Design and Evaluation of Booth’s Algorithm for Drone Applications

Asavari Arun Hangal[1], Divya Shree Tala[2], Dr. Jeeru Dinesh Reddy[3], Dr. Maligi Anantha Sunil [4]
[1,2]Student, BMS College of Engineering
[3,4]Assistant Professor, BMS College of Engineering

***Abstract*—This paper presents the design and evaluation of Booth's Algorithm, which can be used for real-time altitude calculations in drone and UAV systems. Our design increases the processing speed and accuracy of altitude estimates produced during drone flying by utilising the traditional Booth's Algorithm, which is well-known for effectively multiplying binary numbers. This particular design uses 45nm technology using Cadence software to construct a control and data channel flow and an RTL (Register Transfer Level) to GDSII (Graphic Data System II) methodology, which guarantees reliable integration into drone system hardware architectures. To satisfy the demanding requirements of real-time operations for Drones and UAVs altitude and distance calculations, the key design constraints include maximising computational efficiency, minimising latency, minimising overall area needed, and optimising power consumption. Drone design performance can be evaluated by simulations and real-world testing, which demonstrate how effectively the algorithm manages abrupt changes in altitude under various climatic conditions. The results show that high-performance drone applications could make use of Booth's Algorithm, which offers a trade-off between accuracy and resource efficiency while adhering to industry norms for system-on-chip (SoC) design.**

***Index Terms*—Booth’s algorithm, RTL, GDSII, Booth’s Multiplication, Drones, Cadence.**

I. INTRODUCTION

[1] Multiplication algorithms are important in performing multiplication well and within a computer system. Depending on the application, complexity, accuracy and time , various algorithms exist. There are many complex steps behind just multiplying two numbers, which may be used to determine important coordinates or reach a conclusion in massive machines. Selecting the right application specific algorithm requires evaluating a number of variables, including processing large data files, accuracy, and speed. Power-hungry devices like drones depend on efficient multiplication algorithms since they cut down on computing time and resource use. Self-driving drones like UAVs require a lot of power in staying in air and consume a lot of power in making the right calculations for predicting the coordinates and taking the decisions on time, in order to avoid clashes or any damage to the drone.

The computer has to do less work as Booth's algorithm recognised repeated patterns and discards them. This algorithm can be used to determine the right coordinates and altitudes. [8] Speed in calculations is critical for drones and unmanned aerial vehicles (UAVs) due to the real-time nature of their operations. These systems must process massive volumes of sensor data, such as altitude, location, and environmental variables, while also running control algorithms for stability, navigation, and obstacle avoidance. In applications such as autonomous flight, surveillance, or delivery, even minor processing delays might result in erroneous replies, impairing aircraft performance or, worse, triggering collision. Quick and efficient calculations enable us to make decisions in milliseconds, in turn assist in adapting to changing situations, and optimize flight trajectories, which is critical for mission success.

Furthermore, quick computing allows for more efficient use of onboard resources, which is especially important in drones with limited power and computational capabilities. More complicated algorithms, such as Schönhage-Strassen and Karatsuba, are faster for large numbers but have extra overhead, which makes them unsuitable for small systems like drones. [1] Algorithms such as Toom-Cook and FFT, which have complexities approaching O(nlogn), are similarly suited for very large inputs but are generally unnecessary for small to medium-sized tasks where Booth’s algorithm remains practical. Based on these facts we have decided to choose Booth’s algorithm as the best fit algorithm.

II. BACKGROUND

Most of the well-known multiplication algorithms [1] like the classical multiplication, Karatsuba-Ofman, Toom-Cook's, and Schönhage-Strassen, the latter of which employs Fast Fourier Transform (FFT) for multiplication [2]. The classical multiplication algorithm that reduces delay time by using multiplexers can be optimized in hardware [3].

The Karatsuba multiplication algorithm, on the other hand, being a recursive binary subdivision method, due to its efficiency, has been widely adopted, often as part of hybrid algorithms. Karatsuba algorithm can be combined with other techniques like the Nikhilam sutra [4], Vedic multiplier, and Kogge-Stone algorithms [5] to improve the performance and efficiency. Advancements in other algorithms along with these combinations, have significantly contributed to multiplication which is effective and efficient, especially in digital systems and cryptographic applications, where both speed and resource efficiency are crucial.

Booth's algorithm is used in hardware multiplication because it can handle signed integers efficiently and reduces the number of partial products, which reduces the complexity of addition and subtraction computations. By combining the multiplier's bits and allowing it to skip unnecessary steps when it comes across zero sequences, Booth's recoding method optimizes the number of operations compared to traditional multiplication algorithms. This speeds up the whole process and also removes pointless calculations, which is particularly useful for operands that contain long strings of identical bits. Booth's method can be applied to UAVs and Drones as it saves time by discarding pointless calculations[2] in time critical situations. For hardware-based applications like UAVs and Drones where power, area, and latency are crucial design restrictions, given the size of the device and the time- critical nature of the device. Hence because of its quickness and versatility, we have chosen Booth's algorithm.

[3] The authors have built an Innovative Booth's algorithm that can be generalized to N-bit multiplier design. In comparison to the original Booth multiplier, it speeds up the computation from the time complexity and reduces the logic delay. Furthermore, its parallel computation characteristic makes it more suitable for large number multipliers. But we have chosen Traditional Booth's algorithm as Drones and UAVs don't perform large number multiplication and this Innovative Booth's algorithm also claims to have a higher path delay making it almost equivalent to the traditional one.

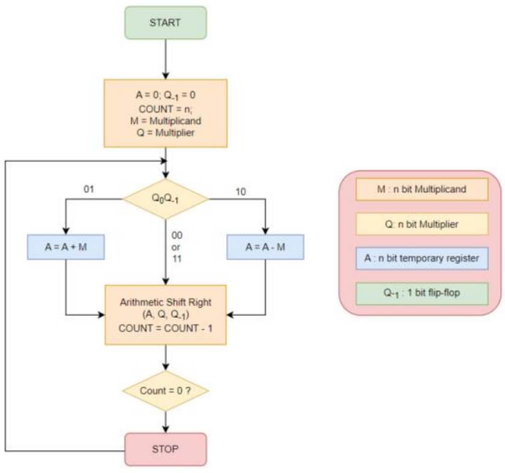


Figure 1: Booth Algorithm's Flow chart

III. METHODOLOGY

Booth's algorithm is considered an efficient method for the multiplication of binary numbers as it examines pairs of bits in the multiplier and performs appropriate additions or subtractions to the accumulator. Signed-digit representation concept is implemented to reduce the number of operations required instead of shifting and adding the multiplicand repeatedly. This approach significantly improves multiplication operations performance, especially for large multipliers. The algorithm which uses Right-Shift - Circular Booth Algorithm is depicted in Figure 1.

Initially, the Accumulator register A and the Q-1 bit are first initialized to zero, while the COUNT serves as a sequence counter that represents the total number of bit set to n = 4. The multiplicand (4 bits) is denoted by M and the multiplier (4 bits) is represented by the letter Q. If the 2 bits of the multiplier are 10, the accumulator undergoes subtraction by the multiplier value. If the 2 multiplier bits are 01, then the multiplicand is added to the Accumulator (A). No change occurs in the Accumulator value if the bit values are 00 or 11. Subsequently, arithmetic shift operations are performed which shifts the contents of A, and a right circular shift is executed which shifts the contents of Q alongside the Q-1 bit.

Based on the obtained bits, the controller decides whether to add the multiplicand to the accumulator, subtract the multiplicand from the accumulator or just shift the contents of the accumulator and the multiplier. The sequence counter is continually decremented by the controller after each iteration until the count reaches 0, signifying the completion of the

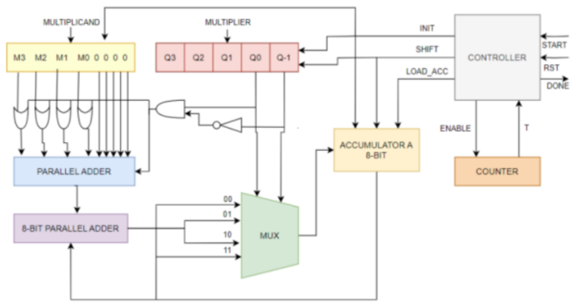


Figure 2: Functional Block Diagram of Booth's Algorithm

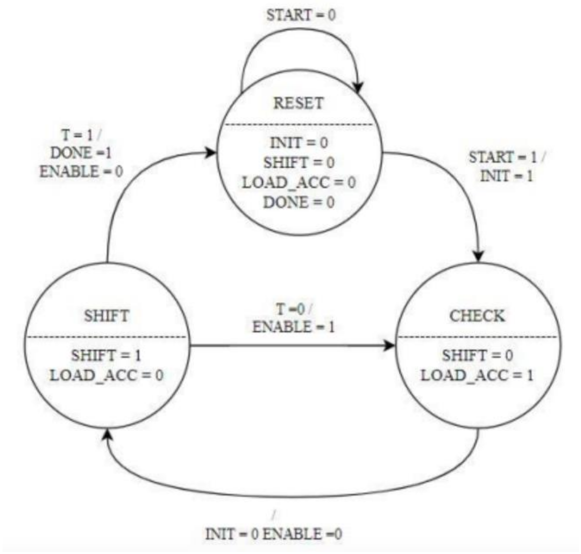


Figure 3: FSM of the Algorithm

entire computation. At this point, the product output is extracted from register A which culminates the multiplication process. Accumulator stores the final result of the multiplication. The functional block diagram which depicts the RTL design architecture approach followed to implement multiplication as shown in Figure 2.

The FSM of the design for the control path is as shown in Figure 3.

The FSM has three states: START, SHIFT, and CHECK.

- START: This is the initial state where the signals in the FSM are reset and initialized. The START state sets the START, INIT, SHIFT, LOAD_ACC, and DONE signals initially to zero.
- CHECK: In this state, the FSM checks for the least significant 2 bits of the multiplier. Based on these bits, it determines whether to

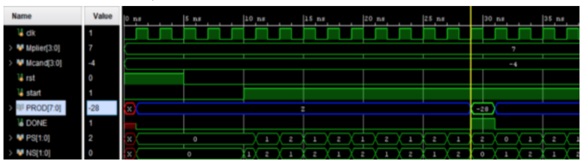


Figure 4: Simulation Result of Signed Multiplier Design

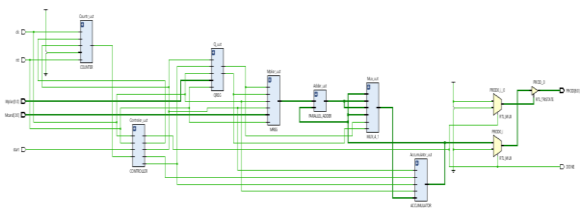


Figure 5: Schematic diagram of the design

subtract the multiplicand to the accumulator or add it. The SHIFT signal is set to 0, and the LOAD_ACC signal is set to 1 if an addition or subtraction is required. For the next iteration, the FSM transitions back to the SHIFT state.
- SHIFT: In this state, the FSM controls the shift operation of the multiplier register. The SHIFT signal is set to 1, and the LOAD_ACC signal is set to 0. The FSM transitions to the RESET state when the shift operation is complete.

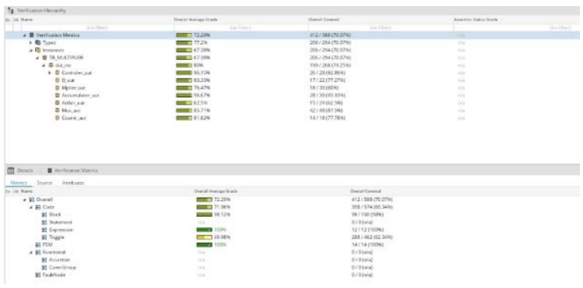
The FSM also includes the signals T, DONE, and ENABLE. T is a clock signal that controls the state transitions. DONE signal indicates if the multiplication process is completed or not. ENABLE controls the overall operation of the FSM. The FSM transitions between these states based on the input signals and the current state.

IV. RESULTS

Simulation result for the RTL design for a typical case of Multiplier=7 and Multiplicand=-4 is as shown in Figure 4. It can be seen product -28 is obtained at the output after performing right shift and addition for 4 cycles. The schematic of the entire design is generated on Xilinx Vivado and is depicted as shown in Figure 5.

We performed Code Coverage analysis on Cadence and got the below results:

- Branch = 87.5%
- Decision = 90.07%
- Condition = 90%
- Statement = 90.07%



Verification Hierarchy

Name	Overall Average Grade	Overall Covered	Assertion Status Grade
Verification Metrics	72.29%	412 / 588 (70.07%)	n/a
Types	77.2%	206 / 294 (70.07%)	n/a
Instances	67.38%	206 / 294 (70.07%)	n/a
TB_MULTIPLIER	67.38%	206 / 294 (70.07%)	n/a
dut_inst	60%	199 / 268 (74.25%)	n/a
Controller_uut	95.15%	26 / 28 (92.86%)	n/a
Q_uut	83.33%	17 / 22 (77.27%)	n/a
Mplier_uut	76.47%	18 / 30 (60%)	n/a
Accumulator_uut	96.67%	28 / 30 (93.33%)	n/a
Adder_uut	62.5%	15 / 24 (62.5%)	n/a
Mux_uut	85.71%	42 / 48 (87.5%)	n/a
Counter_uut	81.82%	14 / 18 (77.78%)	n/a

Details | Verification Metrics

Metrics | Source | Attributes

Name	Overall Average Grade	Overall Covered
Overall	72.29%	412 / 588 (70.07%)
Code	71.36%	398 / 574 (69.34%)
Block	98.12%	98 / 100 (98%)
Statement	n/a	0 / 0 (n/a)
Expression	100%	12 / 12 (100%)
Toggle	49.98%	288 / 462 (62.34%)
FSM	100%	14 / 14 (100%)
Functional	n/a	0 / 0 (n/a)
Assertion	n/a	0 / 0 (n/a)
CoverGroup	n/a	0 / 0 (n/a)
FaultNode	n/a	0 / 0 (n/a)

Figure 6: Code Coverage Analysis

Table I: Comparison of Power, Area and Timing reports after synthesis and after Nano routing respectively

Process/ Characteristic	After Synthesis	After Nano routing
Power	28.24 micro Watts	0.077 Watts
Area	432.630	429.894
Timing	3883ps	3750ps

The overall average grade and overall covered is as shown in Figure 6.

The ASIC design process basically begins with the conceptual stage, where ideas are translated into a formal design specification. This stage mainly involves two main stages, Design Entry and Functional Simulation. In the design entry stage, engineers use Hardware Description Languages (HDLs) like Verilog or VHDL to create a high-level representation of the design required. In our model we have used Verilog. This representation, also known as Register Transfer Level (RTL), defines the logic in terms of registers and the combinational logic between them. In the next stage, i.e. functional simulation stage we had performed functional simulations to verify that the design behaves as intended using a testbench on a software called Vivado.

However, we can't draw accurate timing information from these simulations as the design isn't mapped to physical hardware and the placement and routing is not done yet. Once the functional verification is done, we have moved onto the synthesis stage. Synthesis is an important step that transforms the high-level RTL description into a much more concrete representation. This stage yields a gate-level netlist, and also provides improved estimates and optimization in comparison to the previous stage reports produced.

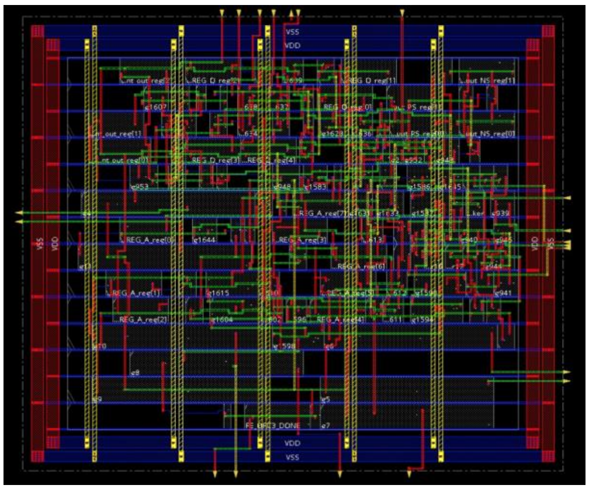


Figure 7: Final GDSII after Nanorouting

The synthesis tool we used in our model was Cadence Genus. Legacy Genus converted the RTL description into a gate-level netlist, representing the design in terms of actual logic gates and flip-flops from a standard cell library. Post-synthesis, we got more accurate timing, power and area estimates. The synthesized netlist is logically equivalent to the RTL design but is optimized for performance, area, and power based on specified constraints given by us. The power is much lesser when compared to the power in Nanorouting report as the blocks are not routing and running with respect to each other's outputs and behaviour. The comparison of power, area and timing reports generated before and after nanorouting is shown in Table I. The final major stage in ASIC design is nanorouting, also known as Place and Route (P&R). The tool we used for this stage is Cadence Innovus. Innovus takes the synthesized logic cells and places them onto the virtual silicon die. Table I showcases the changes in power, slack and area after synthesis and nanorouting. In nanorouting the interconnections between them are routed and can be observed as shown in Figure 7. Post-routing, timing analysis showcases actual wire delays and parasitic delays and its effects, providing the most accurate performance estimates. Power analysis also becomes more precise, considering the actual routing and placement. It reflects various estimates like the leakage power, power consumed by each block and the percentage of power consumed by each block. This is like conducting a final inspection of the building, measuring its actual performance against the initial specifications. Design Rule Checking (DRC) and Layout Versus Schematic (LVS) checks ensure the physical layout adheres to manufacturing constraints and matches the original netlist. Both of which are performed by the cadence tool. Then finally after nanorouting the design is produced and saved.

V. CONCLUSION

Booth's algorithm is proved to be efficient for drone applications as the area consumption and cell count is comparatively less when compared to other multiplication algorithms. Positive slack indicates that the design meets all the timing requirements and works much faster than other algorithms. The algorithm’s ability to perform fast binary multiplication with fewer computational steps greatly reduces latency, which makes it ideal for drones that require swift and precise adjustments to altitude in dynamic environments. Power consumption of the algorithm uses minimal watts which makes it robust and performance efficient.

REFERENCES

[1] Wibowo, F. W. (2018). Comparison of Multiplication Algorithms Based on FPGA. 2018 2nd Borneo International Conference on Applied Mathematics and Engineering (BICAME). doi:10.1109/bicame45512.2018.1570505372

[2] S. Jahani, A. Samsudin and K. G. Subramanian, “Efficient big integer multiplication and squaring algorithms for cryptographic applications,” Journal of Applied Mathematics, vol. 2014, article ID 107109, 9 pages, 2014. http://dx.doi.org/10.1155/2014/107109

[3] Dhanabalan and T. Selvi, “FPGA implementation of 8-bit multiplier with reduced delay time,” International Journal of Computer and Communication Engineering, vol. 2, no. 6, pp. 665-668, November 2013. http://dx.doi.org/10.7763/IJCCE.2013.V2.270

[4] M. N. Angeline and S. Valarmathy, “Implementation of n-bit binary multiplication using n-1 bit multiplication based on nikhilam sutra and karatsuba principles using complement method,” Circuits and Systems, vol. 7, pp. 2332-2338, 2016. http://dx.doi.org/10.4236/cs.2016.79203

[5] M. C. Sudeep, B. M. Sharath and M. Vucha, “Design and FPGA implementation of high speed vedic multiplier,” International Journal of Computer Applications, vol. 90, no. 16, pp. 6-9, March 2014

[6] D. Govekar and A. Amonkar, "Design and implementation of high speed modified booth multiplier using hybrid adder," 2017 International Conference on Computing Methodologies and Communication (ICCMC), Erode, India, 2017, pp. 138-143, doi: 10.1109/ICCMC.2017.8282661

[7] Chengdong Liang, Lijuan Su, Jinzhao Wu, & Juxia Xiong. (2016). An innovative Booth algorithm. 2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC). doi:10.1109/imcec.2016.7867510

[8] Tariq, A. S., Amin, R., Mondal, M. N. I., & Hossain, M. A. (2016). Faster implementation of Booth’s algorithm using FPGA. 2016 2nd International Conference on Electrical, Computer & Telecommunication Engineering (ICECTE). doi:10.1109/icecte.2016.7879580