partial results.

Depending on the unit where the multiplier will be used, LUT fractions can be expanded, and a look-up table can be recreated again. However, the biggest problem of the circuit here is the size of LUT. As FPGA capacity continues to grow in the coming years, it will be possible to perform multiplications in shorter time frames by creating larger lookup tables.

This study presents a new, innovative approach to high-speed multiplication using look-up tables and FPGA hardware. This approach's numerous benefits include increased speed, power efficiency, versatility, scalability, and algorithmic innovation. These benefits can have a significant impact on various applications in the field of digital design and arithmetic units.

References

[1] R. W. Keyes., "Physical Limits of Silicon Transistors and Circuits", *Reports on Progress in Physics*, vol. 68, no. 12, 2005, doi: 10.1088/0034-4885/68/12/R01

[2] B. Parhami, *Computer Arithmetic Algorithms and Hardware Designs Secon Edition*, Oxford University Press, New York USA, 2010, ISBN 978-0-19-532848-6

[3] A. A. H. Abd-Elkader, M. Rashdan, E. A. M. Hasaneen and H. F. A. Hamed, "Efficient implementation of Montgomery modular multiplier on FPGA," *Computers and Electrical Engineering*, vol. 97, 2022, doi: https://doi.org/10.1016/j.compeleceng.2021.107585

[4] A. Behl, A. Gokhale, N. Sharma, "Design and Implementation of Fast Booth-2 Multiplier on Artix FPGA", *Procedia*

Computer Science, vol. 173, pp. 140-148, 2020, doi: https://doi.org/10.1016/j.procs.2020.06.018
[5] R. K. Sakali, S. Veeramachaneni, N. M. Sk, "Preferential fault-tolerance multiplier design to mitigate soft errors in FPGAs", *Integration*, vol. 93, 2023, doi: https://doi.org/10.1016/j.vlsi.2023.102068
[6] L. Malathi, A. Bharathi, A.N. Jayanthi, "FPGA design of FFT based intelligent accelerator with optimized Wallace tree multiplier for image super resolution and quality enhancement", *Biomedical Signal Processing and Control*, vol. 88, part B, 2024, doi: https://doi.org/10.1016/j.bspc.2023.105599
[7] V. Bianchi, I. D. Munari, "A modular Vedic multiplier architecture for model-based design and deployment on FPGA platforms", *Microprocessors and Microsystems*, vol. 76, 2020, doi: https://doi.org/10.1016/j.micpro.2020.103106
[8] E. Özcan, S. S. Erdem, "A fast digit based Montgomery multiplier designed for FPGAs with DSP resources", *Microprocessors and Microsystems*, vol 62, pp. 12-19, 2018, doi: https://doi.org/10.1016/j.micpro.2018.06.015
[9] M. Morales-Sandoval, C. Feregrino-Uribe, P. Kitsos, R. Cumplido, "Area/performance trade-off analysis of an FPGA digit-serial GF(2m) Montgomery multiplier based on LFSR", *Computers & Electrical Engineering*, vol. 32, i. 2, pp. 542-549, 2013, doi: https://doi.org/10.1016/j.compeleceng.2012.08.010
[10] R. S. Özbey and A. Sertbaş, "Klasik Çarpma Algoritmalarının Donanımsal Simülasyonları ve Performans Değerlendirimi", *Inter. Conf. on Electrical and Electronics Engineering (ELECO 2004)*, pp. 303-308, 2004
[11] A. D. Booth, "A Signed Binary Multiplication Technique", The Quarterly Journal of Mechanics and Applied Mathematics. *Math. Oxford University Press*, vol. 4, no. 2, pp. 236-240, 1951, doi: https://doi.org/10.1093/qjmam/4.2.236
[12] C. S. Wallace, "A Suggestion for a Fast Multiplier", *IEEE Transactions on Electronic Computers*, vol. 13, no. 1, pp. 14-17, 1964, doi: 10.1109/PGEC.1964.263830
[13] M. R. Kumar and G. P. Rao, " Design and Implementation Of 32 Bit High Level Wallace Tree Multiplier", *International Journal of Technical Research and Applications*, vol. 1, no. 4, pp. 86 - 90, 2013, Accessed : 29 October 2023 [Online]. Available: https://api.semanticscholar.org/CorpusID:13022315
[14] J. Kulisz, J. Mikucki, "An IP-Core Generator for Circuits Performing Arithmetic Multiplication", *IFAC Proceedings Volumes*, vol. 46, i. 28, 2013, doi: https://doi.org/10.3182/20130925-3-CZ-3023.00006
[15] A. J. Al-Khalili, *Digital Design and Synthesis Lecture Notes (2019)*, Accessed : 29 October 2023 [Online]. Available: https://users.encs.concordia.ca/~asim/COEN_6501/elec650.html
[16] S. Mishra and M. Pradhan, "Implementation of Karatsuba Algorithm Using Polynomial Multiplication", *Indian Journal of Computer Science and Engineering*, ISSN: 0976-5166, vol. 3, no. 1, pp 88 - 93, 2012.
[17] R. T. Kneusel, *Numbers and Computers*, Springer, USA, pp. 136, 2015, ISBN: 978-3-319-17260-6

Conflict of Interest Notice
The authors declare that there is no conflict of interest regarding the publication of this paper.

Ethical Approval and Informed Consent
It is declared that during the preparation process of this study, scientific and ethical principles were followed, and all the studies benefited from are stated in the bibliography.

Availability of data and material
Not applicable

Plagiarism Statement
This article has been scanned by iThenticate ™.