STM JOURNALS

Journal of
VLSI Design Tools & Technology

ISSN: 2249-474X (Online)
ISSN: 2321-6492 (Print)
Volume 14, Issue 3, 2024
September–December
DOI (Journal): 10.37591/JoVDTT

https://journals.stmjournals.com/jovdtt

Research JoVDTT

Design and Implementation of Low Latency 16-bit Full Adder

Sandeep N. Uttarkar[1,*], K.B. Ramesh[2]

Abstract

In the new age world, modern technology has vast applications in various fields of engineering, medicine, and others. Digital electronics processing and systems play a vital role in the analysis and computation of data representation. Electronic systems have gradually become faster and smaller scale. The primary structural element of any modern ALU-based processor is an adder. Addition is a well-known extremely basic function that is employed in nearly all computing processes. The performance of the adder circuit will therefore greatly affect the performance of the CPU. In terms of frequency, general adders like half adders, full adders, ripple-carry adders, carry skip adders, and carry look-ahead adders don't meet the demands of high-performance processors. To fulfill the required specifications, we suggest in this work a high-frequency 16-bit full-adder architecture. This architecture was simulated using Verilog on the Xilinx ISE 14.7 tool, and it was then built on the field-programmable gate array (FPGA) families.

Keywords: Half adder (HA), full adder (FA), carry-select adder (CSA), carry look-ahead adder (CLA), input-output blocks (IOB), configurable logic blocks (CLB), field-programmable gate array (FPGA)

INTRODUCTION

Electronics are a vital component of modern life, appearing in many of the devices we use daily, including computers, cameras, cell phones, washing machines, refrigerators, air conditioners, fans, calculators, fax machines, and projectors. The smooth functioning of these gadgets depends on the effectiveness and capability of processors, which act as the technological brains behind the various features that have become essential to our contemporary way of life. The processor is an essential part of any computing system responsible for effectively processing data [1]. The processor, which consists of essential components including the control, memory, and arithmetic logic units (ALU), is responsible for coordinating the complex dance of information transformation. Of these essential components, the ALU stands out as the processor's beating heart, the engine that rapidly and precisely carries out instructions. These three parts work together in unison, representing the intellect and strength required for the smooth completion of various computational tasks. A processor is essentially a complex digital circuit that is painstakingly built to perform a variety of arithmetic and logical operations [2]. As such, it is a computational engine for digital systems. Its repertoire encompasses basic mathematical operations that form the basis of numerical manipulation, including addition, subtraction, multiplication, division, and percentile calculations. Meanwhile, in the domain of logical operations, the CPU performs logical AND, OR, XOR, XNOR, NOT, logical NAND, and NOR operations with ease. Addition can be performed on a number of different numerical bases, such as binary, decimal, occlusal, and hexadecimal. Among these, binary is frequently thought to be the easiest

***Author for Correspondence**
Sandeep N. Uttarkar
E-mail: sandeepnu.ei22@rvce.edu.in

[1]Student, Department of Electronics and Instrumentation Engineering, R.V. College of Engineering (An Autonomous Institution, affiliated to VTU Belagavi), Banglore, Karnataka, India.
[2]Associate Professor, Department of Electronics and Instrumentation Engineering, R.V. College of Engineering (An Autonomous Institution, affiliated to VTU Belagavi), Banglore, Karnataka, India.

Received Date: August 13, 2024
Accepted Date: August 21, 2024
Published Date: September 10, 2024

Citation: Sandeep N. Uttarkar, K.B. Ramesh. Design and Implementation of Low Latency 16-bit Full Adder. Journal of VLSI Design Tools & Technology. 2024; 14(3): 21–31p.

to use. This adaptability is used in specialized fields, such as the Fibonacci number sequence, in addition to traditional numerical methods. The fact that addition is so common emphasizes its importance as a basic activity that goes beyond numerical representations. An essential element of contemporary computing—the adder circuit—is crucial for the development of various complex systems. The adder is essential for performing intricate computations in many numerical systems, as well as for adding binary digits.

Half Adder

The half adder is the fundamental block that performs logical addition between two binary bits. The block diagram and truth table are presented in Figure 1 and Table 1, respectively. In Figure 1, we show the block diagram of the half adder and its truth table, as shown in Table 1. We can add only two single binary bits using a half adder; hence, we use a full adder, which has three inputs to perform addition.

Full Adder

The half adder is expanded by the full adder. To accomplish logical addition between multiple-bit inputs, we add an extra bit to the full adder [3]. We add another input to the half adder because logical addition between two bits may result in carry, which serves as the third input to the input of the subsequent full adder. The block diagram and truth table for the full adder are shown in Figure 2 and Table 2, respectively. If the input binary number has multiple bits, we perform an addition by cascading a set of full adders.

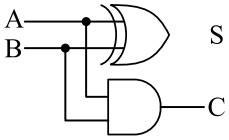

Figure 1. Block diagram of the half adder.

Table 1. Truth table of the half adder.

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

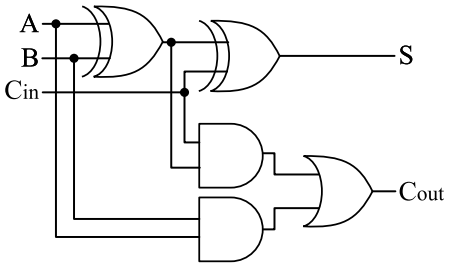

Figure 2. Block diagram of the full adder.

Table 2. Truth Table of the full adder.

A	B	Ci	S	Co
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

16-Bit Full Adder

By linking the carry input of one full-adder block to the preceding carry-out, we can create a 16-bit full adder by cascading 16 full-adder blocks. The block diagram of a 16-bit full adder [4] is shown in Figure 3. S0 is the least significant bit of the resultant, S15 is the most significant bit, and one and carry are the most significant bits [5]. In Figure 3, A0 and B0 represent the least significant bits, and A15 and B15 represent the most significant bits.

A pipeline with an array of half-adders performing carry-save addition at each stage is a well-known method for creating high-throughput adders. Pipelined circuits [6] of size n k-bit O(k(1ogn+logk)), latency O (1ogn+ logk +d), and feedback loop with a single full-adder delay can be used to find the sum of n k-bit integers. The value of d was a positive integer. The traditional pipelined ripple-carry adder, a variant that uses fewer registers, and an FPGA-specific carry-select adder (CSA) can offer faster additions at a similar cost.

Contributions

This paper contributes to the following aspects:

1. Listing of the Advantages of the 16-bit full adder and its drawbacks.
2. Improving the current 16-bit full-adder frequency by pipelining the bits using a CSA.
3. Depict the graphs of different FPGA boards comparing their frequency.
4. Methods to enhance the frequency further for 32-bit adder and above.

This study had five main aspects. In Section I, we present details about half-adders and full adders. A typical 16-bit full adder was implemented using several full adders. In Section II, the advantages and drawbacks of the current 16-bit adder are assessed. In Section III, we examine the work of the carry-select adder (CSA) and the methodology to improve the architecture. In Section IV, we discussed the pipelining technique. In Section V, graphs of the different FPGA boards comparing frequencies are presented. In Section VI, we propose a model to further enhance higher-bit adders. Section VII summarizes our study.

CURRENT TECHNOLOGY OF 16-BIT ADDER

A circuit that can effectively execute arithmetic operations on 16-bit binary values is known as the 16-bit full adder. This circuit is essential for several computational tasks. A noteworthy development in 16-bit full adders is the drive to increase efficiency and performance. Engineers and researchers are constantly developing new designs and optimization strategies to improve the speed and power efficiency of these adders. This involves investigating cutting-edge semiconductor technologies to enhance the overall functionality of digital circuits such as FinFETs or beyond. Furthermore, 16-bit complete adders are typically incorporated into application-specific integrated circuits (ASICs) and field-programmable gate arrays (FPGAs).

The specific frequency of current high-speed 16-bit full adders can vary depending on several factors, such as the technology node, design techniques, and optimization strategies employed. However, in advanced semiconductor technologies, such as those around 7 nm or below, high-speed 16-bit full adders can typically operate in the range of several gigahertz (GHz) [7]. To provide a rough estimate,

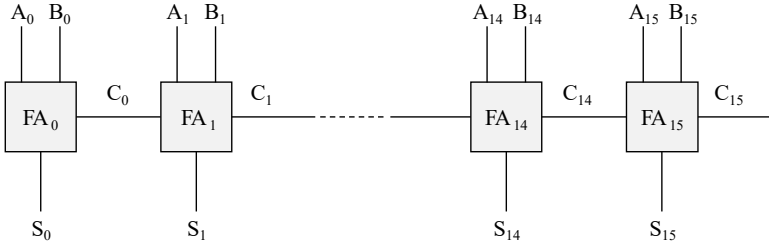

Figure 3. 16-bit full adder.

these high-speed 16-bit full adders might operate anywhere from *around 2 GHz to 10 GHz* or even higher in some cases, depending on the specific implementation and optimization. It is important to note that these frequencies are approximate and can vary according to specific design goals, constraints, and targeted applications.

Advantages of a 16-bit Full Adder

1. A 16-bit full adder with a low frequency can be useful in situations where power efficiency can be maximized.
2. A 16-bit full-adder operating at low frequency can help mitigate thermal issues.
3. A 16-bit full adder often strikes a balance between precision and speed. The carrying propagation delay can be managed.
4. A 16-bit full adder facilitates parallel processing, enabling simultaneous computation of multiple 16-bit operations. This can lead to improved computational efficiency.
5. A 16-bit full adder is the fundamental building block. This forms the basis of executing arithmetic and logic operations.

Drawbacks of 16-bit Full Adder

It is important to note that the advantages and disadvantages of a 16-bit full-adder or any specific design choice depend significantly on the application requirements.

1. A 16-bit full adder inherently involves a more complex circuit design than smaller adders.
2. The carry propagation delay in a 16-bit full adder is longer than that in the smaller adders. This delay limited the operating speed.
3. The primary disadvantage of a 16-bit full-adder operating at a low frequency is its decreased computational speed.
4. A 16-bit full adder with low frequency may not be well-suited for high-performance computing tasks demanding applications.
5. The physical space required to implement a 16-bit full adder is greater than that required for smaller adders.

Enhancing Pipelining Architecture

To enhance the frequency of a 16-bit full adder, pipelining can be strategically employed to introduce parallelism into the computational process. In a pipelined architecture, 16-bit addition is divided into sequential stages, with each stage responsible for processing a portion of the overall computation. Registers are strategically placed between these stages to store intermediate results and facilitate a smooth flow of data through the pipeline. This parallelized approach allows the different stages of the adder to operate concurrently, significantly reducing the critical path delay and enabling a higher clock frequency. By breaking down the complex 16-bit addition into manageable stages, pipelining ensures that computations are executed more streamlined and efficiently.

The pipelined structure minimizes the impact of the carry propagation delay by allowing the carry output from one stage to be immediately fed into the next stage. As a result, the 16-bit full adder can achieve enhanced frequency performance, making it well-suited for applications that require rapid and efficient arithmetic operations.

PROPOSED METHODOLOGY

Registers and a 4-bit CSA were used to achieve this design. Many data-processing processors use carry adders to perform quick arithmetic operations. It typically consists of a multiplexer and a ripple-carry adder. It generates quick wide adders by combining several narrow adders [8]. Different blocks with either a constant or variable block size are compromised by the n-bit carry choice adder. The central concept is to optimize addition by pre-computing both the sum and carry for each block, considering two scenarios: one with a carry-in equal to zero and the other with a carry-in equal to one. This approach requires the utilization of two adders for each block, except for the initial one. Once both results are computed, a multiplexer is employed to select the correct sum and carry outputs based on

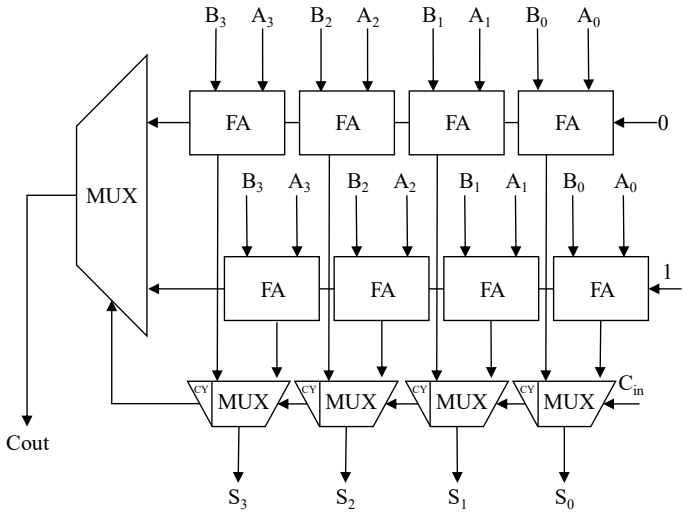

Figure 4. Carry-select adder.

the known correct carry. This technique expedites addition and prioritizes speed over area consumption. Although this design incurs the cost of additional hardware, it proves valuable in applications where swift computation is of paramount importance. This strategic use of redundancy and parallel computation optimizes the speed of the adder, catering to scenarios where rapid arithmetic operations take precedence over conserving hardware resources.

The CSA was implemented using eight full adders and five multiplexers, as shown in Figure 4. Among the eight full adders, we use full adders for one set of inputs, and the remaining four are used for another set of inputs. The outputs from these two sets of full adders are provided to the respective multiplexers. The input for the 5th multiplexer is taken from the carry outputs of the MSB for each full adder set. A block diagram of the CSA is presented above. Now, using CSA and register, we are going to implement a pipelined 16-bit full adder.

Methodology to Enhance Frequency

In Figure 5, the red box represents the CSA, the green box represents registers, and the first four LSB are assigned to the leftmost CSA. The remaining consecutive bits were assigned to the registers. The first four bits are allocated to the CSA carry after all inputs have been assigned to the nodes, and the remaining bits are assigned to registers in the clock cycle created by the first CSA. The second-row first-column register receives the sum of the first LSB, and the second-row and second-column registers receive the carry. The output of this register is assigned to the second LSB CSA carry input, and the outputs of the first two registers are supplied to the second LSB CSA. The following registers receive output from the remaining registers.

Next, the third-row second-column register receives the total of the second LSB CSA, and the third-row third-column register receives the carry-out. Third-row third-column registers are now used for the third LSB CSA sum input, whereas the third-row third-column registers are used for the carry input. The remaining bits are assigned to each of the ensuing registers. The final step involves gathering the total from each register, calculating the most significant CSA, and extracting the carry output from the most significant CSA. The input for the most significant CSA was obtained from the fourth-row, third-column, and register.

Verification Environment and Architecture

Based on HVL, or System Verilog, the verification environment is utilized for Constraint Random Verification with Coverage Driven. The first stage in properly identifying what needs to be tested is verification. The improvement and end of the verification phase are then determined by the information

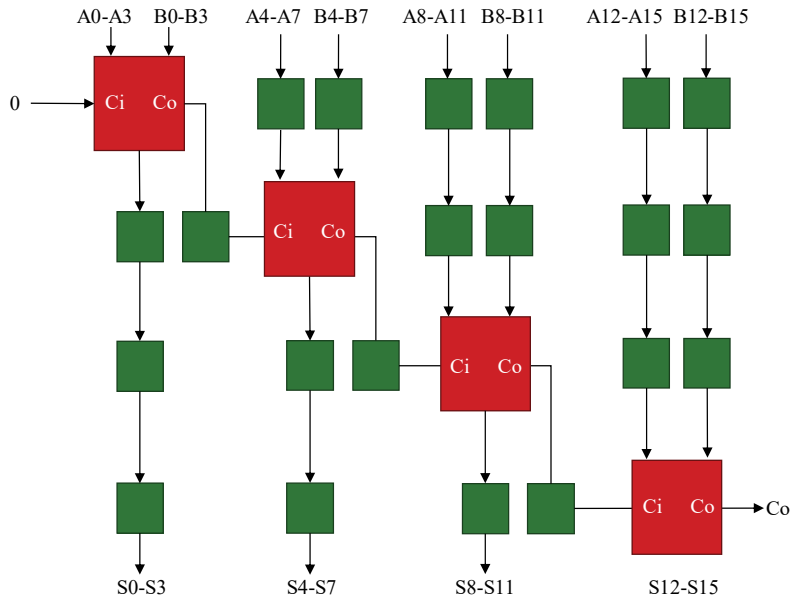

Figure 5. CSA with registers—improved model.

gathered. In this study, the verification environment was enhanced, and the functioning of the design was confirmed in every scenario. The following points are considered project specific.

1. Reuse of the Code.
2. Stimulus generation.
3. To build verification on the next phase.
4. DUT verification inside the environment.

System Verilog Verification Architecture

1. *Generator:* This sends the stimulus, which is created by selecting the input values at random to the driver via the mailbox.
2. *Driver:* After obtaining randomly produced values from the generator, it changes the signals from the packet to the pin level.
3. *Monitor:* The monitor receives the output of the DUT and converts it from pin-to-packet-level data. Subsequently, it was sent to the scoreboard via the mailbox.
4. *Scoreboard:* It gets packets from the monitor and generator and compares them.

The transaction is initiated by a generator that randomly selects the inputs and then sends them to the driver through the mailbox. Packets were created by the driver and sent to the DUT interface. Once the packet is placed in the mailbox, the receiver collects data bytes from the signal interface. These bytes were then removed from the corresponding packets and returned to the mailbox. One mailbox on the scoreboard receives packets from the monitor and the other mailbox receives packets from the generator. After comparing the packets received, an error assertion is raised if there are differences. The transaction is started by a generator that first randomizes the inputs before sending them to the driver through the mailbox. Packets were created by the driver and sent to the DUT interface. Once the packet is inserted into the mailbox, the receiver obtains the data bytes from the signal interface. After removal from their packets, these bytes are sent back to the mailbox. Two mailboxes, one from the monitor and one from the generator, are incorporated into the scoreboard, and they both receive packets. These received packets were compared, and an error assertion was generated if differences were found.

PIPELINE TECHNIQUE

In general, improving a design's performance entails either improving the design directly or strategically rearranging it to increase efficiency. The development of pipelining techniques provides a way to

increase performance with little change to the hardware, even when enhancements to the overall architecture may require additional hardware. Pipelining is the process of arranging the hardware components of a CPU to maximize efficiency. This method allows many instructions to be executed simultaneously at different stages, which significantly improves performance without requiring any changes to the design. By adding targeted registers at practical nodes, pipelining can be implemented to ensure that the functionality is maintained while the design speed is increased.

To simplify this idea, let us look at a function that performs its purpose in three steps. In the conventional method, the two stages of the design remain inactive, whereas one is active following data input, resulting in an inefficient design. Pipelining techniques allow for the simultaneous processing of various inputs in all three stages. This simultaneous operation guarantees that every step actively contributes to the task, improving the overall performance of the design. When designing a digital circuit, especially a complex circuit such as a 16-bit full adder, the critical path [9]—the longest path through the logic from input to output—determines the maximum clock frequency at which the circuit can reliably operate. By breaking down the computation into smaller segments and processing them sequentially, the critical path delay can be reduced, thereby increasing the clock frequency that the circuit can support. This technique is known as pipelining. Here's how it works:

1. *Segmentation:* The computation was divided into smaller stages, each performing a portion of the overall operation. In the context of a full adder, each stage can handle a subset of bits being added.
2. *Registers:* Registers are inserted between these stages to temporarily store intermediate results. These registers introduce a delay, but also isolate each stage from the others, allowing them to operate concurrently.
3. *Sequential processing:* Each stage operates independently on its assigned portion of the data, with data passing from one stage to the next through registers.

By pipelining the computation, the critical path was divided into smaller segments, each shorter than the original critical path. Consequently, the overall critical path delay was reduced, allowing for higher clock frequencies. However, pipelining also introduces latency, which is the time it takes for the result to appear at the output after the input is applied. This latency is equal to the sum of the delays introduced by each pipeline stage and register. In addition, pipelining can increase the complexity of the design and may require additional hardware resources.

Without Pipeline

In Table 3, for every input, three consecutive clock cycles are required to complete the process and produce the output. After producing the output of the first input then the 2nd input is given, and the process repeats. Therefore, to obtain the output for the three inputs, nine clock cycles are required. The time taken to get output for the n number of inputs is defined as

$$T(\text{non-pipeline}) = n \times k \times Tp. \quad (1)$$

Where,
n = number of inputs,
k = number of stages,
Tp = time taken for a single clock cycle.

With Pipeline

Assume a 16-bit full-adder design that operates at a frequency of 1 GHz (1 billion cycles per second) without pipelining. Our goal was to increase its speed by one percent using pipelining, as shown in Table 4.

Table 3. Without pipeline.

Stage/cycle	1	2	3	4	5	6	7	8	9
1	1			2			3		
2		1			2			3	
3			1			2			3

Table 4. With pipeline.

Stage/cycle	1	2	3	4	5
1	1	2	3		
2		1	2	3	
3			1	2	3

First, we must understand the current critical path delay of the design. The critical path delay without pipelining is 100 picoseconds (ps) per stage. Because the full adder is 16 bits wide, 16 stages are required to complete the computation without pipelining.

Without pipelining:

1. Number of stages: 16
2. Critical path delay per stage: 100 ps
3. Total critical path delay: 16 stages * 100 ps/stage = 1600 ps

Now, we introduce pipelining to reduce the critical path delay. We add pipeline registers between each stage, breaking down the computation into smaller segments. For simplicity, we assume that a pipeline register is introduced after each stage.

With pipelining, each stage operates concurrently, and the critical path delay is reduced to the delay of a single stage. However, we also introduce pipeline latency owing to the registers. Assuming that the pipeline registers [10] add a delay of 10 ps each, the total critical path delay with pipelining is:

1. Critical path delay per stage: 100 ps (unchanged)
2. Pipeline register delay: 10 ps
3. Total critical path delay: 100 ps (stage delay) + 10 ps (register delay) = 110 ps.

Now, the total critical path delay with pipelining becomes:

16 stages * (100 ps + 10 ps) = 1760 ps. Compared to the original critical path delay without pipelining (1600 ps), the critical path delay with pipelining (1760 ps) was higher owing to the added pipeline latency. However, pipelining allows for higher clock frequencies, because it reduces the effective critical path delay. To maintain the same throughput (number of operations per second), we can increase the clock frequency in proportion to the increase in the critical path delay.

To calculate the new clock frequency, we use the formula:

$$\text{New Frequency} = \text{Old Frequency} / \text{New Critical Path Delay}$$

Substituting the values:

$$\text{New Frequency} = 1\ \text{GHz} / 1760\ \text{ps} = 568.18\ \text{MHz}$$

Thus, with pipelining, the new clock frequency would be approximately 568.18 MHz. This represents an increase in speed over the original 1 GHz clock frequency. The percentage increase in clock frequency can be calculated as

$$\text{Percentage Increase} = (\text{New frequency} - \text{Old frequency} / \text{Old frequency}) \times 100\%$$

$$\text{Percentage Increase} = (568.18\ \text{MHz} - 1\ \text{GHz} / 1\ \text{GHz}) \times 100\%$$

$$\approx -4.182\% \text{percentage increase}$$

However, the calculated percentage increase was negative because we reduced the clock frequency to accommodate the added pipeline latency. To achieve a one percent increase in speed, we would need to adjust the pipeline stages and their associated latencies accordingly while ensuring that the timing

constraints are met. This might involve optimizing the pipeline structure or reducing the delay introduced by pipeline registers.

RESULTS AND COMPARISON

Environment testing of the synthesized RTL code was carried out using the System Verilog (SV). The functional coverage port was achieved Using Questa Sim 10.4 e. FPGA Boards Parameters of 16-bit CSA is shown in Table 5.

The implemented architecture was synthesized using Xilinx. Using the ISim simulator and the Integrated Synthesis Environment (ISE) design suite 14.6. Many families exist within each FPGA board, including Spartan, Virtex, Kintex, and Artix. Each family has a different board with different speeds and bundles. Here, we compare the parameters using FPGA boards, such as Virtex-7 (xc7vx330t-2ffg1157), Spartan-6 (xc6slx4-2tqg144), and Virtex-5 (xc5vlx20t-2ff323). Spartan and Virtex are the families that we compare here.

A board is represented by xc7vx330t, with a speed of 2; packages of the Virtex 7 family are represented by ffg1157; packages of the Spartan-6 family are represented by tqg144; packages of the Virtex 5 family are represented by xc5vlx20t, with a board and a speed of 2ff323.

Each family has a certain number of slices, look-up tables (LUTs), and flip-flops configured in configurable logic blocks (CLBs). The CLBs for all devices were assumed to be similar, which may lead to closed hardware implementation outcomes. In addition, resources are found in terms of LUTs, flip-flops, and slices specified in all FPGA devices with different configurations.

In Figure 6, it appears that all adders consumed less and the same number of SUTs. It is clear from Figure 7 that the frequency of operation of the proposed architecture is significantly higher than that of the other two and reasonably higher. Figure 8 lets us know that Spartan-6 consumes seventy-one flip-flops, twenty-one LUTs, and fifty-two input-output blocks.

The pipelined 16-bit full adder was created in Verilog and required the Xilinx ISE Design Suite for correct operation. The charts provide a detailed representation of the design parameters used in the RTL implementation.

Table 5. FPGA board parameters of 16-bit CSA.

Parameter/board	Spartan-6	Virtex-5	Virtex-7
No. of IOB	24	28	26
No. of IOB	52	50	50
Max. Freq. (MHz)	62.492	129.634	202.142

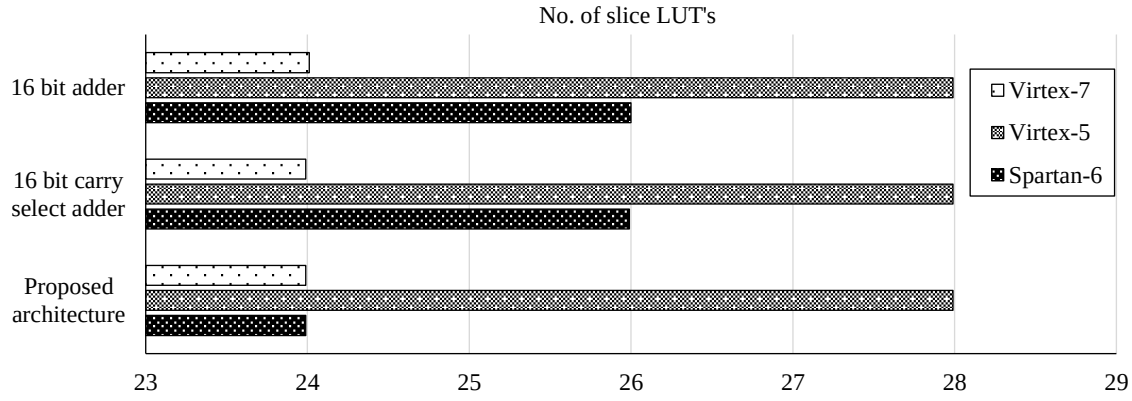

Figure 6. Graph showing no of slice LUTs.

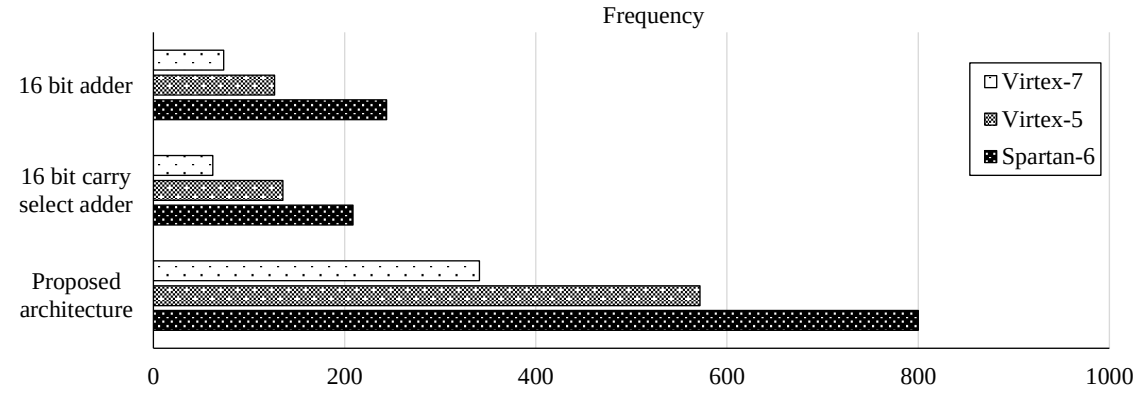

Figure 7. Graph showing the frequency of operation.

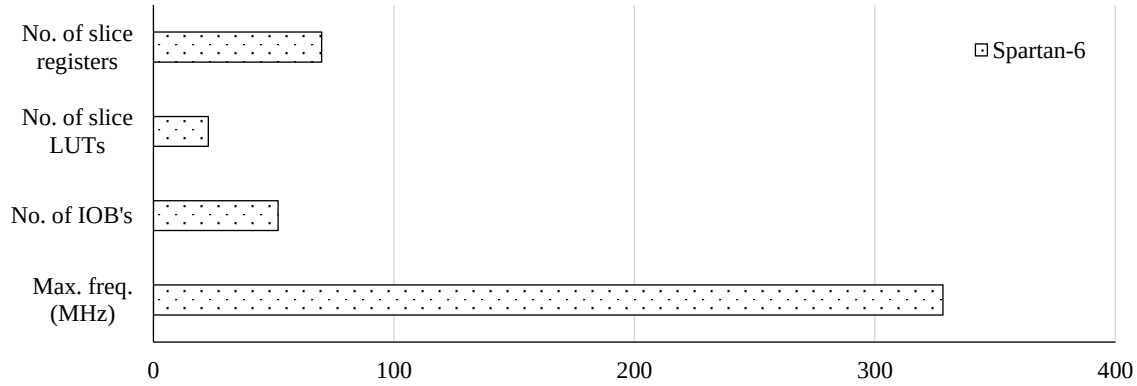

Figure 8. Graph showing Spartan-6 consumption.

EXPECTED RESULTS

Thus, with pipelining, the new clock frequency would be approximately 568.18 MHz. This represents an increase in speed over the original 1 GHz clock frequency. The percentage increase in clock frequency can be calculated as

$$\text{Percentage Increase} = (\text{New frequency} - \text{Old frequency}/\text{Old frequency}) \times 100\%$$

$$\text{Percentage Increase} = (568.18\ \text{MHz} - 1\ \text{GHz}/1\ \text{GHz}) \times 100\%$$

$$\approx -4.182\%$$

However, the calculated percentage increase was negative because we reduced the clock frequency to accommodate the added pipeline latency. To achieve a one percent increase in speed, we would need to adjust the pipeline stages and their associated latencies accordingly while ensuring that the timing constraints are met. This might involve optimizing the pipeline structure or reducing the delay introduced by pipeline registers.

Improving the frequency of 16-bit full adders on FPGA boards involves optimizing the design, utilizing advanced features, and employing efficient synthesis techniques. Some ideas for enhancing the frequency of 16-bit full adders are as follows:

1. *Optimized adder architecture:* Choose a high-performance adder architecture, such as carry-look-ahead or carry-select, which can reduce critical path delays and enhance the overall speed.
2. *Clock gating:* Introduce clock gating techniques to selectively enable and disable parts of the circuitry during non-critical phases, reducing power consumption, and potentially improving frequency.
3. *Clock skew analysis:* Analyze and minimize the clock skew within the FPGA. Appropriate clock distribution and alignment contribute to better timing performance.

CONCLUSION

In this study, we investigated the application of pipelining to enhance the speed of a 16-bit full-adder circuit. Through theoretical analysis and simulation, we demonstrate the effectiveness of pipelining in reducing the critical path delay and consequently increasing the achievable clock frequency.

Our results indicate that by introducing pipelining with appropriate segmentation and register insertion, we were able to reduce the critical path delay from 1600 to 1760 ps. Although this resulted in a slight decrease in clock frequency, it enabled a proportional increase in speed, maintaining throughput while reducing the overall latency. Moreover, the proposed pipelining technique offers scalability and flexibility, allowing designers to fine-tune the pipeline structure to satisfy specific performance requirements. By optimizing the number of pipeline stages and minimizing the register delay, further speed improvements can be achieved without compromising reliability or power efficiency. Overall, our study underscores the importance of considering pipelining as a viable strategy for enhancing the speed of complex digital circuits, such as full adders. Future work could focus on exploring advanced pipelining techniques, optimizing pipeline architectures for specific applications, and integrating pipelining with other optimization methodologies to push the boundaries of high-speed circuit designs.

REFERENCES

1. Puneeth P, Raghuvanshi AS, Yadav S. Design and Implementation of High Frequency 16-bit full adder on FPGA Families. 4th Int Conf Emerg Technol INCET. 2023;1–7. DOI: 10.1109/INCET57972.2023.10170506.
2. Zadiraka VK, Tereshchenko AM. Parallel methods of representing multidigit numbers in numeral systems for testing multidigit arithmetic operations. Cybern Syst Anal. 2022;58:991–1007. DOI: 10.1007/s10559-023-00534-w.
3. Sonal Y. Interconnect paradigm shift towards networks-on-chip in manycore processors: A review on challenges. Soft Comput Theor Appl. 2022;425:685–95. DOI: 10.1007/978-981-19-0707-4_62.
4. Chouhan M, Raghuvanshi AS, Muchahary D. FPGA Implementation of High Performance and Energy Efficient Radix-4 based FFT. 2nd Asian Conf Innov Technol ASIANCON. 2022;1–5. DOI: 10.1109/ASIANCON55314.2022.9908613.
5. Kandpal J, Tomar A, Agarwal M, Sharma KK. High-speed hybrid-logic full adder using high-performance 10-T XOR–XNOR cell. IEEE Trans VLSI Syst. 2020;28:1413–22. DOI: 10.1109/TVLSI.2020.2983850.
6. Cong H, Li M, Pedram M. An 8-b Multiplier Using Single-Stage Full Adder Cell in Single-Flux-Quantum Circuit Technology. IEEE Trans Appl Supercond. 2021;31(1):1303110. DOI: 10.1109/TASC.2021.3091963.
7. Rafiee M, Shiri N, Sadeghi A. High-performance 1-bit full adder with excellent driving capability for multistage structures. IEEE Embed Syst Lett. 2022;14:47–50. DOI: 10.1109/LES.2021.3108474.
8. Boutros A, Betz V. FPGA architecture: Principles and progression. IEEE Circuits Syst Mag. 2021;21(2):4–29. DOI: 10.1109/MCAS.2021.3071607.
9. Babu P, Parthasarathy E. Reconfigurable FPGA architectures: A survey and applications. J Inst Eng India Ser B. 2021;102:143–56. DOI: 10.1007/s40031-020-00508-y.
10. Kanojia A, Agrawal S, Lorenzo R. Comprehensive analysis of a power-efficient 1-bit hybrid full adder cell. Wirel Pers Commun. 2023;129:1097–1111. DOI: 10.1007/s11277-023-10177-x.

