MCIM:
AREA EFFICIENT MULTI-CYCLE INTEGER MULTIPLIERS

A Thesis

by

Ahmad Houraniah

Submitted to the
Graduate School of Sciences and Engineering
In Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in
Computer Science

Özyeğin University
January 2023

Copyright © 2023 by Ahmad Houraniah

MCIM:
AREA EFFICIENT MULTI-CYCLE INTEGER MULTIPLIERS

Approved by:

Prof. Dr. Hasan Fatih Uğurdağ, Advisor
Department of Electrical and Electronics Engineering
Özyeğin University

Asst. Prof. Dr. Tacha Serif
Department of Computer Engineering
Yeditepe University

Prof. Dr. Hasan Fehmi Ateş
Department of Electrical and Electronics Engineering
Özyeğin University

Date Approved: 29 December 2022

To my parents for their continuous love and support

ABSTRACT

Fast multipliers with large bit widths can occupy significant silicon area, which, in turn, can be minimized by employing multi-cycle multipliers. This thesis introduces multiple architectures and parameterized Verilog circuit generators for Multi-Cycle Integer Multiplier (MCIM) designs. When implementing an algorithm in hardware or designing a low-end processor, it is possible that less than 1 multiplication is performed per clock cycle on the average. It is also possible that the multiplications per cycle is a fractional number, e.g., 3.5. In such case, we can surely use 4 multipliers, each with a throughput of 1 result per cycle. However, we can instead use 3 such multipliers plus a multiplier with throughput of 1/2. Resource sharing allows a multiplier with a lower throughput to be smaller, hence area savings. Our MCIM designs offer customization in regards to the throughput, latency, and clock frequency. Three MCIM architectures are proposed in this thesis, each with several variations. The architectures are optimized for different speeds and multiplication sizes, maximizing the area reductions that can be achieved for a given application. While the MCIM idea is for a throughput of 1/n, where n is an integer and $n \geq 2$, the designs, their variations, and the results in this thesis are for n=2 and n=3. Furthermore, combinations of our MCIM designs can offer throughputs of 2/3 and 5/6. All proposed designs were automatically synthesized and tested for various bit widths. MCIM designs offer up to 21%, 42%, 32%, 42%, and 63% of area savings for bit widths 8, 16, 32, 64, and 128, with respect to synthesizing the "*" operator with a throughput of 1. Additionally, the proposed designs can also offer power savings under certain conditions.

ÖZETÇE

Büyük bit genişliklerine sahip hızlı çarpıcılar, önemli miktarda silikon alanı kaplayabilir. Bu alanı çoklu saat darbeli (kısaca darbe) çarpıcılar kullanarak en aza indirebiliriz. Bu tezde, yenilikçi ve verimli Çoklu-Darbeli Tamsayı Çarpıcıları (MCIM) çalışılmakta olup, ilgili tasarımların mimarileri ve parametrize Verilog devre üreteçleri tarif edilmektedir. Donanımda bir algoritmayı gerçekleştirirken veya basit bir işlemci tasarlarken, ortalamada darbe başına 1'den daha az çarpma olması olasıdır. Ayrıca darbe başına çarpmaların mesela 3.5 gibi kesirli bir sayı olması da yaygındır. Böyle bir durumda, her biri darbe başına 1 sonuç veren (akış hızı = 1) 4 adet çarpıcı kullanabiliriz. Bunun yerine, bir önceki çarpıcılardan 3 adet ve ek olarak 1/2 akış hızına sahip 1 adet çarpıcı kullanabiliriz. Kaynak paylaşımı, daha düşük akış hızına sahip bir çarpıcının daha küçük olmasına, dolayısıyla alan tasarrufuna olanak tanır. Bizim MCIM tasarımlarımız, akış hızı, gecikme ve saat frekansı açısından özelleştirme sunar. Bu çalışmada, kendi içinde (gerçekleme açısından) türevleri olan 3 mimari önerilmiştir. Bu mimariler, farklı hızlar ve çarpma boyutları için optimize edilmiş olup, belirli bir uygulama için elde edilebilecek alan küçültmelerini en üst düzeye çıkarırlar. Her ne kadar MCIM fikri 1/n'lik bir akış hızına yönelik de olsa (n tamsayı ve $n \geq 2$), bu tezdeki tasarımlarımız, türevleri ve sonuçlar n=2 ve n=3 içindir. Ayrıca, MCIM tasarımlarımızın kombinasyonları, 2/3 ve 5/6 gibi akış hızları da sunabilir. Önerilen tüm tasarımlar, çeşitli bit genişlikleri için otomatik olarak sentezlenmiş ve test edilmiştir. MCIM çarpıcıları, "*" operatörünün 1 akış hızı için sentezlenmesine göre, 8, 16, 32, 64 ve 128 bit genişlikleri için %21, %42, %32, %42 ve %63'e kadar alan tasarrufu sunar. Ek olarak, önerilen tasarımlar belirli koşullar altında güç tasarrufu da sağlayabilmektedir.

ACKNOWLEDGEMENTS

I have had a long and exciting journey that has taken me one step further toward realizing my dream. I have gained invaluable skills throughout my research. This was only achievable due to the guidance and mentorship of my advisor Prof. H. Fatih Ugurdag, to whom I owe much gratitude. I would also like to thank all my instructors who taught me essential academic and life lessons. My gratitude also extends to my friends and family, who supported me in every way possible. And last but not least, I want to thank all the members of the nEMESysLab for their continuous support and help. They have assisted me in more ways than I can count. I have had an exciting and intellectually stimulating experience throughout my degree, and I only hope my next journey can be as good as this one.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

An integer multiplier is an essential building block for various applications across any domain. Large integer multipliers can be quite expensive due to their area complexities. Large integers are used in a wide range of applications that require a high degree of precision. Floating-point operations are unsuitable for some applications because of rounding operations, increased hardware complexity, and increased latency. Floating-point operations are prone to "catastrophic cancellations" [1]. The addition and subtraction of floating-point numbers that greatly vary in magnitude can result in data loss, as shown in equation 1.

$$10.53 + 2 \times 10^{20} - 2 \times 10^{20} = 0 \tag{1}$$

Fixed-point integers are immune to such cases by nature, and therefore they are preferred for various applications. Moreover, fixed-point multipliers consume significantly less area than floating-point multipliers. The latency of floating-point multipliers can be much longer as well. The CUDA programming model and software environment, that NVIDIA develops, recently introduced a 128-bit integer data type intended for applications that require a higher degree of precision than that offered by floating-point numbers [2]. This signifies the importance of large fixed-point integers. However, multipliers can still require significant hardware resources, even when dealing with fixed-point integers. Decreasing the area complexity for integer multipliers can benefit a large set of applications.

Applications can contain multiple multiplications in their data flow paths; due to the area complexity of these multipliers, the same multipliers are used multiple times to minimize the area complexity of the system. However, these multiplications typically need to be computed within a predefined period. Since a conventional multiplier can only be used once in every clock cycle, using a

single multiplier can severely limit the throughput of the design. The throughput of a design can be maintained by using multiple hardware multipliers operating in parallel. The number of multipliers necessary can be dividing the number of multiplications by the period they need to be computed within (in clock cycles). Using this formula, the number of multipliers required is not always an integer. The conventional approach is to round up this value, which causes one of these multipliers to be underutilized. There also exists a large number of applications containing data flow paths that require less than one multiplication every two or more clock cycles. A fully-pipelined multiplier that can accept new inputs in consecutive clock cycles would also remain underutilized in such applications. Such data flow paths can be found in various applications across any field, especially ones that operate under a low bandwidth.

There also exists a large number of applications containing data flow paths that require less than one multiplication every two or more clock cycles. Such a case is found in RISC-V softcore implementations, where area efficiency is vital, even if it comes at the expense of increasing the latency of ALU operations such as multiplication.

This work presents various multi-cycle integer multiplier (MCIM) designs to decrease the area complexity for underutilized multipliers. MCIM designs offer area-efficient designs with a throughput of less than 1, ranging from $0.\overline{3}$ (1 multiplication every 3 clock cycles) to $0.8\overline{3}$ (5 multiplications every 6 clock cycles), decreasing the overall area complexity for a wide variety of applications. MCIM designs target unsigned integer multiplications and can be extended to handle signed integer multiplication.

This thesis is divided into four chapters. Chapter 1 presents the benefits of an MC multiplier and the previous work. Chapter 2 describes and presents the techniques and architectures proposed in this thesis. Chapter 3 describes the implementation methodology, the implementation results, and the implications of those results. Chapter 4 concludes this thesis.

1.1 Integer Multiplication

The conventional approach of multiplying unsigned integers can be seen in figure 1, comprising two steps, partial product generation (PPG) and partial production summation. The PPG is typically done using AND gates, where one integer is shifted and multiplied with each bit of the other integer. The PPG produces multiple variables that must be summed to produce the multiplication result. The partial product summation then handles this using a tree of full adder (FA) and half adder (HA) cells. A conventional ripple-carry adder structure handling the partial product summation can result in a long critical path and an increased area complexity. The optimization of integer multiplications has been a thoroughly studied topic.

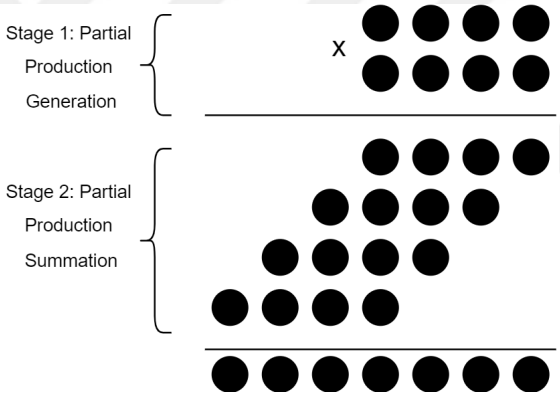

Figure 1: Conventional binary multiplication

Multiplication can be solved using a divide and conquer approach, where any multiplication can be divided into multiple smaller ones, this is often seen as an algorithmic problem, and a great deal of research was done to reduce the algorithmic complexity of multiplications. The same concepts used for reducing the algorithmic complexity can be applied to hardware for area reductions since multiplication can be spread across multiple clock cycles. The conventional schoolbook approach is to divide the multiplication using the distributive property. Any multiplication can be split into multiple smaller multiplications. Equations 2 and 3 show how a single multiplication can be divided into two or three smaller multiplications using

the schoolbook approach, where N is the bit width of the second multiplicand B.

$$Y = A \times B = A \times \{B_1, B_0\} = A \times B_0 + A \times B_1 \times 2^{N/2} \tag{2}$$

$$Y = A \times B = A \times \{B_2, B_1, B_0\} = A \times B_0 + A \times B_1 \times 2^{N/3} + A \times B_2 \times 2^{2N/3} \tag{3}$$

Such an approach can be utilized for area reductions, where a smaller multiplier can be used repetitively to compute several smaller multiplications. Karatsuba multiplication [3] was one of the first algorithms proposed to reduce the complexity of integer multiplication. In a conventional divide and conquer approach, i.e., the schoolbook approach, the complexity for an integer multiplication is $\Theta(n^2)$. Karatsuba [3] proposed an algorithm which decreases the complexity to $\Theta(n^{1.59})$. The Karatsuba algorithm can be described by equations 4,5, 6, and 7 for a single level of division, where the number of smaller multiplications required is reduced from 4 to 3.

$$T_0 = A_0 \times B_0 \tag{4}$$

$$T_1 = A_1 \times B_1 \tag{5}$$

$$T_2 = (A_0 + A_1) \times (B_0 + B_1) \tag{6}$$

$$Y = A \times B = \{A_1, A_0\} \times \{B_1, B_0\} = T_1 \times 2^N + (T_2 - T_1 - T_0) \times 2^{N/2} + T_0 \tag{7}$$

The Karatsuba algorithm can be implemented on hardware for a reduction in area complexity as previously shown in [4, 5] and many others. Single-cycle Karatsuba multipliers only offer area savings for very large multiplications. Such large multiplications have a limited number of applications. This thesis aims to evaluate the area savings that can be offered for MC unsigned integer multiplications. Resource sharing and pipelining can be used to design Karatsuba-based area-efficient MC multipliers. The three smaller multiplications required for the Karatsuba multiplier can be computed using only one smaller multiplier, therefore reducing the area complexity. Since MC Karatsuba multipliers can offer greater area savings than single-cycle Karatsuba multipliers, they become viable options for smaller multiplications that otherwise would be too small for Karatsuba multipliers to be efficient. This increases the number of applications where such a

multiplier can be implemented. Karatsuba multiplication follows a divide and conquer approach, where a large multiplication is split into 3 smaller multiplications, and multiple levels of divisions can be implemented, continuously decreasing the size of the multiplier required. The Karatsuba algorithm allows for an even smaller multiplier than that needed for the schoolbook approach. Resource sharing is a commonly used concept for area reduction, which can be described as reusing an existing circuit instead of having multiple instances of the same circuit. Applying resource sharing is not always straightforward. It can even increase the area complexity if not used correctly. Pipelining is another design technique that increases the maximum frequency without affecting the throughput. Implementing an MC multiplication allows us to implement resource sharing and pipelining to several design parts. The schoolbook and Karatsuba approaches allow for the utilization of a single smaller multiplier.

Integer multiplication is conventionally accomplished by two steps, PPG and summation. Figure 1 represents the multiplication of two 4-bit numbers using the conventional approach. The PPG stage can be split across multiple cycles. Figure 2 represents a pipelined multiplier where the PPG stage is divided across two cycles. This approach can reduce the critical path by half. This also presents an opportunity to reduce the area complexity through resource sharing; the logic responsible for computing stages 1-1 and 1-2 of figure 2 are equivalents. Therefore the same hardware can be used for computing both results. This can reduce the required number of AND gates in the PPG stage by 50%. Resource sharing can also be used to reduce the area complexity of the partial product summation stage.

When implementing resource sharing in the PPG stage, the circuit needs to be reused in consecutive cycles; this means that it will not be able to handle new inputs until the result is computed. This period is referred to as the initiation interval (II). A longer II allows for a greater degree of resource sharing since the multiplication stages can be reused more often. For multiplications larger than 4×4, more than two rows are generated after the PPG stage. Pipelining can also

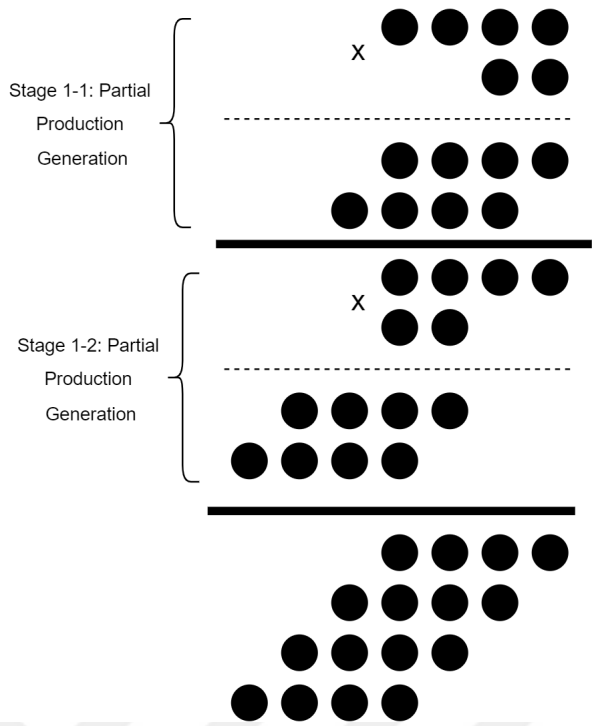

Figure 2: 2-cycle multiplier

be implemented to any multiplication stage simply by placing registers anywhere in the data flow path, which decreases the critical path of a digital circuit without affecting the II.

1.2 Previous Work

Due to the importance of integer multipliers, much research has been done to reduce the area complexity and improve the speed of integer multipliers. Wallace [6] proposed an efficient approach to dealing with integer multiplications that heavily impacted the industry and continues to be applied in modern research. By using carry-save trees for row and column compression, the critical path can be significantly decreased for the summation of more than two numbers. A carry-save tree structure uses FA and HA cells in a parallel fashion, reducing the number of rows to two before the ripple carry adder, thus significantly reducing the critical path. In [7] Dadda presented an improvement over the previously proposed carry save adder tree structure proposed by [6]; this reduced the number of FA and HA cells required for the column and row compression. Ugurdag et al. in [8] proposed a

new and faster carry-save tree structure called row and column compression trees (RoCoCo). Their structure allowed for smaller and faster final adders in integer multipliers. Using these structures, they presented faster multipliers than much of the literature on FPGAs, outperforming Dadda multipliers [7] and the built-in multipliers by Xilinx. Considering how RoCoCo trees can offer an improvement over Wallace [6] and Dadda [7] trees, their architectures were used in this work. Karatsuba et al. In [3] presented an algorithm to reduce the computational complexity of a multiplier using a divide-and-conquer approach. Several works have further reduced the computational complexity. However, these works are focused on algorithmic complexity. Karatsuba multipliers help compute large multiplications, where a CPU cannot compute the large multiplication within one clock cycle. This approach can also be implemented on hardware for a reduction in area complexity, as shown in [4, 5, 9] and many others.

Multi-cycle multiplication has also been studied in the literature; however, most recent works only target FPGA applications. This is due to the more limited resources available on FPGAs. Most of the previous work considers very large multiplications where a very long latency is not considered a limitation. The authors in [4] extensively studied the topic of large integer multiplication, with a focus on FPGA implementation. They proposed several architectures involving Schoolbook multiplication, Comba multiplication, Karatsuba multiplication, Number, and Theoretic Transforms. These architectures are each suited for different applications, each having different characteristics. In their architectures, the latency depends on the bit widths of the multiplicands, allowing for a higher degree of resource sharing depending on the multiplication size, minimizing the number of required DSP slices. Most of their architectures can only handle a single multiplication at a time. The latencies of their designs for a 128×128 multiplication range from 10 to 4400 clock cycles, depending on the specific architecture. Due to the long latencies and high IIs, their work target a different segment of

applications than that of this thesis, focusing on very large multiplications and using very low throughputs. In [5], a design implementing the Karatsuba algorithm was implemented. Due to the recursive nature of the Karatsuba Algorithm, they proposed the use of a "Coprocessor," which was responsible for the sub-functions of the Karatsuba algorithm, such as multiplication, addition, and shifting operations, where the "Coprocessor" has a fixed size. A Block RAM was used to store the intermediate results. Using this approach, they were able to produce a multiplier that could work for different bit widths using the same area resources, the only varying part being the latency required. This is possible since they used DSP slices and Block RAMs to implement the main parts of the design. However, this approach means that the latency can become very long as the multiplication size increases, requiring 120 clock cycles to implement a 128×128 multiplication. Furthermore, this approach implies a long II since the "Coprocessor" is expected to be reused several times for a single computation. This architecture provides an area-efficient approach to computing large multiplications on FPGAs. This architecture further improved upon other FPGA-based MC multipliers proposed in [10, 11], and [12]. Such architectures heavily rely on the usage of DSP slices and Block RAMs, which are expected to increase the speed of an FPGA application while decreasing the slice usage of the design; since ASICs do not have these built-in hardware resources, more optimized solutions can be proposed. Their work also targets a different segment of applications that would not be limited by such a low throughput. The authors in [9] proposed a design based on an MC Karatsuba multiplication; their design achieves significant area reduction for FPGAs, requiring 1/9th of the DSP resources for a conventional 2048×2048 multiplication. For a 2048×2048 multiplication, they achieve a latency of 118 cycles and an II of 9 cycles. Although the design has a relatively low throughput of 1/9, they achieve very significant area savings for large multiplications that would otherwise be too costly in terms of area requirements. However, this design is only feasible for very large multiplications with a more limited number of applications.

A few works have evaluated MC multiplications for ASICs. However, these works offer significant area reductions at the cost of speed/latency. Li et al. presented an area-efficient MC multiplier in [13]. Their designs require an II of N for N×N multiplications and have a latency of N+1. Such a high II and latency can severely limit the design's applications and scalability. In [14], an iterative multiplier for 64×64 was proposed, having an II of 4 cycles and a latency of 10 cycles. They used an internally generated clock using NOT gates. Thus, the clock generation circuitry requires manual modification to work with the system clock. Such an approach can limit the design's capability of being pipelined. Another design was proposed in [15], implementing an MC multiplication circuit based on a modified-Booth encoding. Similar to [14], they also use self-timed clocks. However, the clock generation logic does not require any manual modification to change the operating frequency, which allows it to work with any system clock. Their architecture offered an area reduction of 86.6% in comparison with an array implementation, coming at the cost of an 18.8% reduction in speed. Although this architecture presents significant area savings, a speed reduction can be a major limitation for high-speed applications. Furthermore, relying on a self-timed clock can limit the design's ability to be pipelined. This architecture is meant to be considered as a combinational circuit, and therefore the design can accept inputs in consecutive clock cycles; the system clock, however, would be limited by the maximum frequency of the design. Molina et al. presented a design technique in [16] that allows for the resource sharing of multi-cycle operators. Their technique can be applied as an optimization step at the end of a high-level synthesis process. In their work, they present an example of a 2-cycle array multiplier of size 128×64. This multiplier is implemented using one 2-cycle (2C) multiplier of size 64×64 and two smaller single cycle 32×64 multipliers. Using their techniques, they can apply resource sharing and only need a single 32x64 multiplier, offering an area reduction of 30%. Array multipliers are inefficient by nature. Thus, our designs can offer significantly more area reductions with respect to integer multiplications.

CHAPTER II

PROPOSED DESIGNS

Three architectures for MC multipliers are proposed and implemented in this work. Two of these architectures are based on the schoolbook approach, one of which implements a feedback loop for greater area savings, and the other implements a feed-forward approach for meeting strict timing targets. The third design utilizes the Karatsuba algorithm to implement a greater degree of resource sharing. Two sections will be discussed in this chapter, the first contains a general overview of our methodology for designing area efficient multipliers, and the second will present the proposed designs, including the possible variations for these designs.

2.1 Preliminaries

Integer multiplication can be represented by 3 stages: partial product generation, partial product reduction, and final summation. Any multiplication can be represented as the sum of several smaller multiplications as shown in equations 2 and 3. These smaller multiplications can be computed using the same hardware resource through resource-sharing. This, however, means that the hardware responsible for the smaller multiplications will be used multiple times for each multiplication. Figure 2 represents a multiplier with a pipelined PPG stage. The two stages of this pipeline are equivalent, and therefore they can be computed using the same hardware. This approach reduces both the area requirements and the critical path, where stage 1 of figure 1 requires 16 AND gates, and the critical path consists of 4 AND gates. In contrast, figure 2 requires 8 AND gates, and the critical path consists of 2 AND gates. Multiplications larger than 4×4 result in more rows after the PPG stage. For an 8×8 multiplier with a throughput of 1/2, the PPG stage results in a total of 4 rows for each clock cycle and a total of 8 rows, the number of rows increases linearly as the multiplication size increases.

2.1.1 PPM: Partial Product Multiplier

Resource sharing the PPG stage alone would not provide significant area savings. Even though the number of AND gates required can be reduced by 1/II (50% for 2C designs), for each reduced AND gate we require an additional register. This can be improved by using a narrow compressor (carry-save adder omitting the final addition) after each PPG stage, reducing the number of registers required to store the intermediate values. This approach can reduce the overall area complexity since the narrow compressor is to be resource shared. Furthermore, the depth of the wide compressor would also be reduced, always having a maximum of II*2 (4 for 2C designs) rows to be compressed. This combination of PPG and reduction is what we call a partial product multiplier (PPM). PPMs have a shorter critical path in comparison with regular multipliers since they do not require the final addition stage. The size of the PPM in our architectures depends on two factors: the width of the multiplicands and the II. Any multiplier can be used as a PPM by simply omitting the final addition stage. For this reason, library-based and previously proposed multipliers were used in this work.

RoCoCo [8] provides a row and column compression tree (compressor) that can create fast and efficient integer multipliers. RoCoCo aims at maximizing the bit reductions done compressor, which allows for a smaller final addition. RoCoCo multipliers can be used as PPMs by simply omitting the final addition stage. This architecture can reduce the area complexity and the critical path of the overall design. The compressor inside the RoCoCo-based PPMs operates similarly to the compressor from figure 3, where several of the least significant bits of the second output are reduced to 0. This reduces the required computations in the following stages, i.e., reduces the overall area complexity and the critical path.

DW02_multp is a synthesis-based PPM offered by Synopsys [17]. This PPM is a synthesis-based design and therefore can produce fast and efficient PPMs; however, it also produces signed results. The output size for an M×N multiplication is M+N+2 due to the nature of the design. A sign extension is necessary to add

the results of the DW02_multp. Since the sign of both outputs can vary, the sign extension was implemented using the following three steps.

1. Applying a NOT gate to the most significant bits of all PPMs' outputs (bit position M+N+2, where M and N are the sizes of the multiplicands).

2. Pad the outputs with 1's after the most significant bit (bit positions greater than M+N+2).

3. Sum all constants to reduce the compression size (before synthesis).

Steps 1 and 2 allow us to use DW02_multp for unsigned multiplications, even though DW02_multp produces signed results. Step 3 sums all the constants, which include the sign extension padding bits, reducing the size of the numbers that should be compressed in the next stage, therefore offering further area reductions.

2.1.2 Compressor

Each partial product multiplication produces two results. Based on our MC approach, multiple instances of the PPM will be instantiated, the number of which depends on the specific architecture. Therefore there will be multiple rows to be summed together, ranging from 3-10 rows. For this reason, carry-save adder trees that omit the final adder, i.e., compressors, are used to reduce the critical path. The compressor's size depends on the size of the multiplicands, the number of partial product multiplications required, the type of PPM used, and the specific architecture used. The architectures presented in this work require compressors with depths of 3-10 rows. A compressor for three rows can be described by a series of FAs operating in parallel. Three architectures were used for creating 4:2 compressors: RoCoCo, DW02_tree, and a custom-designed compressor. The architectures that require compressors with more than 4 rows are only efficient for larger multiplications, and for such cases, DW02_tree is the optimal choice which will be discussed in detail section 3.2.2. RoCoCo and the custom compressor are only used for creating 4:2 compressors.

RoCoCo [8] provides a faster approach for row and column compression. Figure 3 represents a compressor generated by RoCoCo, this architecture maximizes the reduction operation without affecting the critical path, and this reduces the size of the final addition since multiple bits in the second output are reduced to zeros.

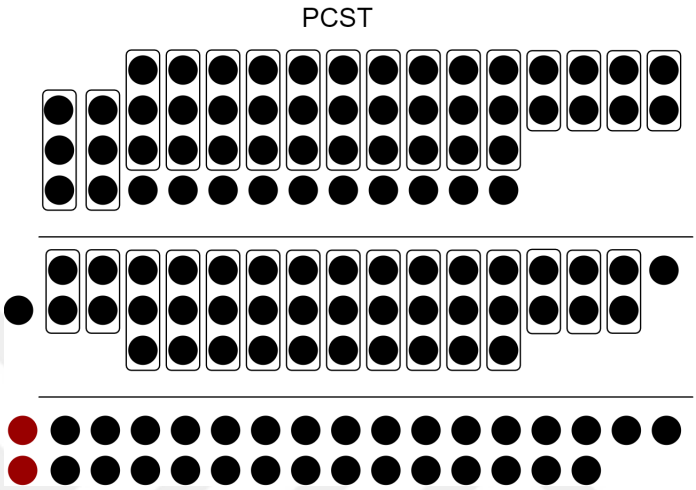

Figure 3: RoCoCo

Similar to DW02_mult, DW02_tree is a synthesis-based compressor offered by Synopsys [17]. Since DW02_tree is a synthesis-based block, the exact compressor can vary based on various synthesis factors, such as timing, size, area, and power. This compressor can be very efficient since the synthesis tool can make more optimizations for the design.

A custom area-efficient compressor was designed for this work as well. This compressor reduces the number of rows to 2 while minimizing the resources required. This compressor is tailored for a feed-forward architecture (which will be discussed in 2.2.2) that uses a DW02_multp PPM. Since this compressor is designed to minimize the hardware resources required, it does not achieve the same bit reductions as RoCoCo. Such an approach can be helpful since the final addition can be spread into multiple cycles, either utilizing a loop and resource sharing or pipelining it. For this reason, this design is capable of outperforming other compressor architectures. This architecture can be described by figure 4

for an 8×8 multiplication, where it offers an area reduction of 1 FA and 7 HAs compared with the RoCoCo-based compressor. However, it requires a larger final.

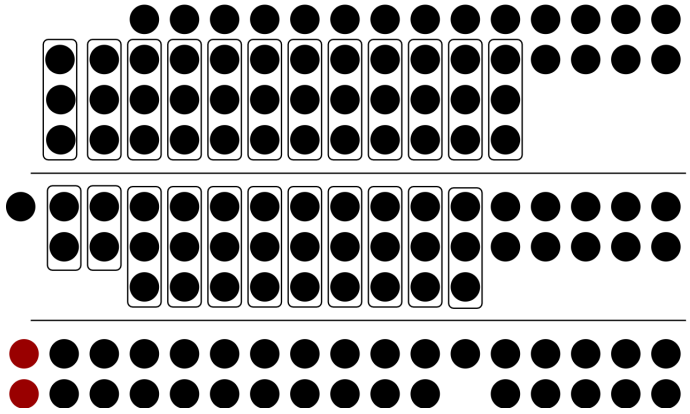

Figure 4: Proposed area efficient compressor 8×8 with sign extention

2.1.3 Final Addition

The final addition stage, which comes after the partial product reduction stage, typically consists of ripple carry adders, which can often be the critical path of a design when no pipelining is implemented. This can also increase the area complexity since the synthesis tool is forced to use larger library cells with greater driving strength and a shorter propagation delay. Furthermore, since the designs target MC multiplications, the final adder will only be utilized for one cycle out of the II (50% for 2C multipliers and 33% for 3C multipliers). This presents another opportunity to implement resource sharing. Various techniques are proposed that can be useful for optimizing the final adder depending on the application.

Figure 5 shows a conventional ripple carry adder (RCA); this approach can have a long critical path if no pipelining is implemented. However, using such an adder allows the design to be easily pipelined since it does not create feedback loops. Pipelining can be implemented by simply placing registers in the path. Pipelining reduces the critical path without requiring additional control logic, and a reduced critical path can also reduce the area complexity.

For 2C multipliers, the final adder is only used 50% of the time, remaining idle for half the time. For this reason, the size of the adder can be reduced by 50%

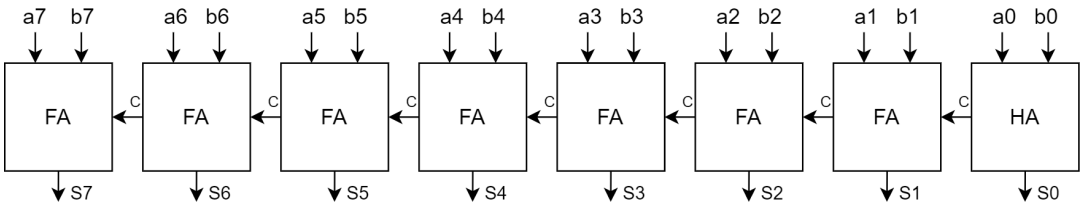

Figure 5: RCA: Ripple carry adder

by applying resource-sharing and using it in 2 consecutive cycles. This approach creates a loop around the adders, which makes pipelining the design complex. For strict timing targets, this approach presents a limitation since the designs are not always able to meet the timing target. For more relaxed timing targets, this method reduces the area complexity because it reduces the size of the final adder. Figure 6 shows an 8-bit 2C resource shared adder (2CA). This method reduces the total number of FAs required from 8 to 4 in this example.

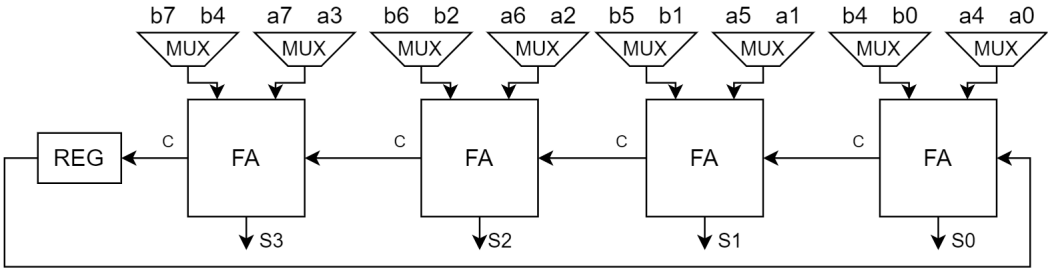

Figure 6: 2CA: 2-cycle resource shared adder

A simple 2C resource shared adder like the one presented in figure 6 reduces the number of FAs required but comes at the cost of an increased critical path. For this reason, we present a further optimized approach that can implement resource sharing while maintaining a short critical path. The critical path of the adder in figure 6 can be reduced by adding an additional register in the middle of the path. This architecture was implemented for an II of 2 cycles. Although this approach reduces the critical path by half, it can create an overlap of variables. Inputs are expected to arrive every two clock cycles, and the latency of the adder would be four cycles; thus, it creates an overlap of variables. This can be solved

by placing an additional register in the path, making the latency of the final adder five cycles. This way, even when variables arrive every two cycles, they do not overlap. The scheduling for an adder of bit width 8 can be seen in table 1, and the circuit implementing this schedule is described in figure 1. Although the II is 2, inputs can still arrive at odd intervals, and since this approach is based on a fixed schedule, this would again create an overlap of variables. This is solved by introducing a 1-cycle delay for when the inputs arrive with odd intervals; this delay is kept until the following odd interval input is received. This, however, means that the latency can change by one cycle depending on the arrival time of previous inputs, meaning the design would have to have a valid out signal rather than a constant latency. Furthermore, this requires an internal state machine and reset signals at the start of the operation. All this results in a more complicated design and thus reduces the gains that can be achieved. The 2-cycle pipelined adder (2CPA) can operate at high frequencies, but it produces a long latency and requires complex control logic.

Table 1: Scheduling of an 8-bit 2-cycle pipelined adder

Clock	FA1	FA2	FA3	FA4
1	A[0]+B[0]	A[1]+B[1]	-	-
2	-	-	A[2]+B[2]	A[3]+B[3]
3	C[0]+D[0]	C[1]+D[1]	-	-
4	A[4]+B[4]	A[5]+B[5]	C[2]+D[2]	C[3]+D[3]
5	x+x	x+x	A[6]+B[6]	A[7]+B[7]

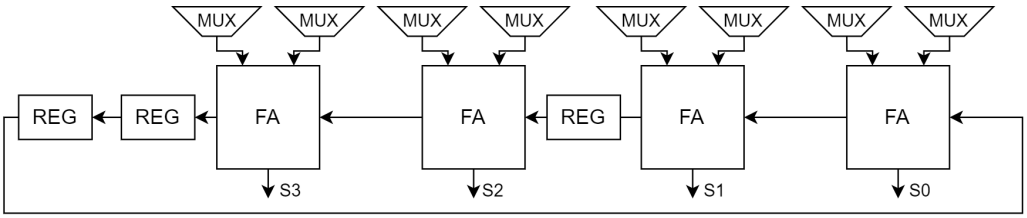

Figure 7: 2CPA: 2-cycle pipelined adder

The concepts we present in this thesis are compatible with any II greater than

2. As the II increases, we can further decrease the resources required for the final addition by implementing a greater degree of resource sharing. The final adder can be resource shared across any number of cycles (up to the bit width of the adder), where a loop can be created around a smaller adder, reusing its hardware for II cycles. Figure 8 represents a 3C adder with a bit width of 6. The total number of FAs required for the final addition can be reduced from $M + N$, where M and N are the bit widths of the multiplicands, to $\lceil (M + N)/Initiation\_Interval \rceil$. For the example given in figure 8, the total number of FAs required is reduced from six to two. However, it requires registers to store the intermediate results, which can limit the area reductions offered as the II increases. Similar to 2CA, any N-cycle resource-shared adder creates a loop around the FAs, limiting the design's ability to be pipelined. Nevertheless, the critical path of this loop decreases as II increases.

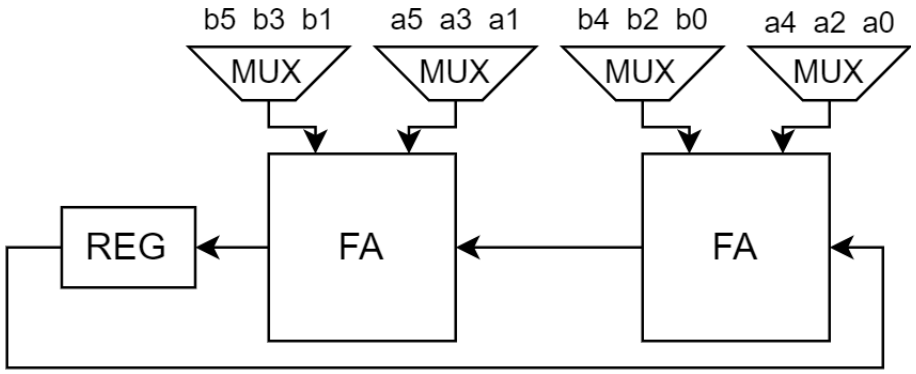

Figure 8: 3CA: 3-cycle resource shared adder

2.1.4 Resource-sharing the Compressor

MC multipliers offer various opportunities for resource sharing since any stage of multiplication can be reused for II times. However, resource sharing can also create feedback loops in the design. Feedback loops limit a design's ability to be pipelined; therefore, there exists a trade-off between the design's ability to be pipelined, which affects the maximum frequency, and the degree of resource sharing to be implemented. A design containing no feedback loops can easily be

pipelined simply by placing registers in the path. This allows a design to meet very strict timing targets at the cost of increased latency. Designs containing feedback loops have a fixed critical path since the feedback loop cannot be pipelined without requiring additional control logic. Therefore, feedback loops can pose a limitation for high-speed applications. Since the area of a standard library cell depends on its driving strength and speed, pipelining allows us to reduce the area usage since the synthesis tool can use smaller cells that have longer propagation delays. Due to this, a feed-forward design can significantly outperform a design that has a feedback loop in its data flow path when the clock target is strict enough, even if the designs implementing a feedback loop can meet timing. A feedback loop makes it possible to implement a greater degree of resource sharing, maximizing the area reductions when dealing with more relaxed timing targets. Therefore, both approaches can be advantageous depending on the target frequency.

2.2 Specifics of Proposed Designs

Three designs are proposed in this work using the concepts previously discussed, each having different variations in terms of the II and the sub-modules used. These designs are optimized for different applications in terms of multiplication bit width and target frequency, maximizing the area reductions that can be achieved.

2.2.1 Feedback Design

Since feedback loops allow for more resource sharing, we propose an architecture where all three stages of the multiplier (PPM, compressor, and final adder) are resource-shared. These stages are all fully utilized for 100% of clock cycles under normal operation (when inputs are arriving back to back). This is achieved by creating a feedback loop around the compressor and the final adder, reducing the depth and width of both modules. The feedback loop in this design contains a 3:2 compressor and a ripple carry adder. For an M×N multiplication, this architecture uses an $Mx(\lceil N/II \rceil)$ PPM, a final adder, and a 3:2 compressor of width M+$\lceil N/II \rceil$. This design is represented by figure 9. The number of FA

cells in the feedback loop, which represents the critical path of the design, can be calculated using equation 8, where M and N are the bit widths of the multiplicands.

$$CriticalPath = 1 + (\lceil M/II \rceil + N), \tag{8}$$

This approach uses the least amount of resources out of our designs. However, the feedback loop can be a limiting factor. This design is based on the schoolbook approach and can be represented by 2 for 2C multiplication. This architecture can

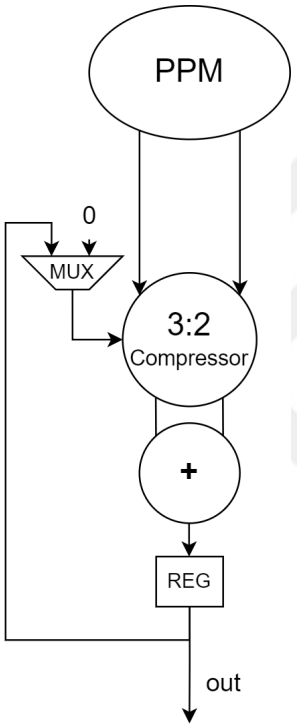

Figure 9: 2-cycle feedback architecture

be extended for any II by simply adding the appropriate number of registers after the loop and decreasing the size of both the compressor and the final adder, where the output would be a concatenation of the current and previous outputs. The II and the area complexity of the three stages, which are the PPM, compressor, and the final adder, have an exponential decay relationship. However, the number of registers that store intermediate results increases linearly. Due to this relationship, we see diminishing returns as the II increases. Therefore, continuously increasing the II after some point would also increase the area complexity of the design. For

this reason, we only examined the cases for an II of 2 and 3. Algorithm 1 describes the behavior of this design for an II of 2 and a 16×16 bit multiplication.

Algorithm 1 *Feedback_2_cycle*

```
1: procedure FEEDBACK_2_CYCLE(A, B, start)
2:     if start then
3:         mo1, mo2 ← PPM(A, B[7 : 0])
4:     else
5:         mo1, mo2 ← PPM(A, B[15 : 8])
6:     co1, co2 ← compressor(mo1, mo2, start? 0 : sum[23 : 8])
7:     sum = co1 + co2
8:     out = {sum, sum_prev[7 : 0]}
```

2.2.2 Feed-forward Design

Multipliers are frequently used with high-frequency applications; for this reason, architectures with short critical paths can be very beneficial. The critical path of a circuit is essential for both area and speed. The maximum frequency is limited by the critical path, and the area complexity is indirectly affected by the critical path, where a design synthesized using its maximum frequency consumes significantly more area than one synthesized for a more relaxed frequency. When operating under strict timing conditions, the synthesis tool instantiates larger library cells that have a shorter delay to meet timing. In contrast, the synthesis tool can use small library cells that have longer propagation delays when dealing with relaxed timing targets. The critical path is caused by the combinational circuit that has to be executed within the same clock cycle. If this combinational logic were to be divided across multiple clock cycles, the critical path would be decreased, and therefore both speed and area would improve. As previously discussed, having a feedback loop in the design limits the design's ability to be pipelined. For this reason, we present another design that contains no feedback loops and therefore is easily pipelined. This architecture has three main steps. Firstly, the partial product multiplications are computed using one module, thus requiring II clock cycles. In the final clock cycle, there remain II*2 results that need to be added, these are first sent to a compressor, and then the result of the compression is sent

to a ripple carry adder. Only the PPM is resource shared in this architecture. Therefore, it does not create any feedback loops. The area savings of this design come from the fact that it requires a smaller PPM (PPG and reduction stages), which contribute a large portion of the overall complexity for an integer multiplier. Pipelining allows us to continuously decrease the critical path at the cost of longer latency. The bit widths of the submodules in this design are interdependent since the type of PPM used affects the size of the compressor. The compressor has at most II*2 rows to be reduced, and several bit-positions contain fewer rows due to the shifting operations. DW02_multp produces signed outputs that require sign extension, while RoCoCo produces unsigned results; thus, the compressors required for these two PPMs are not identical. Algorithm 2 represents the main design concept of the feed-forward design for an II of 2 and multiplication size of 16×16 bits.

Algorithm 2 *Feedforward_2_cycle*

procedure FEEDFORWARD_2_CYCLE(A, B, start)
  if start **then**
    $mo1, mo2 \leftarrow PPM(A, B[7:0])$
  else
    $mo1, mo2 \leftarrow PPM(A, B[15:8])$
  $co1, co2 \leftarrow compressor(mo1 << 8, mo2 << 8, mo1_{prev}, mo2_{prev})$
  $out \leftarrow Finaladder(co1, co2)$

This architecture is ideal to be used with an II of 2. In such a case, the PPM's area can be reduced by around 50% while keeping the control logic simple and not requiring a significant number of registers for storing intermediate results. Since the PPM has two outputs and is used twice, we require a 4:2 compressor, where the inputs would be the first 2 PPM results and the shifted version of the second 2 PPM results, similar to equation 2. This design is represented by figure 2.2.2. This architecture can also be extended for different IIs; however, maintaining a feed-forward approach becomes a limitation. A feed-forward design with an II of 3 would require a 6:2 compressor. This increase in the compressor's size undermines the area reductions gained by the smaller PPM. Furthermore, such a

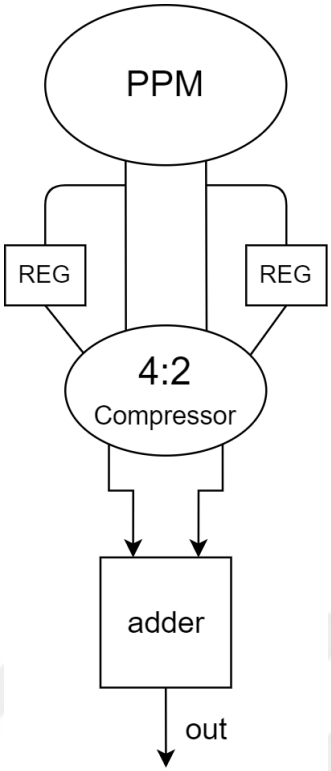

Figure 10: 2-cycle feed-forward archiecture

design requires significantly more registers to store the intermediate results until all PPMs are computed. A better approach for implementing this architecture with an II greater than 2 would be to add a loop around the 4:2 compressor. Figure 11 represents such architecture. However, it still requires more registers to store the intermediate results. Furthermore, this approach also requires a significantly larger compressor, requiring 96% more FAs and HAs for the case of 3C versus 2C II. All this resulted in a higher area requirement when compared to the 2C version. Therefore, a feed-forward design with this approach is only viable for a 2C II.

2.2.3 Karatsuba Multiplication

Karatsuba-based multipliers have great potential for area savings, as shown in much of the previous work. The divide and conquer approach of Karatsuba multiplication allows the PPM to be significantly reduced in size. However, it is vital to maintain a simple control logic to achieve reductions in area complexity. A

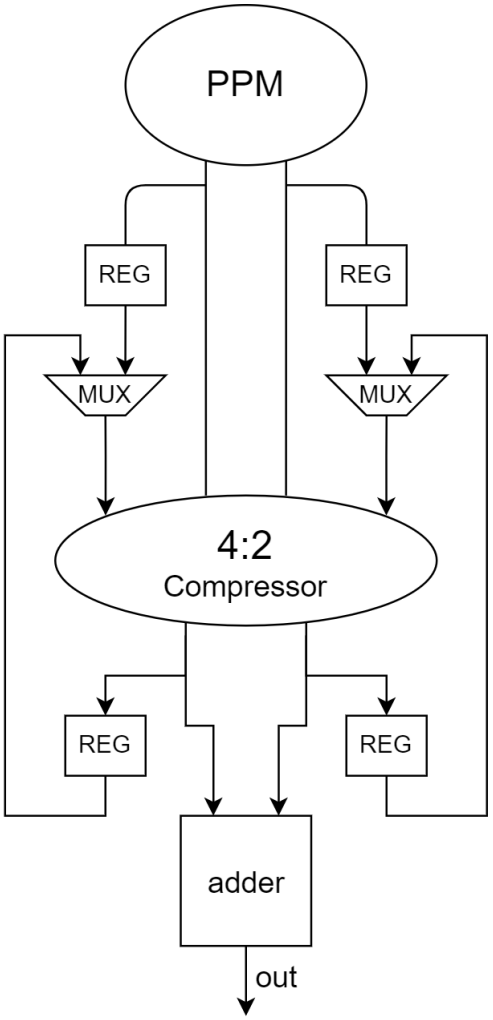

Figure 11: Multi-cycle feed-forward architecture

Karatsuba-based MCIM can be implemented using only one smaller PPM. The PPM is to be used 3 times for each multiplication operation, and therefore such a design has a minimum II of 3. Figure 12 represents a Karatsuba design with an II of 3. Such a design can be implemented with multiple levels of Karatsuba PPMs, which offers greater savings as the multiplication size grows. Karatubsa-based MCIM designs can outperform the previously discussed schoolbook-based designs for large multiplications. Algorithm 3 represents the implementation of the design. Since the Karatsuba algorithm contains subtraction operations, additional logic is required to handle these operations. The subtraction operations are handled by applying two's complement on the PPM results. Applying two's complement to an integer is done by applying a not gate to all the bits and then

incrementing the result. Incrementing this value using a ripple carry adder is inefficient since it will require a long path of FAs. A better approach is to use the compressor to handle the increment operations. There are 4 integers that need to be subtracted when dealing with the Karatsuba algorithm (since each PPM generates two results). Instead of adding 4 "1s" to the compressor, we can add those beforehand and only add a "1" in bit-position 4+N/2 of the compressor's inputs. The PPM can be implemented using DW02_multp, RoCoCo, as well as a combinational Karatsuba-based PPM depicted in 4. Using a Karatsuba-based PPM means several levels of Karatsuba divisions are implemented since even the Karatsuba PPM can utilize other Karatsuba-PPMs in a recursive fashion.

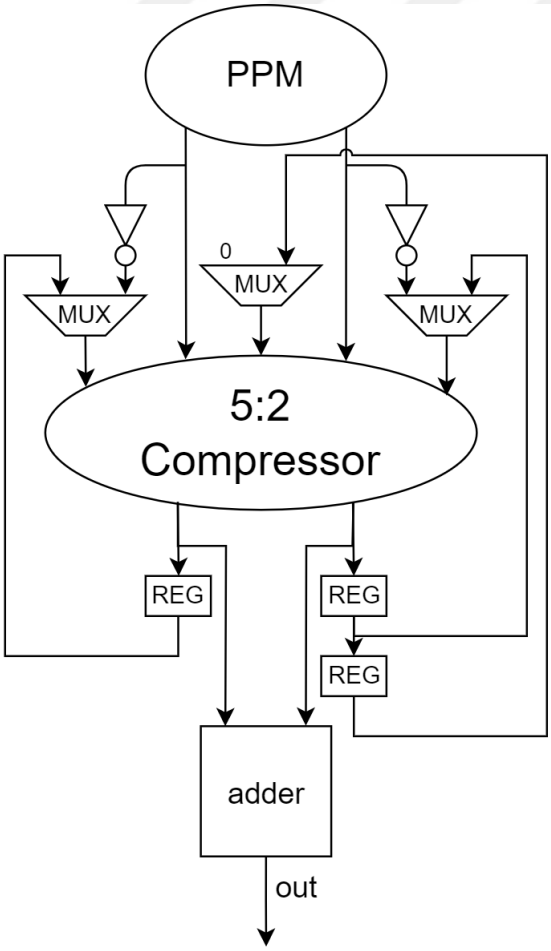

Figure 12: Multi-cycle Karatsuba multiplier

A Karatsuba-based PPM can be created using three smaller PPMs and a 10:2 compressor. Such a PPM can work with all the previously discussed designs.

Algorithm 3 Three Cycle Karatsuba Multiplier

1: **procedure** MC_Karat(A, B, Start)
2: N = length (A)
3: **if** Start **then**
4: $mo1, mo2 \leftarrow PPM(A[N/2-1:0], B[N/2-1:0])$
5: **else if** $Start_{prev}$ **then**
6: $mo1, mo2 \leftarrow PPM(A[N:N/2], B[N:N/2])$
7: **else**
8: $mo1, mo2 \leftarrow PPM(A[N/2-1:N/2], B[N/2-1:0])$
9: $out1, out2 \leftarrow compressor(mo1_{prev2} << N, mo2_{prev2} << N,$
$mo1_{prev2} << N/2, mo2_{prev2} << N/2$
$-mo1_{prev} << N/2, -mo2_{prev} << N/2$
$-mo1 << N/2, -mo2 << N/2$
$mo1, mo2)$
10: $out \leftarrow Finaladder(co1, co2)$

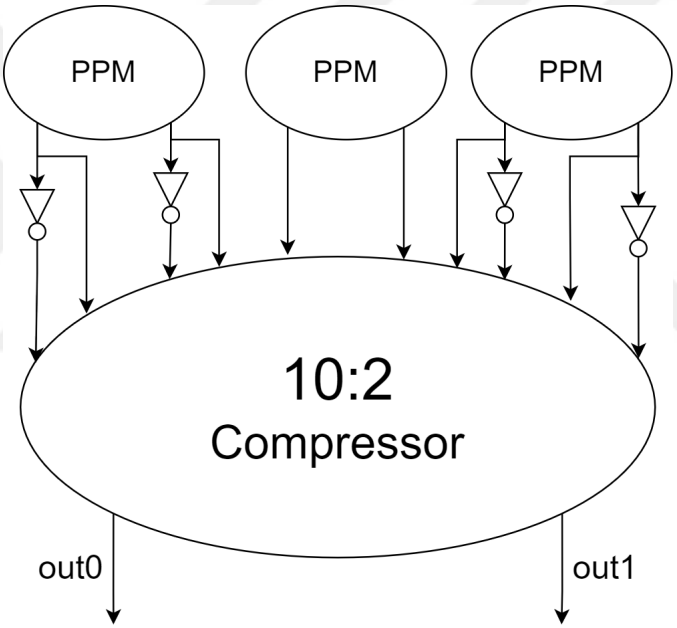

Figure 13: Combinational Karatsuba PPM

However, it only performs well for large multiplications. Algorithm 4 represents the recursive algorithm used to create different levels of combinational Karatsuba multipliers. The subtraction operations are handled by compressors similar to the approach used for MC Karatsuba multipliers. The final addition is not contained in any loop in the design. Thus, RCA and 3CA can be used for the final addition. 3CA significantly reduces the hardware requirements for the final addition, reducing the number of required FAs by up to 66%.

Algorithm 4 Combinational Karatsuba

1: **procedure** $Karat\_sc$(A, B, levels)
2: N = length (A)
3: **if** $Levels > 1$ **then**
4: $m1o1, m1o2 \leftarrow Karat\_sc(A[N/2-1:0], B[N/2-1:0], levels-1)$
5: $m2o1, m2o2 \leftarrow Karat\_sc(A[N-1:N/2], B[N-1:N/2], levels-1)$
6: $m3o1, m3o2 \leftarrow Karat\_sc(A[N/2-1:0] + A[N-1:N/2],$
$B[N/2-1:0] + B[N-1:N/2], levels-1)$
7: **else**
8: $m1o1, m1o2 \leftarrow PPM(A[N/2-1:0], B[N/2-1:0])$
9: $m1o1, m1o2 \leftarrow PPM(A[N:N/2], B[N:N/2])$
10: $m1o1, m1o2 \leftarrow PPM(A[N/2-1:N/2], B[N/2-1:0])$
11: $out1, out2 \leftarrow compressor(m2o1 << N, m2o2 << N,$
$m3o1 << N/2, m3o2 << N/2$
$-m2o1 << N/2, -m2o2 << N/2$
$-m1o1 << N/2, -m1o2 << N/2$
$m1o1, m1o2)$

Most of the proposed MCIM designs compute the final addition stage separately. Omitting the final addition presents from these designs presents a new MC PPM. Karatsuba-based architectures can be further extended for different IIs that are factors of 3. For example, a 6-cycle II can be implemented using a 2C PPM based on the feed-forward architecture previously discussed. An II of 9 can be implemented using a Karatsuba-based PPM as well. As the II increases, more registers and control logic is required in the design; for this reason, a longer II can result in minimal gains, if any; furthermore, it can increase the critical path to an undesirable level.

CHAPTER III

IMPLEMENTATION AND RESULTS

The designs proposed in this work were designed to be modular, where the design can be generated and tested according to various specifications, thus making them easy to integrate for any application. Automation scripts were used for both the implementation of these designs and the result generation. The implementation methodology will be discussed in the first section of this chapter, while the results and findings will be discussed in the second section.

3.1 Implementation Approach

The proposed designs were tested thoroughly, using different bit widths, latencies, and timing targets. This required several steps, including design generation, design simulation, synthesis, applying registers, Netlist simulation, and reporting power. This was automated, where a series of scripts were written to handle these tasks accordingly. This allows for a greater degree of testing required for a complete evaluation of all these designs. The designs are generated using RTL generation scripts written in Python, synthesized using the Synopsys Design Compiler and a TSMC 40 nm technology, and simulated using Icarus Verilog.

3.1.1 Design Generation

In this work, we propose three main architectures, which can have variations in the type of compressors, PPMs, and final adders. Due to this, design generation scripts were used to accommodate these variations easily. All designs are instantiated with a wrapper, which automatically sets the input/output delays and loads. The wrapper applies a register to each of the inputs and outputs of the design except the clock signal. There are three different generators for MCIM designs, one for each architecture. All the generators create both a wrapper and a testbench,

thus allowing for easy and accurate testing for each of the designs. The feedback design's generator takes the size of the multiplicands, the II, the type of PPM (DW02_multp or RoCoCo), and the added pipeline stages as inputs and then generates the design. The feed-forward design has some differences in the design generation parameters since it has a fixed II of 2 and has multiple options for the compressor and final adder to be used. This generator takes the multiplicands' size, the PPM and compressor's type, and the number of pipeline stages as the input parameters. The Karatsuba design also uses a separate generator, which takes the size of the multiplicands, the number of levels of division to be implemented, and the number of pipeline stages. These variations are required to achieve optimal performance since each offers some advantages.

3.1.2 Retiming

Retiming is a technique used to optimize digital circuits by moving flip-flops. It can significantly reduce the critical path and improve the area. Figure 14 demonstrates the capability of retiming, where the critical path is decreased from 25 ns to 10 ns. Furthermore, since strict timing targets require larger library cells, reducing the critical path can also decrease the area complexity.

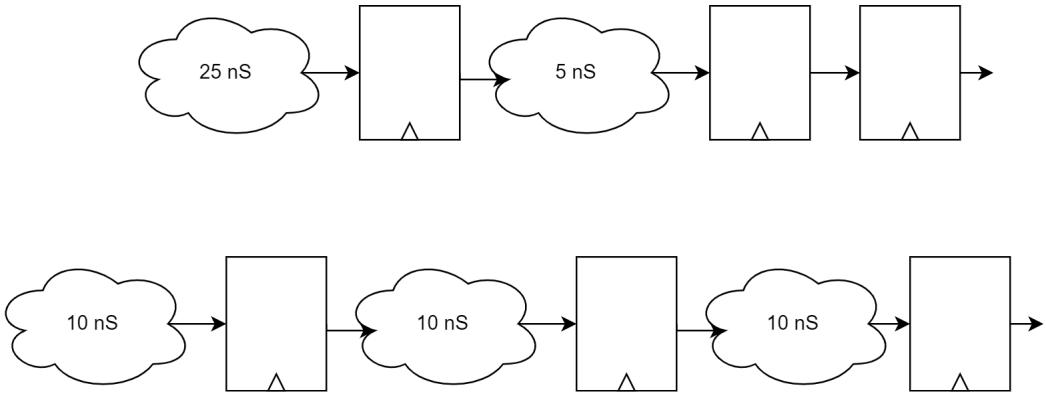

Figure 14: Retiming example

The retiming feature is included by the Synopsys Design Compiler, this feature was utilized to achieve optimized pipelining for any design. Pipelining can be implemented by placing registers in the design; the synthesis tool can freely

move these registers as long as they do not affect a feedback loop. Increasing the pipeline's depth also increases the latency, but it allows us to decrease both the area complexity and the critical path while maintaining the same throughput.

3.1.3 Over-constraining

Synthesis is not a linear computation; it has to handle various constraints to meet timing, optimize area, and minimize power consumption. Over-constraining is another technique used to meet strict timing targets when the synthesis tool initially fails. This is done by further restricting the timing target, allowing the synthesis tool to make decisions that expect a more strict timing target. This can often allow our designs to meet the required timing target even if the initial synthesis attempt was unsuccessfully in meeting timing. The automation scripts attempt over-constraining whenever a design does not meet timing; this is done by reducing the timing target by 5% and reattempting synthesis. Over-constraining is attempted three times before increasing the depth of the pipeline (adding registers to the design and applying retiming).

3.1.4 Timing Target Selection

We need to use the same target timing for a fair comparison of results between any two designs. The area complexity, power consumption, and operating frequency depend on each other. However, in most applications, the operating frequency is a design decision that needs to be preserved. To accurately represent the area savings that can be offered, we first select a timing target and synthesize all the designs according to that target. We set the initial timing target as the maximum frequency that can be achieved by a standard multiplication using the "*" operator (which will be referred to as Star) without adding any additional pipeline stages. Additionally, we test whether over-constraining can achieve a higher maximum frequency. Timing is then increased by 15% and 30% to see how the designs perform in more relaxed timing targets. Additionally, the target is set to 0.31, representing the clock-to-q + setup + hold delay for 1 FA placed between registers,

using the smallest library cell for a FA. This is useful to check how effectively the designs can be retimed to meet very strict timing targets. The designs are also synthesized with additional pipeline stages until the latency reaches the maximum latency required by the other designs since increasing the pipeline stages can reduce the area complexity. Algorithm 5 represents the automated synthesis script responsible for generating these results. Since the designs are tested using a strict

Algorithm 5 Automated Result Generation Script

```
1: procedure SynthesizeDesigns(Target:T)
2:     resultsTable = []
3:     for ii ∈ {T, T * 1.15, T * (1.3), 0.31} do
4:         for cd ∈ designs do
5:             Generate Design cd
6:             Area, Slack, Power, Latency ←GenerateResults(T)
7:             resultsTable.append(cd, T, Slack, Area, Power, Latency)
8:         for cd ∈ designs do
9:             while cd.latency < max(latencies) do
10:                Generate Design cd with latency = max(latencies)
11:                Area, Slack, Power, Latency ←GenerateResults(T)
12:                resultsTable.append(cd, T, Slack, Area, Power, Latency)
```

timing target, the target is not always met from the initial synthesis attempt. The script will first attempt to meet timing using over-constraining. If the design cannot meet timing, a pipeline stage is added. This iteration is repetitively done until either timing is met or the number of pipeline stages exceeds 8. This usually means that the target timing cannot be met even when pipelining is used. 15 represents the iterative script for synthesizing a design.

3.1.5 Simulation and Power Analysis

All architectures are simulated using a set of 200 randomly generated inputs. Self-checking test benches were used, where the output is sampled depending on the latency of the design. Furthermore, both design and netlist simulations were performed.

Power consumption can be an important aspect to consider when designing a digital circuit. All the proposed designs were analyzed for power consumption

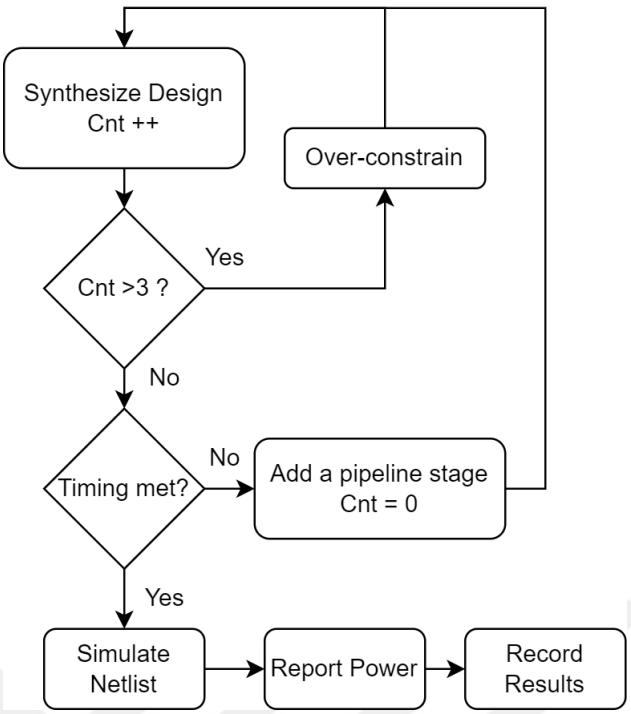

Figure 15: Design synthesis script

using randomly generated inputs. The multiplier receives the inputs consecutively with the minimum number of clock cycles between the inputs (II). This represents a worst-case scenario, where we analyze the power consumption under heavy loads. We performed post-synthesis simulation on all the designs, generating a file that contains the switching activities of all nets and ports. This file is then used by the synthesis tool (Synopsis Design Compiler) to estimate the power consumption of a design, including both dynamic and static power consumption in the estimation. Analyzing power using the post-synthesis netlist simulation can generate more accurate results when compared to the RTL simulation since it contains more accurate switching activities. Power consumption can be affected by several factors. However, there are three main factors to consider, the critical path of the circuit, the length of the MC path (which includes the parts to be resource shared), and the area of the design. Under strict timing conditions, the synthesis tool uses large library cells to meet timing, which consumes more power than smaller library cells having longer delays. Longer MC paths produce more glitches in the design; these

glitches (part of the switching activity) increase power consumption. And lastly, a larger area implies that there are more gates in the design, thus consuming more power since even idle gates contribute to static power consumption.

3.2 Results

All the designs were tested thoroughly in this work; they were synthesized and tested for a wide range of bit widths and timing targets. Each design has advantages and disadvantages, offering significant area saving under the right conditions.

3.2.1 Sub-module Synthesis

The core sub-modules were separately synthesized to better explain the complete designs' synthesis results. None of the final adders implementing spread offered any advantage over other architectures. Therefore these modules were not synthesized separately. When synthesized independently, the results of these modules allow us to make predictions about the optimal modules to use for a specific multiplication. However, since these modules do not produce identical results, complete design testing is still required for an accurate comparison. The PPMs were synthesized separately to compare area complexity and the critical path. The results are presented in table 2. In terms of the critical path, both designs are competitive. The RoCoCo PPM has a shorter critical path for 8×4 multiplications, while the DW02_multp PPM achieved shorter critical paths for both 16×8 and 32×16 multiplications. This can be explained by the fact that the RoCoCo PPM is a fixed circuit created before synthesis. In contrast, the DW02_tree PPM is formed during synthesis, thus allowing the synthesis tool to make further optimizations. Looking at the area requirements, we see that the DW02_multp PPM usually requires less area; this might seem more efficient than the RoCoCo PPM. However, DW02_multp produces signed results, which requires a larger compressor to handle the results. Therefore both designs are viable options, and the overall performance of a complete MC multiplier should be considered.

Table 2: Synthesis results for PPMs

Multiplication Size	Design	Max Frequency	Area Using Max Frequency	Area Using Relaxed Timing Target (10 ns)
8×4	DW02_multp	0.36	205	105
8×4	RoCoCo	0.32	332	113
16×8	DW02_multp	0.45	2636	581
16×8	RoCoCo	0.47	2251	602
32×16	DW02_multp	0.61	7852	2119
32×16	RoCoCo	0.62	10566	2734

The compressors were also synthesized separately to observe the individual performances. Table 3 presents these results. Since the DW02_multp PPM produces signed results while the RoCoCo PPM produces unsigned results, the compressor's exact size depends on the type of PPM used. The Custom compressor presented is only compatible with DW02_multp. For this reason, all the compressors were generated for designs that use a DW02_multp PPM. All these compressors achieved the same maximum frequency, which is expected since they all implement a 4:2 compressor. DW02_tree and the Custom compressor always seem to outperform RoCoCo. However, RoCoCo reduces the bits even further, and the result of RoCoCo reduces several of the least significant bits to 0's for one of the outputs. Thus reducing the resources required for the final addition stage. The Custom compressor and DW02_tree have competitive results, each outperforming the other under some conditions. Therefore all these architectures are viable options that can outperform each other under some conditions.

3.2.2 Synthesis of Complete Designs

Synthesis of the sub-modules alone does not clearly describe the expected results. The outputs of the different compressors and PPMs are not identical. RoCoCo-based modules perform more data reductions than other architectures, reducing

Table 3: Synthesis results for compressors

Multiplication Size	Design	Area using Max Frequency (0.26 ns)	Area using a relaxed timing target (10 ns)
8×8	DW02_tree	511	124
8×8	RoCoCo	565	138
8×8	Custom	487	113
16×16	DW02_multp	876	247
16×16	RoCoCo	1080	274
16×16	Custom	967	226
32×32	DW02_multp	1795	492
32×32	RoCoCo	2149	548
32×32	Custom	1924	454

the sizes of the following modules. Therefore, it affects the area of the other sub-modules. Additionally, the performance of these modules does not change linearly as the multiplication size grows. Each sub-module presented can outperform others depending on the multiplication size and target frequency. All the proposed MCIM designs and variants should be synthesized and compared to perform a thorough comparison. All MCIM designs were synthesized under a wide range of multiplication sizes and various timing targets. See appendix A for a complete set of synthesis results. Since MCIM each target a specific type of application concerning either the multiplication size or the operating frequency, all the designs were synthesized for a wide variety of these cases. However, we present 4 tables in this section representing the different application areas. The feedback design targets relaxed timing targets for any multiplication size. A relaxed timing target can show the true area requirements of each design since a strict timing target would affect the results depending on the critical path. We represent the relaxed timing conditions case by using a timing target of 10 ns. Such a target allowed all the designs to meet timing without requiring any additional pipeline stages. Table 4 presents the synthesis results for a 16×16 multiplication, which

represents a small multiplication. Since the timing target is very relaxed, retiming and over-constraining were not required, therefore, the latency presented in table 4 represents the minimum latency for these designs. All MCIM designs are compared to the standard multiplier that is generated using the "*" operator (Star).

Table 4: Synthesis results for 16×16 bit multiplications under relaxed timing conditions (target = 10 ns)

Design	II	Final Adder	PPM	Comp	Latency	Area	Power (uW)
Star	1	N/A	N/A	N/A	1	1348	**79**
Feedback	2	N/A	DW02	FAs	2	**942**	100
	2	N/A	RoCoCo	FAs	2	960	108
	3	N/A	DW02	FAs	3	**748**	93
	3	N/A	RoCoCo	FAs	3	757	94
Feed-forward	2	RCA	DW02	DW02	2	1096	124
	2	2CA	DW02	DW02	3	1352	159
	2	2CPA	DW02	DW02	6	2102	250
	2	RCA	DW02	RoCoCo	2	1145	127
	2	2CA	DW02	RoCoCo	3	1181	148
	2	2CPA	DW02	RoCoCo	6	2053	251
	2	RCA	DW02	Custom	2	1105	124
	2	2CA	DW02	Custom	3	1420	168
	2	2CPA	DW02	Custom	6	2086	248
	2	RCA	RoCoCo	DW02	2	1051	120
	2	2CA	RoCoCo	DW02	3	1115	138
	2	2CPA	RoCoCo	DW02	6	2120	244
	2	RCA	RoCoCo	RoCoCo	2	1122	122
	2	2CA	RoCoCo	RoCoCo	3	1155	140
	2	2CA	RoCoCo	RoCoCo	6	2129	243

The feedback designs are best suited for applications with a somewhat relaxed timing target; thus, the area complexity of these designs is usually the lowest. For 16×16 multiplications, the feedback designs can offer around 30% area savings while using an II of 2, and up to 44% using an II of 3. However, it comes at the cost of an increase in power consumption. Therefore, a trade-off exists between the area complexity and power consumption for such applications with a relaxed timing target. We can see that the 2CA final adder cannot offer any benefits in terms

of area savings when compared to either the feed-forward design implementing no spread (SCA) or the feedback design. This can be explained by the fact that under very relaxed timing conditions, the area complexity coming from the final adder is lower than the required logic to implement resource sharing, and therefore under these conditions, a 2C spread is not expected to offer any additional area savings. 2CPA is designed to offer area savings for strict timing targets. Therefore, under such relaxed timing conditions, it cannot offer any area savings. This is due to the added complexity in the control logic, which is required for scheduling inputs based on their arrival time, as well as the increased number of registers required for storing intermediate results. Karatsuba designs are not suitable for such small multiplications. The savings gained by the smaller PPM are not very significant and are undermined by the increased size of the compressor; thus, they do offer any significant area savings for such small multiplications. For this reason, they are not included in tables 4 and 5. The proposed designs consume more power than the Star multiplier since they have a longer MC path. A longer MC path increases the number of glitches in a circuit, which increase the dynamic power consumption. Regular Star multiplication employs no resource sharing, and therefore only a part of the circuit is active at the same time since it is fully pipelined (i.e., it can accept new inputs every clock). In our testing, inputs are received every two cycles, and therefore a single cycle multiplier remains idle 50% of the time.

An important aspect of ASICs is their ability to operate at very high frequencies. For this reason, the ability of any multiplier to operate at high frequencies is important. All MCIM designs were tested under high frequencies as well. They were synthesized using a very strict timing target of 0.31 ns, which is equal to the clock-to-q + setup + hold delay for 1 FA placed between registers. Such a strict target shows how efficiently these designs can be pipelined. Table 5 contains the synthesis results of the proposed designs, both retiming and over-constraining were used to meet timing. Feed-forward designs using a 2CA final adder have a longer

critical path than the feedback designs and are unsuitable for high-frequency applications. Designs that use the 2CPA final adders achieve high frequencies as intended. However, the added complexity from the more complicated control logic resulted in a greater area complexity. Therefore, 2CPA is consistently outperformed by a pipelined RCA using the same latency. For such high-frequency applications, the only viable options are fully feed-forward designs because of their ability to be efficiently pipelined without requiring additional control logic. As seen

Table 5: Synthesis results for 16×16 bit multiplications under strict timing conditions (target = 0.31 ns)

Design	II	Final Adder	PPM	Comp	Latency	Area	Timing (ns)	Power (mW)
Star	1	N/A	N/A	N/A	7	5178	0.31	7.52
Feed-forward	2	RCA	RoCoCo	DW02	9	**3963**	0.31	7.13
	2	RCA	DW02	DW02	9	3984	0.31	7.92
	2	RCA	RoCoCo	RoCoCo	7	4065	0.31	**6.57**
	2	RCA	DW02	Custom	9	4065	0.31	7.83
	2	RCA	DW02	RoCoCo	9	4200	0.31	7.98
	2	2CPA	DW02	DW02	11	4971	0.31	10.17
	2	2CPA	DW02	Custom	10	5115	0.31	9.72
	2	2CPA	RoCoCo	RoCoCo	12	5192	0.31	9.58
	2	2CPA	RoCoCo	DW02	12	5202	0.31	10.11
	2	2CPA	DW02	RoCoCo	11	5307	0.31	10.06
	2	2CA	DW02	DW02	5	4394	0.46	4.43
	2	2CA	DW02	RoCoCo	7	4208	0.49	4.18
	2	2CA	DW02	Custom	5	4600	0.48	4.46
	2	2CA	RoCoCo	DW02	5	3255	0.55	2.96
	2	2CA	RoCoCo	RoCoCo	6	3434	0.58	3.05
Feedback	2	N/A	DW02	FAs	4	3712	0.46	4.47
	2	N/A	RoCoCo	FAs	6	3554	0.49	4.34
	3	N/A	DW02	FAs	9	3625	0.46	5.54
	3	N/A	RoCoCo	FAs	9	3096	0.44	4.78

in 5, the feed-forward architectures can offer up to 23% area savings. Additionally, they can offer up to a 13% reduction in power consumption. The Star multiplier is not able to meet timing without pipelining. It requires a minimum pipeline depth of 6 for the designs to meet timing and a depth of 7 to achieve the optimal area. The Star multiplier achieves a similar latency to the proposed designs while

having a greater area complexity, which explains the power reduction offered by MCIM designs for such strict timing targets.

Area reductions are much more significant for large multipliers due to their large area complexities. All MCIM designs were synthesized and tested for large multiplications. All MCIM designs were tested for both strict and relaxed timing targets to evaluate the area savings that can be achieved for different applications. The Karatsuba MCIM designs were specifically designed for large multiplications. Thus, they have an advantage when it comes to large multiplications. Single cycle Karatsuba multipliers only save area for very large multiplications (as large as 512×512) bits based on our testing. Such multiplications have a limited number of applications. Using an MC approach, the size at which Karatsuba multipliers become efficient becomes much smaller, increasing the number of applications. The proposed MC Karatsuba multipliers can save area with respect to Star for any multiplications of size 32×32 and greater. This demonstrates the significant gains in terms of area complexity for Karatsuba MCIM designs. However, when compared to our other MCIM designs, Karastuba-based architectures only offer an advantage for multiplications of size 128x128 and greater. The synthesis results for 128x128 bit multiplications under relaxed timing targets are presented in table 6. Synthesis under relaxed timing conditions shows the minimum area required for any design since having a long critical path does not increase the area complexity.

The 2-level Karatsuba design that uses RCA consumed the least area of these designs, offering area savings of up to 58%. However, Karatsuba designs consume significantly more power than other MCIM designs. This is due to the more complicated control logic, the longer MC path, and the registers required to store the data. Additionally, the data flow path is more complicated, resulting in a much more complex switching activity, i.e., glitches, which results in higher power consumption. The other proposed architectures are still viable options since they offer considerable area savings without consuming much more power than Star and even provide power savings in some cases. The feed-forward architecture

Table 6: Synthesis results for 128×128 bit multiplications under relaxed timing conditions (target = 10 ns)

Design	II	Levels	Final Adder	PPM	Comp	Latency	Area	Power (mW)
Star	1	N/A	N/A	N/A	N/A	1	66319	2.89
Feed-forward	2	N/A	RCA	RoCoCo	DW02	2	59250	3.09
	2	N/A	RCA	RoCoCo	RoCoCo	2	59774	3.11
	2	N/A	RCA	DW02	DW02	2	37042	**2.40**
	2	N/A	RCA	DW02	Custom	2	37790	2.38
	2	N/A	RCA	DW02	RoCoCo	2	38248	2.40
Feedback	2	N/A	N/A	DW02	FAs	2	42913	2.55
	2	N/A	N/A	RoCoCo	FAs	2	57971	3.00
	3	N/A	N/A	DW02	FAs	2	**30217**	2.68
	3	N/A	N/A	RoCoCo	FAs	2	38610	3.10
Karatsuba	3	1	RCA	DW02	DW02	3	31743	23.15
	3	1	3CA	DW02	DW02	5	**27929**	21.63
	3	2	RCA	DW02	DW02	3	27913	21.63
	3	2	3CA	DW02	DW02	5	27929	21.63
	3	3	RCA	DW02	DW02	3	30040	21.96
	3	3	3CA	DW02	DW02	5	29657	21.96

implemented using DW02_multp, and DW02_tree consumes 17% less power while offering an area saving of 44%. Furthermore, this architecture has a lower II and latency than the Karatsuba designs. The 3 cycle feedback designs present a balance between power consumption and area savings, offering an area savings of 54% and a power saving of 4%. The 2C spread can offer area savings for the feed-forward design. However, they are always outperformed by the feedback designs for such timing targets. The designs implementing a pipelined spread are designed for applications where a strict timing target is required and therefore are not expected to offer any gains for such relaxed frequencies. Therefore neither 2CA nor 2CPA are included in table 6.

As the multiplication size grows, so does the critical path. As discussed previously, this would have a negative impact on both the maximum achievable frequency as well as the area. Testing using the same strict timing target used for table 5 (0.31 ns) is only achievable by fully feed-forward designs. Therefore, a better measure of how these designs behave under strict timing conditions is to

test them using their maximum frequency. The true critical path depends on the path that lies inside a loop since this is the path that cannot be pipelined. The designs were synthesized using an impossible target frequency of 0 and using 6 pipeline stages. Feed-forward designs using 2CA have a longer critical path than the feedback designs and consume more area. For this reason, they would not provide additional area savings to other MCIM designs. The feed-forward designs implementing a pipelined spread have a short critical path. However, they consume more area than a retimed feed-forward design using RCA, as seen in table 5. Feed-forward designs implementing no spread can be fully retimed to meet any timing target, similar to the Star multiplier. Table 8 presents the timing performance of the designs that can offer potential area saving but are restricted by their critical path. The long critical path can be a restriction for high-frequency applications and can increase area complexity. The designs implementing a 2C spread have a longer critical path than the feedback designs, which can be seen in table 5. Designs implementing a pipelined spread can achieve much higher frequencies than feedback designs. However, a retimed feed-forward design implementing no spread can achieve the same frequencies while requiring a shorter latency and a lower area complexity. The area and power results are not included in table 7 since these achieve different operating frequencies, which directly affect the area and power consumption.

Table 7: Maximum frequencies for 128×128 bit multiplication

Design	II	Levels	Spread	PPM	Comp	Latency	Timing (ns)
Feedback	2	N/A	N/A	DW02	FAs	8	0.8
	2	N/A	N/A	RoCoCo	FAs	8	0.78
	3	N/A	N/A	DW02	FAs	9	0.76
	3	N/A	N/A	RoCoCo	FAs	9	0.77
Karatsuba	3	1	RCA	DW02	DW02	9	0.54
	3	1	3CA	DW02	DW02	11	0.76
	3	2	RCA	DW02	DW02	9	0.59
	3	2	3CA	DW02	DW02	11	0.74
	3	3	RCA	DW02	DW02	9	0.65
	3	3	3CA	DW02	DW02	11	0.8

Based on the results from table 7, we can see that designs using RCA are faster due to pipelining. The critical path of the designs using 2CA is limited by the feedback loop. Karatsuba designs have a much smaller feedback loop since it only contains the compressor. All these designs are capable of meeting a clock target of 0.8 ns, to test the real performance of these designs, they are all synthesized with a clock target of 0.8 ns, which represents a strict timing target for a large multiplication that can be met by all our designs. The results of synthesis under such timing conditions are presented in table 8. Neither 2CA nor 2CPA provided area savings. Therefore, they are not included in this table. Under

Table 8: Synthesis results for 128×128 bit multiplications under strict timing conditions (target = 0.8)

Design	II	Levels	Final Adder	PPM	Comp	Latency	Area	Power (mW)
Star	1	N/A	N/A	N/A	N/A	4	121634	28.06
Feed-forward	2	N/A	RCA	DW02	DW02	4	67884	30.39
	2	N/A	RCA	DW02	Custom	4	**64778**	29.85
	2	N/A	RCA	RoCoCo	DW02	5	129428	43.83
Feedback	2	N/A	N/A	DW02	FAs	4	92240	55.43
	3	N/A	N/A	DW02	FAs	6	48068	32.58
	3	N/A	N/A	RoCoCo	FAs	6	92657	54.49
Karatsuba	3	1	RCA	DW02	DW02	7	**44888**	35.60
	3	1	3CA	DW02	DW02	6	53607	41.19
	3	2	RCA	DW02	DW02	7	47604	38.04
	3	2	3CA	DW02	DW02	7	53838	43.63
	3	3	RCA	DW02	DW02	7	54884	46.10
	3	3	3CA	DW02	DW02	7	55296	46.21

these circumstances, designs using a RoCoCo Compressor could not meet timing within a reasonable latency. Additionally, designs using a RoCoCo PPM required a much larger area, requiring even more area than a single cycle Star multiplier. This means that both the RoCoCo PPM and compressor do not scale very well for large multiplication, where the area complexity and critical path increase to an undesirable degree. This is because RoCoCo generates gate-level modules that are harder to optimize by the synthesis tool. For such large multiplications, Karatsuba

designs are the best performers, offering area savings of up to 63% but consuming 65% more power. Thus, there again exists a trade-off between power consumption and area savings. The feed-forward design that uses DW02_multp and the Custom compressor offer area savings of 47% while consuming 38% more power. Since the feed-forward achieved the same latency as that of Star, therefore, it can offer the most area savings if there was a latency constraint. The 3-cycle (3C) feedback design that uses DW02_multp achieves area savings of 60% while consuming 51% more power. When MCIM designs offer such savings, even utilizing two 3C designs becomes viable; such a combination can achieve a throughput of 2/3 while offering an area saving of up to 26%. Additionally, a combination of one 2C multiplier and one 3C is also viable, offering a combined throughput of 5/6 and an area saving of up to 9%. Depending on the application's requirements, which include power consumption, area limitation, and throughput requirement, any of these designs or a combination of these designs can be the optimal choice.

A 2C multiplier was presented in [16] that can offer around 30% area reductions. Their work does not specifically target integer multiplications, they present an example of a multi-cycle array multiplier and show that their technique can offer around 30% area reductions. However, array multipliers are inefficient, and better multipliers can be implemented using more conventional approaches. We implemented and synthesized their architecture to produce comparable results. Table 9 presents the synthesis results for the implementation of the work presented in [16] using a clock target of 10 ns. MCIM designs are much more efficient in terms of area complexity than the multiplier presented in [16], providing more than double the area savings.

Table 9: Area savings for 128 × 64 bit multiplications

Design	II	Area	Savings
Array Multiplier	2	63387	–
[16]	2	45103	29%
Feedback	2	21886	65%

3.2.3 Discussions

The proposed MCIM designs can offer significant area savings for various applications. Table 10 contains the area savings offered by the best-performing design under different bit widths and timing targets; this table presents the optimal design under either strict or relaxed timing conditions, irrespective of the II. The Feed-forward design is best suited for strict timing targets, the feedback design is best suited for more relaxed timing targets, and the Karatsuba design is best suited for multiplications of sizes 128 and greater. The area reductions offered by Karatsuba designs increase as the multiplication size increases.

Table 10: Area savings for different bit widths

Design	II	PPM	Comp	Timing (ns)	Latency	Area	Savings
8×8							
Star	1	N/A	N/A	0.31	4	1377	-
Feed-forward	2	RoCoCo	RoCoCo	0.31	5	1088	21.02%
Star	1	N/A	N/A	0.5705	2	738	-
Feedback	2	DW02	FAs	0.5705	4	600	18.68%
16×16							
Star	1	N/A	N/A	0.31	7	5179	-
Feed-forward	2	RoCoCo	DW02	0.31	9	3964	23.46%
Star	1	N/A	N/A	1.001	1	2160	-
Feedback	2	DW02	FAs	1.001	3	1255	41.89%
32×32							
Star	1	N/A	N/A	0.31	10	17790	-
Feed-forward	2	DW02	Custom	0.31	9	13653	23.25%
Star	1	N/A	N/A	1.287	2	6057	-
Feedback	2	DW02	FAs	1.287	3	4093	32.42%
64×64							
Star	1	N/A	N/A	0.4	7	51638	-
Feed-forward	2	DW02	Custom	0.4	7	47496	8.02%
Star	2	N/A	N/A	1.3915	1	22841	-
Feedback	3	DW02	FAs	1.3915	3	10061	55.95%
128×128							
Star	1	N/A	N/A	0.8	4	121634	-
Karatsuba_1	3	DW02	Custom	0.8	7	44888	63.09%
Star	2	N/A	N/A	1.457	1	89165	-
Feedback	3	DW02	FAs	1.457	3	33372	62.57%

MCIM architectures can be categorized using their II. The II affects the throughput of these designs and, therefore, the possible applications. 2C MCIM designs have a throughput of 1/2, i.e., they can compute one multiplication in two cycles, while 3C designs have a throughput of 1/3. Additionally, a combination of two MCIM designs is possible. A combination of 2C and 3C designs can achieve a throughput of 5/6, and a combination of two 3C designs achieves a throughput of 2/3. These combinations can still offer area savings for large multipliers since the combined area of these designs is less area than a conventional Star multiplier. MCIM designs can be used when i multiplications are required within j clock cycles, and i/j is not an integer.

$$x = (multipliers \mod period)/period \tag{9}$$

$$OMCIM = \begin{cases} \text{3CM} & \text{if } 0 < x <= 0.\overline{3} \\ \text{2CM} & \text{if } 0.\overline{3} < x < 0.5 \\ \text{min(2*3CM, Star)} & \text{if } 0.5 < x <= 0.6\overline{6} \\ \text{min(2CM+3CM, Star)} & \text{if } 0.6\overline{6} < x < 0.8\overline{3} \end{cases} \tag{10}$$

$$Multipliers = \lfloor multiplications/period \rfloor * Star + OMCIM \tag{11}$$

Equations 9, 10, and 11 can be used to determine the optimal way for selecting which and how many MCIM designs to use, where multiplications represents the number of required multiplications, period is the number of clock cycles these multiplications need to be computed within, 2CM are 2C multipliers, 3CM are 3C multipliers, and Star is a standard multiplier. Equation 9 calculates the throughput required for an MC multiplier. Equation 10 represents the optimal MC design, which can be comprised of either a single MCIM design or a combination of two. The "min" function in equation 10 represents the minimum area. This is because a combination of two MCIM multipliers does not always offer area savings compared to a single cycle multiplier, and 2C multipliers can outperform 3C multipliers for small multiplications. Since area savings offered by our designs are more significant for larger multiplications, we can still provide area savings when

comparing two MCIM designs to a fully-pipelined multiplier. Equation 11 represents the most area-efficient solution that maintains the same throughput, where "OMCIM" means the optimal MCIM design combination. Another case when MCIM designs can be used is if there is a latency constraint. In table 8, we can see that both the feed-forward design and Star achieve the same latency for a clock target of 0.8. This is because the feed-forward architecture can be pipelined very efficiently. If there is a latency constraint of 4 cycles, and the throughput of these multipliers does not exceed 1/2, multiple instances of the feed-forward design can be used to calculate any number of multiplications, therefore providing a great degree of area savings. A similar case can be seen in the appendix in table 14. Such cases are only seen for large multiplications operating under a high frequency. However, in most cases, using an MCIM design can slightly increase the latency. This is because Star multipliers are somewhat faster than the proposed MCIM designs, which might require more pipeline stages, usually around one or two additional clock cycles.

CHAPTER IV

CONCLUSION

Various applications require fractional multiplications, e.g., 3.5 multiplications per clock cycle (7 multiplications are needed within two clock cycles). The conventional way of dealing with such cases is to utilize complete multipliers for these fractional multiplications. Or in other words, using a multiplier for only 50% of clock cycles. This approach is not optimal since it requires more area than necessary and does not take full advantage of the allocated resources. This work presents a range of multi-cycle unsigned integer multipliers that offer fractional multipliers. These designs provide significant area savings for a variety of applications. MCIM designs are designed to replace fully-pipelined multipliers, i.e., multipliers that can accept inputs in a consecutive clock cycle, for applications where they remain underutilized. Three main architectures are proposed in this work, each having different variations that can increase the area savings provided. The architecture of all our designs relies on the same concepts: separating multiplication into three stages (PPM, compressor, and the final adder) and then applying resource sharing to reduce the area complexity. Multiple architectures are used and tested for each of these stages that rely either on previously published works (RoCoCo [8] and Synopsys DesignWare library [17]), or custom architectures presented in this work. MCIM designs are designed to maximize the area savings that can be provided for any application. Our architectures target specific types of applications concerning the multiplication bit width, target frequency, and required throughput. Under relaxed timing conditions, a longer critical path is not a limitation. Relaxed timing conditions allow for more resource sharing to be implemented, even if it requires feedback loops. The feedback architecture is designed for such applications. Feedback designs are often the optimal choice for

applications running that do not require very high operating frequencies, offering up to 57% area savings. An important advantage of ASIC designs is their ability to run at high frequencies. Feedback loops limit a design's ability to be used for such applications. Therefore, an architecture that does not implement feedback loops was implemented, allowing it to meet very strict timing targets through pipelining. Pipelining can significantly increase the maximum frequency since the critical path can be continuously divided into smaller parts. Feed-forward designs can meet very strict timing targets while maintaining a lower area complexity than conventional multipliers. Multiplications with large bit widths can consume a significant portion of any design's area complexity. Area savings for large multipliers are much more significant since large integer multipliers consume significant silicon area. The third architecture is based on the Karatsuba algorithm, which can significantly reduce the area complexity for large multiplications. These multipliers can offer considerable area savings for large multiplications. Karatsuba MCIM designs can offer the most area savings from our MCIM designs for any multiplication greater than 128×128. The area reductions provided by Karatsuba multipliers increase as the multiplication size grows, offering up to 63% area savings for 128×128 multiplications. Karatsuba multipliers are suitable for low- and high-frequency applications, even though they contain feedback loops. The feedback loop used for Karatuba MCIM designs has a short critical path. Therefore, designs can still meet strict timing targets. However, the feed-forward design will always be faster since it can be continuously pipelined. All these designs offer fractional multipliers since they have a throughput lower than 1. The feedback architectures provide a throughput of 1/2 and 1/3. The feed-forward architectures provide a throughput of 1/2, and the Karatsuba architecture provides a throughput of 1/3. The throughput is determined by the number of multiplications that can be computed within 1 clock cycle. The throughput of the Karatsuba and the feed-forward design can be further decreased. However, we see diminishing returns as the throughput is further reduced. A lower throughput implies that

more control logic and memory elements are required, which undermines the area savings achieved by the smaller sub-modules. MCIM designs can offer up to 21%, 42%, 32%, 42%, and 63% area reductions for multipliers of bit widths 8, 16, 32, 64, and 128, with respect to a standard multiplication using the "*" operator. MCIM designs can be used for applications requiring a multiplier with a throughput between 1/3 and 5/6. Multiple instances of MCIM multipliers can also be used for applications with a latency constraint. Under strict timing conditions, MCIM designs can provide significant area savings while maintaining a latency equal to that of Star. In such cases, multiple MCIM designs can be used, providing great area reductions. All MCIM designs use Verilog generators that allow for various variations to be made regarding the latency, throughput, and submodules to be used, maximizing the area reductions that can be offered. Verilog generators allow MCIM designs to be used as building blocks, allowing them to be easily pipelined and modified according to the application requirements. Overall, these designs can offer significant area savings for unsigned integer multiplications for various applications. Moreover, these designs can be extended to handle unsigned integer multiplication as well. All MCIM designs are tested thoroughly for a wide range of parameters. The designs are synthesized using the Synopsys Design Compiler and a TSMC 40 nm technology.

APPENDIX

SYNTHESIS RESULTS

The complete synthesis results are listed below in tables 11, 12, 13, 14, and 15. The number in the subscript for Karatsuba designs refers to the levels of divisions implemented, and the letter "s" symbolizes that a 3C spread was used (3CA). The feed-forward designs only use an RCA final adder since it always produces optimal results. The tables present the best-performing variation of each architecture for the given timing constraint. Not all designs can meet the most strict timing targets, which is why they are not included in the tables.

Table 11: Synthesis results for 8×8 bit multiplications

Design	II	PPM	Comp	Latency	Area	Timing (ns)	Power (mW)
Feed-forward	2	RoCoCo	RoCoCo	5	1088	0.31	2.01
Star	1	N/A	N/A	4	1377	0.31	1.93
Feedback	3	DW02	FAs	5	517	0.5705	0.67
Feedback	2	DW02	FAs	4	600	0.5705	0.72
Feed-forward	2	DW02	Custom	3	659	0.5705	0.75
Star	1	N/A	N/A	2	738	0.5705	0.45
Karatsuba_1	3	DW02	DW02	5	1319	0.5705	1.02
Star	1	N/A	N/A	1	1630	0.5705	0.3747
Feedback	3	DW02	FAs	5	447	0.656075	0.57
Feedback	2	DW02	FAs	3	484	0.656075	0.54
Feed-forward	2	DW02	DW02	3	531	0.656075	0.60
Star	1	N/A	N/A	2	544	0.656075	0.36
Karatsuba_1	3	DW02	DW02	5	1154	0.656075	0.89
Feedback	3	DW02	FAs	3	411	0.74165	0.46
Feedback	2	DW02	FAs	3	485	0.74165	0.49
Feed-forward	2	DW02	DW02	3	549	0.74165	0.52
Star	1	N/A	N/A	3	607	0.74165	0.39
Karatsuba_{1s}	3	DW02	DW02	4	1134	0.74165	0.91

Table 12: Synthesis results for 16×16 bit multiplications

Design	II	PPM	Comp	Latency	Area	Timing (ns)	Power (mW)
Feed-forward	2	RoCoCo	DW02	9	3964	0.31	7.07
Star	2	N/A	N/A	7	5179	0.31	7.51
Feedback	3	DW02	FAs	4	1553	0.77	1.38
Feed-forward	2	DW02	DW02	3	1687	0.77	1.40
Feedback	2	DW02	FAs	4	1798	0.77	1.44
Star	2	N/A	N/A	2	2240	0.77	0.77
Karatsuba_1	3	DW02	DW02	5	2415	0.77	2.01
Star	1	N/A	N/A	1	5833	0.77	1.01
Feedback	3	DW02	FAs	4	1365	0.8855	1.14
Feedback	2	DW02	FAs	3	1366	0.8855	1.02
Feed-forward	2	DW02	DW02	3	1602	0.8855	1.20
Star	2	N/A	N/A	2	2011	0.8855	0.66
Karatsuba_{1s}	3	DW02	DW02	4	2457	0.8855	2.05
Feedback	2	DW02	FAs	3	1255	1.001	0.89
Feedback	3	DW02	FAs	4	1420	1.001	1.05
Feed-forward	2	DW02	DW02	3	1552	1.001	1.09
Star	2	N/A	N/A	1	2160	1.001	0.47
Karatsuba_{1s}	3	DW02	DW02	4	2282	1.001	1.93

Table 13: Synthesis results for 32×32 bit multiplications

Design	II	PPM	Comp	Latency	Area	Timing (ns)	Power (mW)
Feed-forward	2	DW02	Custom	9	13653	0.31	23.27
Star	1	N/A	N/A	10	17790	0.31	27.97
Feedback	3	DW02	FAs	4	3836	0.99	2.65
Feedback	2	DW02	FAs	3	4848	0.99	2.89
Feed-forward	2	DW02	DW02	4	5733	0.99	3.49
Star	1	N/A	N/A	2	6994	0.99	1.53
Karatsuba_{1s}	3	DW02	DW02	4	5519	0.99	4.58
Star	1	N/A	N/A	1	17123	0.99	2.42
Feedback	3	DW02	FAs	4	3348	1.1385	2.15
Feedback	2	DW02	FAs	3	4445	1.1385	2.45
Feed-forward	2	DW02	DW02	3	4962	1.1385	2.79
Karatsuba_1	3	DW02	DW02	4	5277	1.1385	4.40
Star	1	N/A	N/A	2	6512	1.1385	1.33
Feedback	3	DW02	FAs	3	3596	1.287	2.02
Feedback	2	DW02	FAs	3	4093	1.287	2.11
Feed-forward	2	DW02	DW02	3	4814	1.287	2.52
Karatsuba_1	3	DW02	DW02	4	5066	1.287	4.25
Star	1	N/A	N/A	2	6057	1.287	1.18

Table 14: Synthesis results for 64×64 bit multiplications

Design	II	PPM	Comp	Latency	Area	Timing (ns)	Power (mW)
Feed-forward	2	RoCoCo	RoCoCo	7	47496	0.4	38.71
Star	1	N/A	N/A	7	51638	0.4	39.48
Feedback	3	DW02	FAs	4	10444	1.21	5.83
Karatsuba$_{1s}$	3	DW02	DW02	4	13243	1.21	10.89
Feedback	2	DW02	FAs	3	14399	1.21	5.08
Feed-forward	2	DW02	DW02	3	15297	1.21	5.80
Star	1	N/A	N/A	2	23499	1.21	3.59
Star	1	N/A	N/A	1	51028	1.21	6.00
Feedback	3	DW02	FAs	4	10061	1.3915	5.04
Karatsuba$_1$	3	DW02	DW02	4	12677	1.3915	10.32
Feedback	2	DW02	FAs	3	13389	1.3915	4.33
Feed-forward	2	DW02	DW02	3	14526	1.3915	5.06
Star	1	N/A	N/A	2	22841	1.3915	3.18
Feedback	3	DW02	FAs	4	9897	1.573	4.45
Karatsuba$_1$	3	DW02	DW02	4	12572	1.573	10.07
Feedback	2	N/A	DW02	3	13809	1.573	4.11
Feed-forward	2	RCA	DW02	4	14541	1.573	4.78
Star	1	N/A	N/A	2	20451	1.573	2.74

Table 15: Synthesis results for 128×128 bit multiplications

Design	II	PPM	Comp	Latency	Area	Timing (ns)	Power (mW)
Karatsuba$_1$	3	DW02	DW02	7	44888	0.8	35.60
Feedback	3	DW02	FAs	6	48068	0.8	32.58
Feed-forward	2	DW02	Custom	4	63778	0.8	29.85
Feedback	2	DW02	FAs	4	92240	0.8	55.43
Star	2	N/A	N/A	4	121634	0.8	28.06
Feedback	3	DW02	FAs	4	33372	1.457	14.65
Karatsuba$_{1s}$	3	DW02	DW02	4	35762	1.457	28.00
Feedback	2	DW02	FAs	3	48911	1.457	12.82
Feed-forward	2	DW02	DW02	3	51110	1.457	14.39
Star	2	N/A	N/A	2	89166	1.457	9.57
Star	2	N/A	N/A	1	153767	1.457	15.42
Feedback	3	DW02	FAs	4	32779	1.67555	12.86
Karatsuba$_1$	3	DW02	DW02	4	35091	1.67555	26.17
Feedback	2	DW02	FAs	3	46086	1.67555	10.93
Feed-forward	2	DW02	DW02	3	48378	1.67555	12.40
Star	2	N/A	N/A	2	86143	1.67555	8.37
Feedback	3	DW02	FAs	4	33761	1.8941	12.32
Karatsuba$_{1s}$	3	DW02	DW02	4	34430	1.8941	26.72
Feedback	2	DW02	FAs	3	47185	1.8941	10.54
Feed-forward	2	DW02	DW02	3	48532	1.8941	11.52
Star	2	N/A	N/A	2	73726	1.8941	7.41

REFERENCES

[1] D. Goldberg, "What every computer scientist should know about floating-point arithmetic," *ACM Computing Surveys*, vol. 23, pp. 5–48, 1991.

[2] C. Hoekstra, K. Shukla, and M. Harris, "Implementing high-precision decimal arithmetic with CUDA int128." `https://developer.nvidia.com/blog/implementing-high-precision-decimal-arithmetic-with-cuda-int128` (accessed 14 July 2022).

[3] A. A. Karatsuba and Y. P. Ofman, "Multiplication of many-digital numbers by automatic computers," in *Proc. USSR Academy of Sciences (DAN SSSR)*, pp. 293–294, Russian Academy of Sciences, 1962.

[4] C. Rafferty, M. O'Neill, and N. Hanley, "Evaluation of large integer multiplication methods on hardware," *IEEE Transactions on Computers*, vol. 66, pp. 1369–1382, 2017.

[5] I. San and N. At, "On increasing the computational efficiency of long integer multiplication on FPGA," in *Proc. IEEE Int. Conf. on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pp. 1149–1154, 2012.

[6] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Transactions on Electronic Computers*, vol. EC-13, pp. 14–17, 1964.

[7] L. Dadda, "Some schemes for parallel multipliers," *Alta Frequenza*, vol. 34, pp. 349–356, 1965.

[8] F. Ugurdag, O. Keskin, C. Tunc, F. Temizkan, G. Fici, and S. Dedeoglu, "RoCoCo: Row and column compression for high-performance multiplication on FPGAs," in *Proc. East-West Design & Test Symp. (EWDTS)*, pp. 98–101, 2011.

[9] M. Langhammer and B. Pasca, "Folded integer multiplication for FPGAs," in *Proc. ACM/SIGDA Int. Symp. on Field-Programmable Gate Arrays (FPGA)*, pp. 160–170, 2021.

[10] J. Von Zur Gathen and J. Shokrollahi, "Efficient FPGA-based karatsuba multipliers for polynomials over F_2," in *Proc. Selected Areas in Cryptography (SAC)*, pp. 359–369, Springer, 2005.

[11] S. Gao, D. Al-Khalili, and N. Chabini, "Efficient scheme for implementing large size signed multipliers using multigranular embedded DSP blocks in FPGAs," *Int. Journal of Reconfigurable Computing*, 2009.

[12] F. de Dinechin and B. Pasca, "Large multipliers with fewer DSP blocks," in *Proc. IEEE Int. Conf. on Field Programmable Logic and Applications (FPL)*, pp. 250–255, 2009.

[13] J. Li, Y. Du, and J. Wang, "Design a pocket multi-bit multiplier in FPGA," in *Proc. IEEE Int. Conf. on ASIC (ASICON)*, pp. 275–279, 1996.

[14] M. R. Santoro and M. A. Horowitz, "SPIM: a pipelined 64*64-bit iterative multiplier," *IEEE Journal of Solid-state Circuits*, vol. 24, pp. 487–493, 1989.

[15] M.-C. Shin, S.-H. Kang, and I.-C. Park, "An area-efficient iterative modified-booth multiplier based on self-timed clocking," in *Proc. IEEE Int. Conf. on Computer Design: VLSI in Computers and Processors (ICCD)*, pp. 511–512, 2001.

[16] M. C. Molina, R. Ruiz-Sautua, J. M. Mendias, and R. Hermida, "Area optimization of multi-cycle operators in high-level synthesis," in *Proc. IEEE Design, Automation & Test in Europe Conf. & Exhibition (DATE)*, pp. 1–6, 2007.

[17] Synopsys, "DesignWare Library." `https://www.synopsys.com/dw/buildingblock.php` (accessed: 2021-03-01).

VITA

Name: Ahmad Houraniah

- **Education**
  - **MSc Computer Science**, CGPA: 3.91/4, Özyeğin University, 2022
  - **BSc Electrical and Electronics Engineering**, CGPA: 3.79/4, Özyeğin University, 2020
- **Experience**
  - **Research Assistant at Özyeğin University, September 2019 onwards**

    Working in the field of computer arithmetic, reconfigurable hardware, and high-performance computing.
  - **Teaching Assistant at Özyeğin University, September 2020 onwards**

    Grading papers, conducting lab experiments, and aiding students in their education for the digital design course.
  - **Digital Design Intern at Yongatek, August 2019 to September 2019**

    Worked on and received training for both digital design and verification with UVM.
- **Skills**
  - **Technologies**

    Python, C++20, C, Java, MySQL, AWS, Apache HTTP Server, Verilog, SystemVerilog, VHDL, Microcontrollers, Assembly programming

- **Tools & Software**

  Linux, GitHub, Matlab, Vivado, ISE Design Suite, Cadence tools (Genus, Innovus, Tempus, QRC, Xcelium), QuestaSim, Quartus, LaTex, MS Office

- **Patterns & Practices**

  Object Oriented Programming, Software Design Patterns, Machine Learning, Functional Programming, Complexity Analysis, FPGA Design, ASIC design, RTL Verification

- **Research Experience**

  - **MCIM**

    Fast and efficient multi-cycle integer multipliers for ASICs.

    Tools Used: Synopsys Design Compiler, Python

  - **JugglePAC**

    Fast and efficient pipelined reduction circuit for floating-point number accumulation on FPGAs.

    Tools Used: Xilinx ISE Design Suite, Python

  - **WCSP**

    Memory and Instruction Word Customization of Synthesized Processors for a Given Program

    Tools Used: Xilinx ISE Design Suite, Python

- **Publications**

  - **2CIM: Area-efficient 2-cycle integer multipliers**

    Ahmad Houraniah, H. Fatih Ugurdag, Cengiz Emre Dedeagac,

    Turkish Journal of Electrical Engineering and Computer Sciences, 2022 (Under review)