Proceedings of the 2023 International Conference on Machine Learning and Automation
DOI: 10.54254/2755-2721/37/20230516

Enhancing multiplication efficiency: Harnessing Booth algorithm and Wallace tree synergy for 32-bit signed multipliers

Panpan Chen

School of Engineering, University of Sydney, Sydney, 2050, Australia

chenpanpanau@163.com

Abstract. This paper focuses on exploring performance optimization strategies for 32-bit signed multipliers in the realm of digital computation. The ingenious combination of the Booth algorithm and the Wallace tree takes center stage as the core of this research. Through a thorough analysis of this combined technique, the paper unveils its multifaceted advantages in multiplication operations. The unique contribution of the Booth algorithm is elaborately discussed, which involves streamlining the multiplication process by reducing the number of addition operations. Subsequently, the exceptional performance of the Wallace tree in accelerating the merging of partial products is highlighted. Through the synergistic interplay of these two techniques, the efficiency of multiplication operations is significantly enhanced, particularly suited for domains necessitating high-speed arithmetic computations, such as graphics processing and digital signal processing. The paper delves further into the significance of optimizing resource utilization, particularly in the field of integrated circuit design. By minimizing addition operations and optimizing parallel processing, the complexity of hardware implementation is reduced, leading to a notable decrease in resource requirements. This advantage proves invaluable in modern integrated circuit design, aiding designers in striking an optimal balance between spatial efficiency and power consumption.

Keywords: Multiplication Efficiency, Booth Algorithm, Wallace Tree, 32-bit Signed Multipliers.

1. Introduction

In the realm of digital computing, optimizing multiplication operations holds significant importance. Among the array of techniques available, the combination of the Booth algorithm and the Wallace tree stands out as a compelling strategy, particularly when aiming to enhance the performance of a 32-bit signed multiplier [1]. This amalgamation offers a multifaceted set of benefits that extend across various domains, such as graphics processing and digital signal processing, where swift arithmetic operations are a necessity.

The synergy between the Booth algorithm and the Wallace tree yields remarkable performance improvements. The Booth algorithm's prowess lies in its ability to curtail the count of additions required, effectively streamlining the process. Meanwhile, the Wallace tree excels in expediting the amalgamation of partial products, which collectively contributes to substantial time savings. This acceleration in multiplication performance holds undeniable value for applications that thrive on expeditious computations [2]. Notably, the advantages extend beyond mere performance enhancements. By minimizing the need for addition operations and optimizing parallel processing, a reduction in hardware complexity and resource utilization is achieved. This attribute proves particularly advantageous in the intricate realm of integrated circuit design. As designers strive to strike a balance between spatial efficiency and power consumption, the capacity to economize on hardware resources becomes a pivotal asset [3]. Moreover, the scalability of these techniques augments their appeal. While initially tailored for 32-bit multipliers, their applicability gracefully extends to larger bit widths, such as 64-bit or 128-bit architectures. Crucially, this scalability doesn't translate to an exponential upsurge in complexity, reinforcing the feasibility of incorporating these techniques into a diverse array of computing contexts.

In summation, the integration of the Booth algorithm and Wallace tree within a 32-bit signed multiplier framework underscores its potential to yield heightened efficiency, diminish resource requisites, and exhibit robust scalability. In the landscape of modern integrated circuit design challenges, these attributes collectively assume paramount significance, meeting the burgeoning demands of contemporary computational needs.

2. Basic theory analysis

The Wallace Tree and Booth algorithms are two important algorithms used in digital computing and signal processing, especially when implementing multiplication operations in computer hardware. Wallace Tree: A Wallace tree is a hardware implementation of a fast multiplier, named after its inventor, Chris Wallace. Wallace trees use a parallel strategy to compute the product of two numbers, greatly speeding up multiplication operations. Wallace tree multipliers are commonly used in modern computers, graphics processing units GPUs), digital signal processors DSPs), and other digital systems [4]. Booth’s algorithm is an algorithm for multiplying binary numbers, named after its inventor, Andrew D. Booth. The main feature of Booth's algorithm is that it reduces the number of additions required for multiplication, thereby increasing computational speed. Booth's algorithm is used in many digital systems, especially in resource-constrained embedded systems. The significance of these two algorithms is mainly due to their ability to increase the speed and efficiency of digital computation. Fast multiplication operations are the key to many complex algorithms and applications, such as image processing, audio coding and decoding, and deep learning. By optimizing multiplication operations, we can make these applications run faster or on smaller, cheaper, and lower power hardware devices.

Wallace tree and Booth algorithms are two commonly used multiplier design strategies to speed up the addition process and reduce the number of operations for multiplication, respectively. These two algorithms can be combined to achieve efficient multiplication operations when designing 32-bit multipliers [5]. These two algorithms have their own advantages combined for multiplication optimization, Booth's algorithm, which is an optimized multiplication algorithm used to reduce the number of additions and subtractions required in a multiplication operation. Booth’s algorithm reduces the number of multiplication operations that need to be carried out by looking at consecutive bits in the multiplication operands and then using shifts and addition or subtraction operations. A Wallace tree is an adder structure used to perform multiple bitwise addition operations in parallel [6].

In multipliers, Wallace trees are used to speed up the process of adding partial products. Wallace trees can significantly reduce the number of clock cycles required by performing addition operations in parallel. According to the information found, in the design of a 32-bit multiplier, the number of multiplication operations can be reduced by first encoding the operands using the Booth algorithm. The resulting partial product is then fed into the Wallace tree to perform the addition operations in parallel and obtain the final product. In this way, the advantages of both the Booth algorithm and the Wallace tree can be utilized to achieve efficient 32-bit multiplication operations.

3. 32-bit multiplication algorithm implementation

3.1. 32-bit multiplication is implemented using the booth algorithm

Booth's algorithm is a multiplication algorithm for signed multiplication of binary numbers. It works for integers and fixed-point binary numbers. The main advantage of this algorithm is that it reduces the number of multiplication operations. Following is the detailed principle of Booth's algorithm: The basic concept is understanding the representation of two signed integers [7]. Typically, use two methods to represent negative numbers: inverse and complement. Booth's algorithm is based on the principle of complement. Speaking of complement code, then must explain the basic principles of complement code: The first is the complement of a positive number is the same as its prime number. Then the complement of a negative number is the inverse of its original code plus one [8].

The next Booth algorithm is mainly divided into three categories as for Initialization, Store the complement of the multiplied number M and the multiplier Q in a register,then,Set an extra bit Q_{-1} to 0. This is the first bit of the rightmost bit of multiplier Q, initially set to 0.next Set an accumulator AC, initially set to 0,at last Set a counter n indicating the number of bits of the operation.as for main steps, For each iteration, we consider the rightmost two bits of the multiplier Q: the current bit Q_0 and the previous bit Q_{-1}, (1) If Q_0Q_{-1} = 00, no operation is performed.(2) If Q_0Q_{-1} = 01, add AC to the complement of the multiplied number M. (3) If Q_0Q_{-1} = 10, add AC to the inverse complement (or negative complement) of the multiplied number M. (4) If Q_0Q_{-1} = 11, no operation is performed. At the end of each iteration, Perform a right shift operation on registers AC and Q. That isQ_{-1}, acquires the value ofQ_0, Q_0acquires the value of Q_{-1} and so on. Then Decrease the value of the counter. The Booth algorithm considers consecutive 1's and 0's, which reduces the number of accumulations [9]. For regular multiplication methods, it is possible that many additions are required, but with Booth's algorithm, this number may be reduced. However, Booth's algorithm is suitable for signed binary multiplication. It is based on complementary multiplication, so the complementary representations of the multiplier and the multiplied must be determined first.

3.2. 32-bit multiplication using Wallace tree

Wallace tree is an efficient addition hardware structure for digital multiplication operations. Its core idea is to reduce the number of partial products generated in multiplication as fast as possible, thus speeding up multi-bit multiplication operations.

When performing binary multiplication, we get multiple partial products. The traditional approach is to accumulate these partial products one by one. In contrast, Wallace trees use a parallelization strategy to speed up the process. There are three principles to introduction the Wallace tree, (1) Partial Product Generation: First AND is performed on each bit of the multiplier and the multiplicand to generate the partial product. (2) Partial product reduction: The partial products are then summed in parallel using a multi-level adder structure to reduce their number. In each stage, three partial products are added to produce two outputs. If the sum is 2 or 3, then a rounding bit is produced. The output and any rounding are fed to the next level. (3) Partial Accumulation Addition: Finally, when the number of partial products has been reduced to a sufficiently small number, an ordinary adder can be used to complete the accumulation process and obtain the result of multiplication [10].

The Wallace tree takes advantage of parallel processing to greatly accelerate the partial accumulation addition process in multibit multiplication. This approach is very useful in digital signal processing and computer arithmetic, especially in hardware implementations, as it can significantly increase the speed of multiplication operations.

4. Integrated design of Wallace tree and booth algorithm

4.1. Define ports

A full adder module is first defined with two input signals a and b, and a rounded input signalC_{in}. It outputs two signals, sum for the result of the addition andC_{out} for the rounded output. In hardware circuits, multiple full adders can be combined to build more complex adders such as adder arrays and multipliers.

4.2. code writing

Firstly, module definition for the implementation of a 32-bit multiplier. This multiplier receives two 32-bit input signals, I_mul_1 and I_mul_2, and outputs a 64-bit result letter, O_dataout, which, of the input and output signals, represents the product of the two input multipliers. This next section of code declares several internal signals wires), including signals of 32-bit, 64-bit, and 65-bit widths, as well as one- and two-dimensional arrays. These signals can be used to connect various parts of the module internally for data transfer and computation.

Next, the generate block is used to generate a 32-bit wide y-array to store the bits of the input signal "I_mul_2". Each element of the array consists of three consecutive bits, where y[1] consists of bits 1 and 0 of I_mul_2, and the rest of the elements y[3], y[5]...y[31] consist of bits 3, 5, 7...31 of I_mul_2, respectively .

In the code, the input signal I_mul_1 is first shifted left by one bit, and then the result is inverted and assigned to the S_mu1l_buf signal. y array is stored by using the decoding of the three-bit code to get the corresponding operation, and for the three-bit code 100/101/110 there exists a complementary subtraction, and the computed number of splices is the inverse of A or the left-shift of A by 1 bit and then inverted, so that the plus the inverse code part is completed .

The next step is the signal multiplication step, this code implements an important step in the multiplier, which is to multiply the input signal I_mul_1 with the selection signals y[2*i-1], the purpose here is to decide which bits of the input signal I_mul_1 need to be involved in the multiplication operation based on the values of these selection signals, and finally store the result in the S_data_n array .

The last part of the adder is designed to implement part of the multiplier's functionality, specifically instantiating the adder module cyclically through a generate block and performing the addition operation on the elements of the S_data_n array and storing the result in the S_data_s array, as well as storing the rounding in the S_data_c array. The intermediate result and the rounding case of the multiplier are computed step by step by nesting the loop and finally the complete multiplication result is obtained .

4.3. Testbench

The whole purpose of Testbench is to verify the functionality of multiplier module. It initializes the input signal to 2 in the initialization block and then waits for 50 clock cycles, during which it generates new multipliers 1 and 2 continuously and randomly through the always block to simulate different inputs. At the same time, it calculates the output result 2 by direct multiplication to compare it with the output result 1 of the multiplier module to verify the correctness of the multiplier module. The test ends when the simulation is executed to stop .

And the whole process is shown in figure 1.

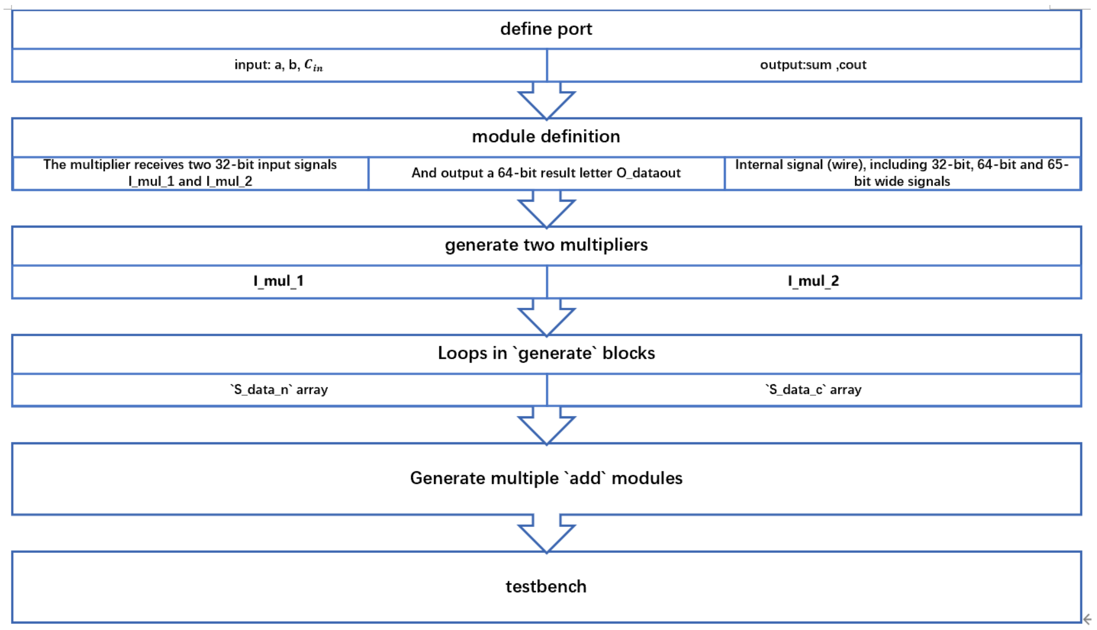
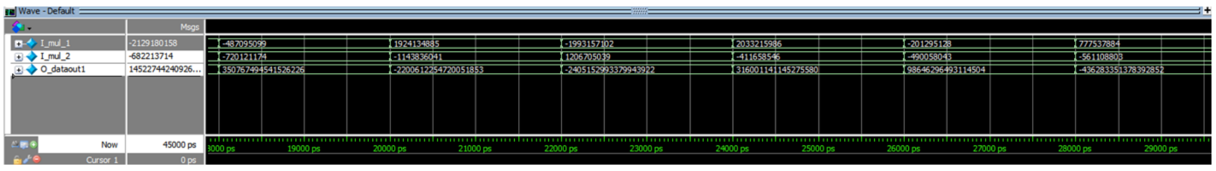

define port
input: a, b, C_{in}
output:sum ,cout
module definition
The multiplier receives two 32-bit input signals I_mul_1 and I_mul_2
And output a 64-bit result letter O_dataout
Internal signal (wire), including 32-bit, 64-bit and 65-bit wide signals
generate two multipliers
I_mul_1
I_mul_2
Loops in `generate` blocks
`S_data_n` array
`S_data_c` array
Generate multiple `add` modules
testbench

Figure 1. The whole process

4.4. Result analysis

A 32-bit signed multiplier was finally implemented, and an adder was used to sum the partial products. In the multiplication operation, each bit of the multiplier is multiplied by the multiplied number to get the partial product and then all the partial products are added together to get the result. This addition of partial products is a key step in the implementation of a multiplier. Multiplication in a multiplier means that each bit of the multiplier is logically operated with the multiplied number to get the partial product which is shown in figure 2.


```
Wave - Default
                    Msgs
I_mul_1     -2129180158          -487095099          1924134885           -1993157102           2033215986          -201295128          777537884
I_mul_2     -682213714           -720121174          -1143836041          1206705039            -411658546          -490058043          -561108803
O_dataout1  14522744240926...    350767494541526226  -2200612254720051853 -2405152993379943922  316001141145275580  98646296493114504   -436283351378392852

Now         45000 ps   3000 ps  19000 ps  20000 ps  21000 ps  22000 ps  23000 ps  24000 ps  25000 ps  26000 ps  27000 ps  28000 ps  29000 ps
Cursor 1    0 ps
```

Figure 2. Result analysis

5. Conclusion

In conclusion, this paper has introduced an innovative approach that combines the power of the Wallace tree and Booth's algorithm to optimize the performance of a 32-bit signed multiplier. The primary goal of this research has been to devise an optimized algorithm for multiplication operations, leveraging the strengths of both techniques.

However, this endeavor is not without challenges. Implementing a 32-bit multiplier using these methods introduces complexities due to the involvement of various hardware components, such as adders and shifters. To address this, adopting modular and hierarchical design approaches can simplify the process and enhance efficiency. Latency is another challenge that arises from the use of the Wallace tree's parallel processing capabilities, potentially leading to increased processing time. To mitigate this, exploring more efficient parallel adder structures like Dadda trees can help reduce latency. Resource consumption becomes a concern, especially in FPGA or ASIC implementations where parallel processing demands more hardware resources. Optimizing the design by eliminating unnecessary logic gates and streamlining interconnections is crucial. Additionally, selecting appropriate Booth encoding versions, tailored to the specific application needs, can strike a balance between resource consumption and performance. Testing and verification are critical due to the intricate nature of the design. Employing formal verification methods, simulation tools, and automated test case generation ensures comprehensive testing coverage, guaranteeing the multiplier's correct functionality. Moreover, the parallel processing paradigm could lead to increased power consumption. Implementing low-power design techniques, optimizing clock frequency, and employing effective power management strategies can help mitigate this concern. Lastly, the design's scalability to accommodate different bit widths is essential. Employing parametric design principles empowers the multiplier to flexibly adapt to various numerical scales.

In summary, while challenges exist in implementing a 32-bit signed multiplier using the Booth algorithm and Wallace tree, strategic design and optimization can effectively overcome these obstacles. The resulting multiplier design showcases efficiency, reduced power consumption, and the ability to seamlessly scale to meet diverse computational needs. This research contributes not only to innovative algorithmic solutions but also to the advancement of computational efficiency in modern digital systems.

References

[1] Fu Chengjie,Zhu Xiaolei,Huang Kejie & Gu Zheng.(2021).An 8-bit Radix-4 Non-Volatile Parallel Multiplier. Electronics(19). doi:10.3390/ELECTRONICS10192358.

[2] N. Arumugam & B. Paramasivan.(2021).An integrated FIR adaptive filter design by hybridizing canonical signed digit (CSD) and approximate booth recode (ABR) algorithm in DA architecture for the reduction of noise in the sensor nodes. Multidimensional Systems and Signal Processing(4). doi:10.1007/S11045-021-00783-Y.

[3] M. Arulkumar & M. Chandrasekaran.(2021).An Improved VLSI Design of the ALU Based FIR Filter for Biomedical Image Filtering Application. Current Medical Imaging(2).

[4] John Mar M. Castillo, Sarah Mae B. Macabangon, Ricardo Gary B. Garcia III... & Michael John M.Villar.(2019).Construction of Carbon Fiber Industry Chain Integration Framework Based on Industrial Technology Breakthrough. Indian Journal of Public Health Research & Development(1).

[5] Lakshmi kiranMukkara & Kota VenkataRamanaiah.(2018).FPGA Implementation of High Speed Digital FIR Filter. International Journal of Advanced Networking and Applications(2).

[6] Sharmila Hemanandh & Siva Subramanium.(2015).Performance Evaluation of High Speed Radix 8 Tree Based Multiplier. Indian Journal of Science and Technology(33). doi:10.17485/ijst/2015/v8i33/123353.

[7] Himanshu Bansal,K. G. Sharma & Tripti Sharma.(2014).Wallace Tree Multiplier Designs: A Performance Comparison Review. Innovative Systems Design and Engineering.

[8] Subhashis Maitra & Amitabha Sinha.(2013).High efficiency MAC unit used in digital signal processing and elliptic curve cryptography. ACM SIGARCH Computer Architecture News(4). doi:10.1145/2560488.2560490.

[9] Renping Wang.(2012).Full-custom Design and Implementation of High-performance Multiplier..(eds.)Proceedings of 2012 2nd International Conference on Materials Engineering for Advanced Technologies(ICMEAT 2012 V631-632)(pp.1462-1468).TRANS TECH PUBLICATIONS.

[10] Xiang Ping Qian,Wei Ming Qiao,Zhong Zu Zhou... & Lan Jing.(2012).A Digital Regulator for FPGA Implementation. Advanced Materials Research(433-440). doi:10.4028/www.scientific.net/AMR.433-440.4547.