POLITECNICO DI TORINO

Master's Degree in Electronic Engineering

Master's Degree Thesis

Multi-Precision Multiplier

Supervisors

Prof. Fabrizio RIENTE

Prof. Giovanna TURVANI

Candidate

Gianluca PAUTASSO

October 2022

Abstract

Machine Learning requires an enormous amount of mathematical computation per second. Several architectures have been proposed to match the computation requirements and improve the calculation efficiency. Among these, the Systolic Array accelerators show promising results. These accelerators are composed of several Processing Elements (PEs), arranged on multiple symmetrical lines, which include a Multiply-And-Accumulate module. Specific multi-precision multipliers are increasingly popular since they can execute different precision multiplications and they can be integrated into Systolic Array accelerators.
In this work, a multi-precision multiplier is proposed. The objective of the design is to build up a multiplier formed by combining smaller and equal multipliers. Depending on the number of small multipliers that are used, the size of the final multiplier changes. The mathematical foundation is based on decomposing the operands as the addition of several numbers and then applying the distributive and associative properties of mathematics.
Different designs have been implemented with small multipliers able to execute 4x4 and 8x8 multiplications. A total of six multipliers have been implemented: three multipliers are made with 4x4 small multipliers and their output size is 16,32,64. The other three multipliers are realized with 8x8 small multipliers and their output size is 32,64,128.
The multiplier benefits of hardware re-utilization and shows promising results in terms of area and power consumption. Less energy is consumed and less surface area is required compared to previously developed solutions.
Compared to other structures performing similar functions, reusing multipliers can save about 30% of the area with a penalty of only 8% of the delay.

Keywords: Systolic Array, Machine Learning, Processing Elements, Multiply-And-Accumulate

Acknowledgements

Thanks to the dearest people who have supported me on this journey. Without your presence and unceasing support I would not have made it

Table of Contents

List of Figures

Acronyms

AI
artificial intelligence

AI
Convolutional Neural Network

CPU
Central Processing Unit

GPU
Graphic Processing Unit

PE
Processing Elements

PE
Network On Chip

HLM
High-Level Mutliplier

FIFO
first in, first out

Chapter 1

Introduction

1.1 A world full of Deep Learning

Nowadays, artificial intelligence and its utilization are widespread in every field of science. Deep Learning [1] completely changed the way to identify an object and has opened a new perspective for computer vision science. Autonomous driving[2], fake news detection[3], Natural Language Processing[4], and disease's diagnosis [5] are clear evidence of how critical the impact of these studies is on our lives. Deep Learning takes advantage of the CNNs which allows getting a very high inference accuracy. A Convolutional Neural Network or CNN is a type of artificial neural network, which is widely used for image/object recognition and classification. CNNs are inspired by the architecture of the brain. Just like a neuron in the brain processes and transmits information throughout the body, artificial neurons or nodes in CNNs take inputs, process them, and sends the result as output. In CNNs, there could be multiple hidden layers, which perform feature extraction from the image by doing calculations. This could include convolution, pooling, rectified linear units, and fully connected layers. Unfortunately, the outstanding results of the CNNs, come at the price of high computational complexity.
The huge amount of data and the complexity of the operations to be executed put the strain on the standard CPUs, which turn out to be too slow for these applications. The research then began to look for some architectures that could provide a viable alternative. The GPUs could be a solution[6]. Instead of emphasizing context switching to manage multiple tasks, GPU acceleration emphasizes parallel data processing through a large number of cores. They are more appropriate for executing a great number of operations and they are widely used. GPU workloads scale almost linearly with additional core count, so adding more compute units increase their performance. It has a corresponding increase in die size and power consumption and in some applications, GPUs energy consumption turns out to be

a problem[6].
To meet the needed performance and to increase the efficiency, new architectures have been proposed [7],[8]. The hardware accelerators show impressive improvements from the performance/energy perspective. These architectures are becoming increasingly popular thanks to the results they achieve working with artificial intelligence applications. An accelerator employs a series of strategies to accelerate the calculation and improve the throughput. Some of them consist of reusing the memory and decreasing the precision of the calculation. One popular accelerator architecture is the Systolic Array, that is composed of an array of Processing Elements (PE) which are connected according as a Network On Chip (NoC). Each one of them is composed of a Multiply-And-Accumulate module (MAC), a register, and a FIFO. Each MAC module is composed of an adder and a multi-precision multiplier, which can execute multiplications of different sizes.

1.1.1 The trend in research

For decades all efforts were concentrated to increase the frequency and decrease supply voltage and transistor size and trying to achieve better performances. Nowadays the scenario has changed. The frequency's rise already reached saturation due to technical reasons and, in particular, due to power consumption reasons. The current trend consists of increasing the number of cores/chips in architecture and keeping the same frequency. Using more elements is beneficial because it is possible scaling down the Vdd of a circuit, not increasing the frequency and increasing the throughput. The cost is paid in terms of area which is going to increase to ensure always better performance with lower power consumption. In the field of hardware accelerators, the trend is following the same strategy. To keep low power consumption and to increase performance, the idea is to increase the number of PEs. Having more PEs means achieving better performances.

1.1.2 Aim of the work

A multi precision multiplier is an architecture which can be configured to execute different precision multiplications. It can do matrix products multiplications and it turns out to be useful in Machine Learning's applications
The scope of this project is to study and design a multi-precision multiplier, as alternative to have multiple multipliers of different precision. This can efficiently done by reusing small multiplier to build up a big multiplier.
The main idea behind our multi-precision multiplier is the utilization of the sub-products multiplication. A standard multiplication, indeed, can be solved by using smaller multipliers and by doing a sum of products, each one with the necessary bit extension. This mechanism is very useful in our research because it allows us

to design a multiplier able to execute multi-precision multiplications and, achieve excellent results from the point of view of area utilization. This that can be implemented in the PEs of a hardware accelerator. The data precision needed depends a lot on the application, and having it be configurable, without losing throughput, would be very convenient and efficient . It should achieve improvements from an Area and Power consumption point of view. Disposing of this multiplier would mean being able to increase the number of PEs situated on a hardware accelerator. This would lead to higher throughput and, consequently, better performance. The aim is to find a suitable architecture in the state of art, which can be modified and adapted to this specific application. In the upcoming chapters, there is an analysis of the trend in the multiplier design, followed by a comparison among them. Then there is the explanation of the theoretical background and the behavior of the architecture developed. Finally, an explanation of the results leads to the conclusions.

Chapter 2

Background

In this chapter the knowledge present in the state of art is analyzed.

2.1 The current trend

The world of research is very focused on finding new solutions, that can satisfy the new technologies and can deal with the new challenges that we are facing. In the field of the hardware design of multi-precision multipliers, it is possible to distinguish two main attitudes which are being used:

- Low Power approach: it is to try to keep the throughput of the multiplier fixed and to apply some techniques to reduce the power consumption of the circuit;
- High-Performance approach: it is to try to keep the power consumption of the circuit fixed and to increase the throughput

In the following sections, the two trends are explained and for each some architecture of the stat of art are analyzed.

2.2 Low power architectures

2.2.1 DVS multi-precision multiplier

This multiplier[9] was designed to improve the power consumption of a potential ALU in multimedia and communication applications. The peculiarity of this multiplier is the presence of a Dynamic Voltage/Frequency Scaling unit, shown in figure 2.1[9], which regulates the values of the frequency and the Vdd depending on the Throughput required by the user. Furthermore, the multiplier has a system, based

on a razor flip flop, that implements an error detection strategy to improve the DVS mechanism. Thanks to this mechanism, the circuit is able to save a great amount of energy by regulating the main parameters depending on the requirements.

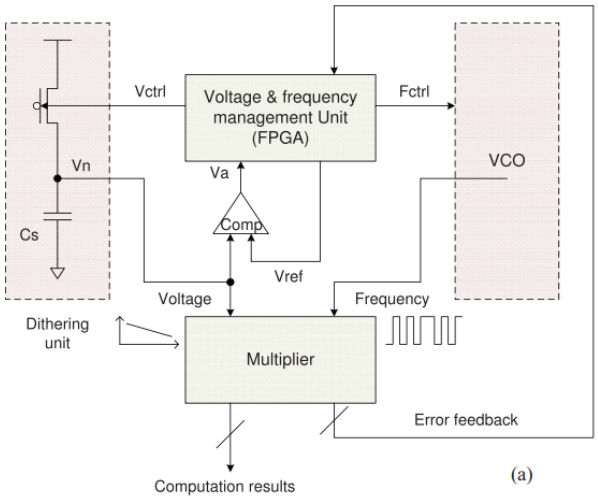


Figure 2.1: Razor Dynamic Voltage/Frequency Scaling unit

The multiplier is based on Booth Radix-4 core multipliers, and it can execute 1/3/9 multiplication 8x8, 1/2/3 multiplications 16x16, and 1 multiplication 32x32. It is composed of 9 units, each one containing an ALU and the DVS logic. The operating frequency is about 50 MHz and the technology used is 0.35 um.

2.2.2 Multi-precision multiplier with truncated partial product

The truncated multiplier[10] is designed to reduce the power consumption and area, to facilitate applications in the field of deep learning and Deep Neural networks. The main idea of the multiplier is that the power consumption in low precision mode can be reduced by turning off the unnecessary circuit parts. The product of two numbers has twice the large bit-width concerning the initial numbers. Often, only the MSB bits of the product are useful for the following computations. Due to this reason, the proposed multiplier is truncated, and thanks to this, the area and the power consumption are reduced while keeping throughput unchanged. Its block diagram is shown in 2.2[10].

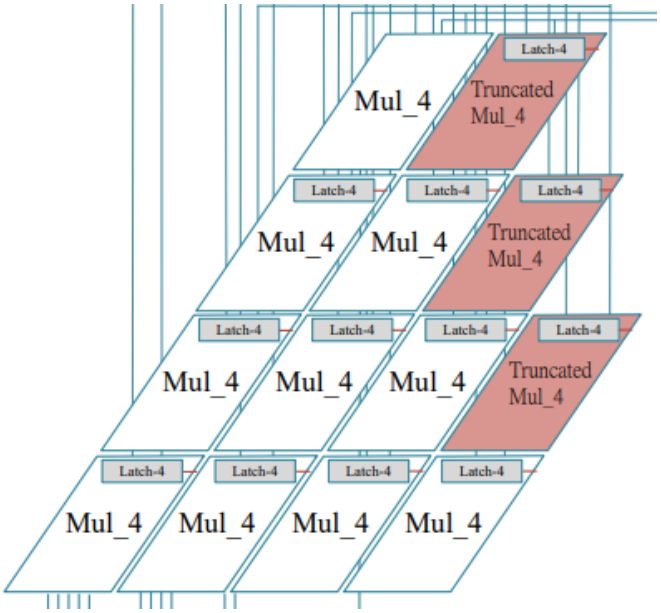


Figure 2.2: Truncated multiplier's block diagram

The multiplier can execute one multiplication of different precision modes: 4,8,12,16 bits. The architecture is composed of 4x4 multipliers and those reds are truncated. The implemented technology is 40 nm and the operating frequency is about 400 MHz.

2.2.3 Low power and configurable Booth

The multiplier[11] has been made for Digital Signal Processing Applications, where. usually, it is necessary for flexibility and low power consumption. There are several techniques to reduce power consumption and increase the operating frequency and they are visible in figure 2.3[11].

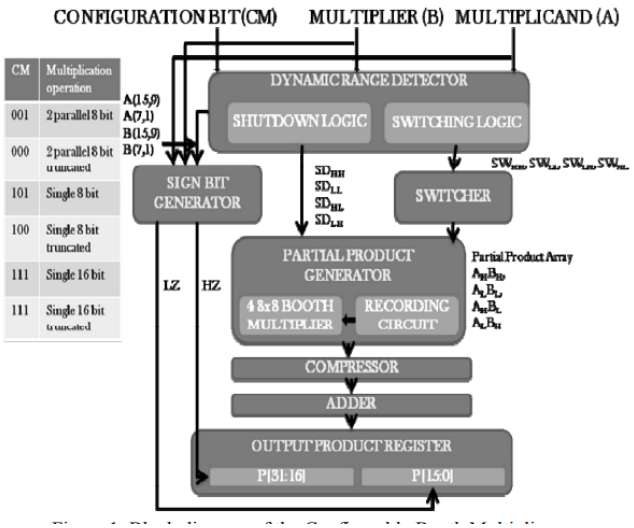


Figure 2.3: Low power and configurable Booth diagram

The "Shutdown logic" detects if any partial products will be zero and shut down the circuit parts which are unnecessary for the computation. The "Switching logic" detects if any Booth encoded product is zero and exchanges the operands it is more convenient for the computation. The "sign bit generator" detects if one of the operands is zero and it, immediately, outputs the result saving energy. The "truncation and compensation circuit" disables the least significant bits computation to further reduce the power consumption.
The multiplier can execute 1 multiplication 16x16 or 2 8x8. The technology implemented is 90 nm and the operating frequency is about 50 MHz.

2.2.4 Approximate multiplier

This multiplier[12] is a multi-precision architecture suitable for digital signal processing applications. Its peculiarity is the ability to work in both full-precision and approximate mode. Depending on the input word length, only the required numbers of the sub-multiplications are activated. The idea consists of subdividing the main multiplications into smaller sub-products executed by different sub-units. The diagram of the multiplier is shown in figure 2.4[12].

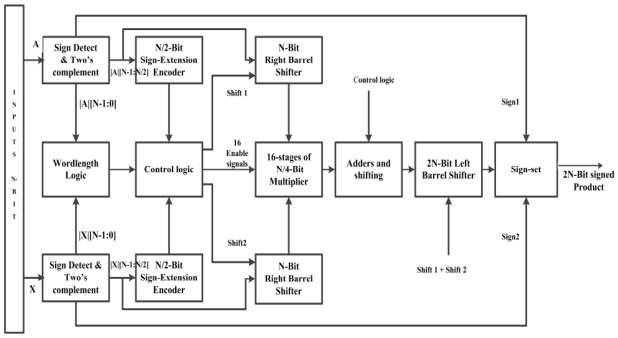


Figure 2.4: Approximate multiplier diagram

If any of the operands has a word length less than 3/4-bit length, the approximate mode is activated. In this case, the operands are truncated and the multiplication is computed using only 1/4 of the PEs.
The multiplier can execute multiplication of 8x8 or 16x16. The technology implemented is 90 nm and the working frequency is about 200 MHz.

2.3 High performance architectures

2.3.1 A flexible multiplier for media processing

This multiplier[13] is supposed to reduce energy consumption while offering flexibility and high performance, for applications in the field of graphic and signal processing. It is based on the recursive technique, which consists of building wider vector elements using narrower ones and adding together the partial results in an iterative way. It is suitable for high-performance applications.
The multiplier can execute double precision, single precision, and multi-precision integer 8x8, 16x16, 32x32. The mechanism exploits the sum of products to compute all multiplications. The block diagram is visible in the figure 2.5[13].

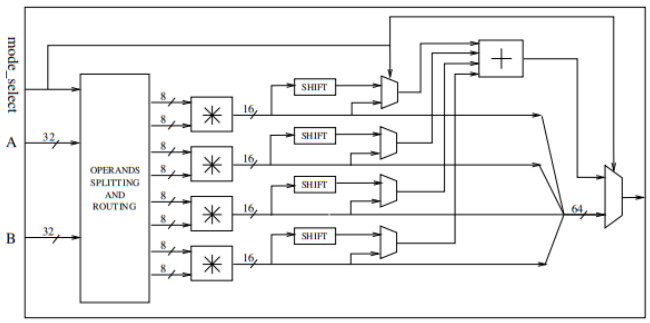


Figure 2.5: Recursive multiplier diagram

This specific architecture is composed of 8-bit integer multipliers, it is, therefore, possible to compute 4 8x8 bit multiplication or combine them with shifters and adders to compute a 16x16 ore single precision or double precision multiplications. The technology implemented is 130 nm and the operating frequency is about 330 MHz.

2.3.2 Multiple-Precision Floating-Point Dot Product Unit

This high-speed multiple-precision multiplier[14] was designed for deep learning applications, in particular object recognition and natural language processing. It computes integer and floating point multiplications, indeed, it divides the multiplication in exponent and mantissa. In the figure 2.6[14] the block diagram is shown.

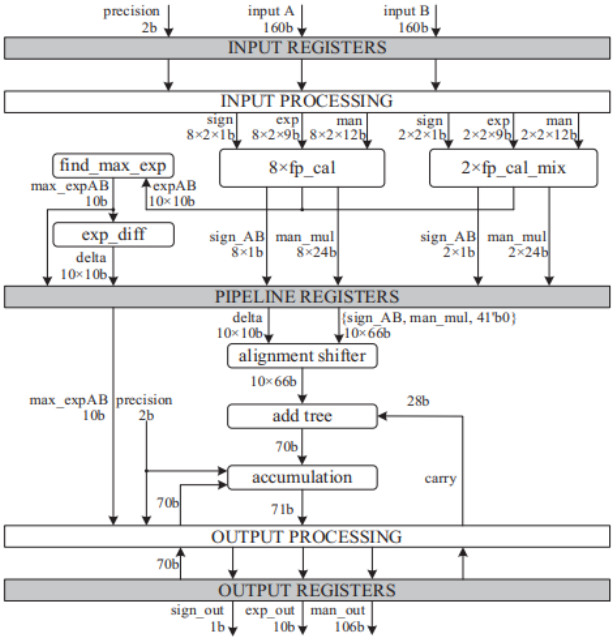


Figure 2.6: CSA multiplier

It is composed of different multipliers that can execute multiplications of 12x12, 12x5,17x15.
The results are then summed by an adder tree, which is structured with a chain of Carry Select Adder. The Alignment shifter aligns the results depending on their exponent.
The technology implemented is 55 nm and the operating frequency is about 450 MHz.

2.3.3 Dual-Mode Double Precision multiplier

This multiplier[15] aims to improve the floating point computation and, at the same time, save area. In order to execute the floating point multiplications, the multiplier decomposes the input numbers into sign, exponent, and mantissa. It is divided into four stages shown in the figure 2.7[15]: in the first one the data from the inputs are extracted and the sign of the multiplication is computed; furthermore, the mantissa multiplications are executed. In the second stage, there are two levels of Dadda Tree. In the third stage, the specific elements calculate and do the appropriate shift of the exponent. In the fourth stage, the exponent and the mantissa are unified and the final output is provided.

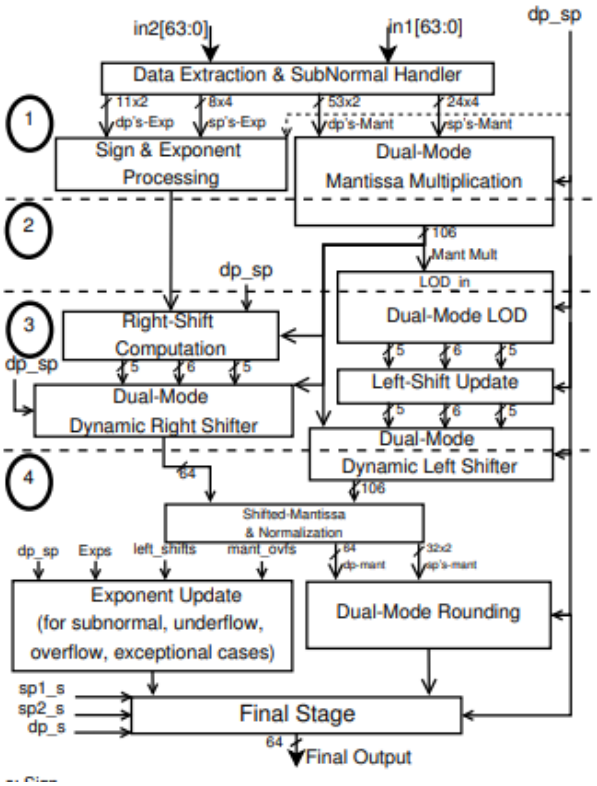


Figure 2.7: Dual-Mode Double Precision multiplier

The multiplier can execute multiplication 1 double precision or 2 multiplications which can be 32x32, 16x16, or 8x8. The technology implemented is 90 nm and the operating frequency is about 1 GHz.

2.3.4 High-performance multi format multiplier

The multiplier described in [16] is made for digital signal applications, and in particular, it has great performance with matrix multiplication. It is composed of several multipliers, distributed according to a hierarchical scheme. The strategy employed is based on the sum of products, which consists of dividing the numbers to be multiplied, into smaller parts. The multiplier is therefore composed of smaller multipliers, shifters, and adders used to compose the final result. Thanks to its hierarchic shape, the multiplier shows great performances in terms of latency, and also it can save a great amount of area thanks to the hardware re-utilization. Its block diagram is shown in the figure 2.8[16].

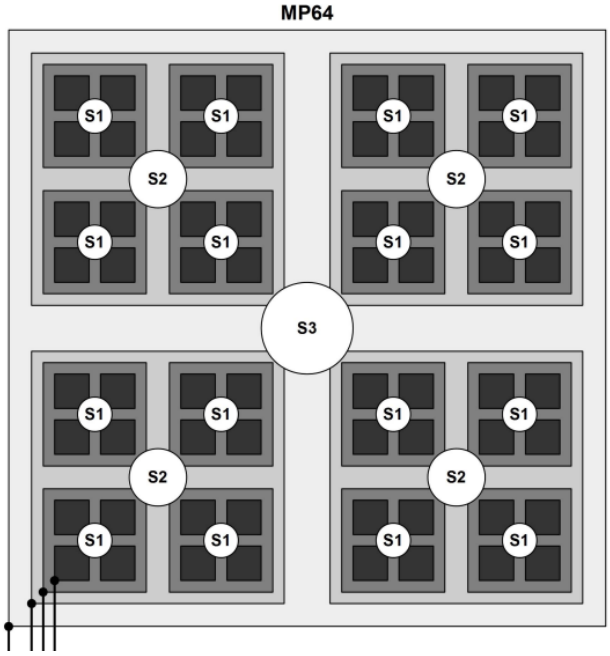


Figure 2.8: High-performance multi format multiplier

The multiplier can execute multiplications 8x8, 16x16, 32x32, and 64x64. It can also do matrix multiplications exploiting the smaller multipliers. The technology implemented is 28 nm and the operating frequency is about 660 MHz.

2.4 The choice

In our specific project, the ideal architecture should be able to execute parallel and multi-precision multiplications. It should be able to reach high performance (deep learning application) and it should be also able to reduce the area and the power consumption.
The architectures analyzed are the best available in the state of art. All of them have specific characteristics and potentialities. The low-power architectures show very promising results in terms of power consumption, but they don't have enough granularity in the multiplication's precision. Furthermore, they need the usage of additional units to implement all the mechanisms to save energy. This represents an increment in the Area of the multiplier and, consequently, in the PE's area.
The High-performance architectures are better in terms of latency and granularity. All of them are able to execute multi-precision multiplications and, in some cases, they are able to execute also floating point operations. The limitation of some is that can't execute parallel multiplications. In addition, executing floating point

operation requires specific units, due to the division into mantissa and exponent, and this is an increment in terms of Area. The multiplier, described in [16], is an architecture able to compute multiplications of different precision and execute parallel multiplications. It can reach good performance, in terms of operating frequency, and thanks to its structure, it can lead to a reduction in terms of Area and power consumption.

The high-performance multi-precision multiplier, described in [16], exploits the calculation of products of parts of the numbers, to calculate the multiplication. This method is very popular and it is used to compute column multiplications. This mechanism draws inspiration from some popular implementations:

- Vedic multiplier[17]: The Vedic multiplier implements the "Vertical and Crosswise" algorithm, shown in figure2.9[17]. The multiplication is decomposed into 4x4 multiplications. The result of the first multiplication is used with the following partial product to generate the 16x16 product. The result is that any multiplications can be executed utilizing 4x4 multipliers.

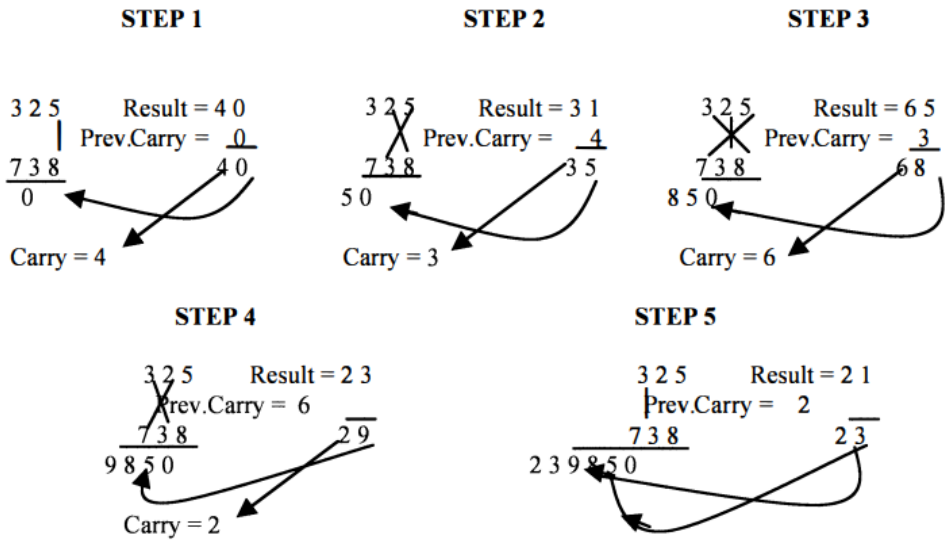


Figure 2.9: "Vertical and Crosswise" algorithm

In the picture is possible to see an example of this algorithm: The digits on the two ends of the line are multiplied and the result is added with the previous carry. When there are more lines in one step, all the results are added to the previous carry. The least significant digit of the number thus obtained acts as one of the result digits and the rest act as the carry for the next step.
This mechanism can be extended to the binary case using the AND gate. The

final architecture is efficient in speed and area. It is also flexible and can be extended to a larger case like 32,64 bits.

- Karatsuba multiplier [18]: The Karatsuba multiplier decomposes a multiplication NxN into 3 multiplications N/2xN/2 and some additions and subtractions. It is based on the following algorithm:
  Starting from 2 numbers x and y, it is possible to write them in the following way:

$$x = a * 10^m + b \tag{2.1}$$

$$y = c * 10^m + d \tag{2.2}$$

  Now the final multiplication can be computed as:

$$x*y = (a*10^m+b)*(c*10^m+d) = a*c*10^{2m}+(a*d+b*c)*10^m+b*d \tag{2.3}$$

Both these two approaches aim to simplify the procedure to execute a multiplication. They use a system of adders, shifters, and sub-multiplications to divide the initial operation into smaller ones. The limit of both these strategies is that "they can't be used if it is necessary to preserve all of the n-bit operand products" [16]. This means that, computing a specific multiplication, doesn't allow to compute a bigger one, exploiting the intermediate results. Instead, the mechanism based on the sub-products multiplications can be applied recursively, and, by simply using adder and shifters, it is possible to compute larger multiplications.

Chapter 3

Architecture

3.1 Subproducts multiplication

By combining the results of the "small" multipliers with adders and shifters, and by doing it recursively, it is possible to compute the results of larger size multiplications.

In the state of art, several solutions have been discussed. In [16] it is discussed the possibility to decompose a standard multiplication of N-bit into 4 sub-multiplication of N/2-bit and one addition.
This method consists of dividing both multiplication operands, A and B, into two parts, high and low, Ah, Bh, Al, Bl. Multiplying each part of the first operand by each part of the second one, and then summing all products, it is possible to get the result of the initial multiplication X = A * B.
In order to get the four distinct numbers (Ah, Al, Bh, Bl), starting from the main ones, it is necessary to shift the High part of n bits. Let's assume the presence of two operands, A and B, which are binary numbers of 2N-bit. Dividing them into the high and low parts and doing the appropriate shift, the following scenario is obtained:

$$A = 2^n * AH + AL \tag{3.1}$$

$$B = 2^n * BH + BL \tag{3.2}$$

Now it is possible to execute the multiplication in the following way:

$$X = AB = (2^n AH + AL)(2^n BH + BL) = \tag{3.3}$$

$$2^{2n} AHBH + 2^n AHBL + 2^n LBH + ALBL \tag{3.4}$$

The 4 sub-products, which are 2N-bit numbers, need to be extended by implementing shifters to fit the 4N-bit addition. After this simple mathematical elaboration,

instead of doing multiplication of 2n-bit, the multiplication can be solved by doing 4 multiplication of n bit and one final addition of 4N bit. This lead to the situation represented in the figure 3.1.

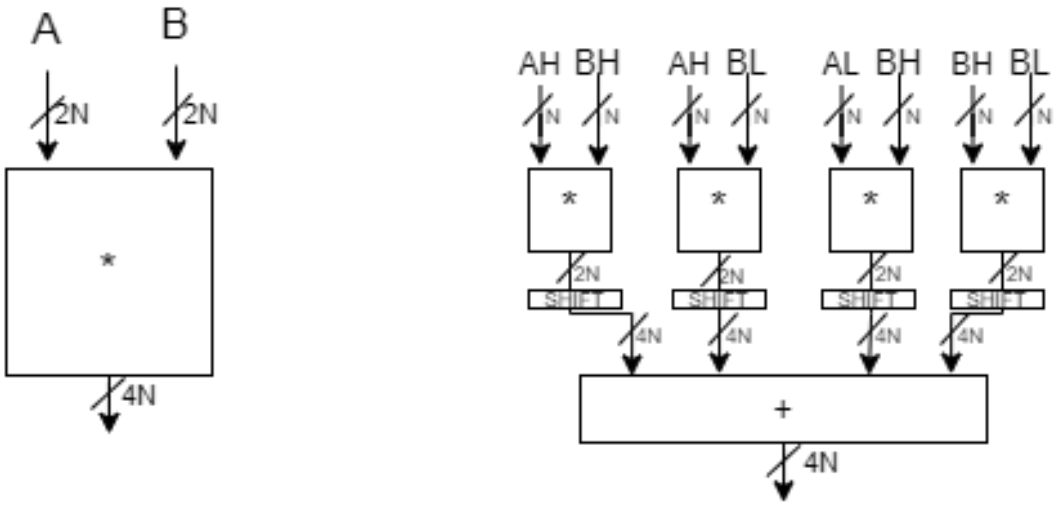


Figure 3.1: Sub-products multiplication

An architecture based on this mechanism can, therefore, be used to solve both parallel multiplications of N-bit or a larger one of 2N-bit. This feature has been reached by utilizing "smaller" multipliers, an adder, and shifters.
The multiplier implements all these components to execute different precision multiplications in a efficient way. The smaller multiplier are reused to compute larger size multiplications according to the hardware reuse strategy.

3.2 Extension to larger multiplications

The decomposition explained above can be easily extended to larger multiplications, by applying recursively the algorithm just explained.
The mathematical rule tells that it is possible to decompose a multiplication of N bit into 4 of N/2. Applying the same strategy to a multiplication of 4N bit, we can reach an equivalent situation. The main multiplication of 4N can be decomposed and executed by doing 4 four sub-products of 2N bit and one addition of 4N. By further applying it, each multiplication of 2N bit can be decomposed into smaller multiplication of N bit. In the end, the result is shown in the following equation:
The operands:

$$A = 2^{3n}Ahh + 2^{2n}Ahl + 2^{n}Alh + All \tag{3.5}$$

$$B = 2^{3n}Bhh + 2^{2n}Bhl + 2^{n}Blh + Bll \tag{3.6}$$

The multiplications become:

$$\begin{aligned} X = AB = (2^{3n}Ahh + 2^{2n}Ahl + 2^{n}Alh + All) * (2^{3n}Bhh + \\ + 2^{2n}Bhl + 2^{n}Blh + Bll) \end{aligned} \tag{3.7}$$

Which can be rewritten in the following way:

$$\begin{aligned} X = & 2^{6n}AhhBhl + 2^{5n}AhlBhh + 2^{4n}AlhBhh + 2^{3n}AllBhh + \\ & 2^{5n}AhhBhl + 2^{4n}AhlBhl + 2^{3n}AlhBhl + 2^{2n}AllBhl + \\ & 2^{4n}AhhBlh + 2^{3n}AhlBlh + 2^{2n}AlhBlh + 2^{2n}AllBlh + \\ & 2^{3n}AhhBll + 2^{2n}AhlBll + 2^{n}AlhBll + AllBll \end{aligned} \tag{3.8}$$

The initial multiplication of 4N bit can be solved by doing 16 multiplications of N bit each one and by doing 4 additions of 2N bit and, one addition of 4N bit.

Once again the same mechanism can be applied to a bigger multiplication of 8N bit. The initial multiplication of 8N bit can be performed by doing 64 multiplications of N bit, 16 additions of 2N bit, 4 additions of 4N bit, and one addition of 8N bit.
The result of this method is the possibility to compute large multiplication starting from smaller ones. This is an important feature for the multiplier, which can implement hardware re-utilization to execute multi-precision multiplications. Indeed, using this technique, the same multiplier can compute the following multiplications NxN, 2Nx2N, 4Nx4N, and 8Nx8N. The NxN precision is executed by the core multipliers and the other precision can be reached by a chain of adders and shifters.

3.3 Unsigned and signed multiplications

This section discussed how the architecture deals with signed and unsigned numbers and which is the strategy to preserve the result's sign if it is necessary.

3.4 Unsigned

The Sub-products strategy discussed extensively in the previous section, divides the original number of 2N-bit into two sub-numbers. The two sub-numbers are binary numbers, each one of N bit and they have to be processed by the multiplier.

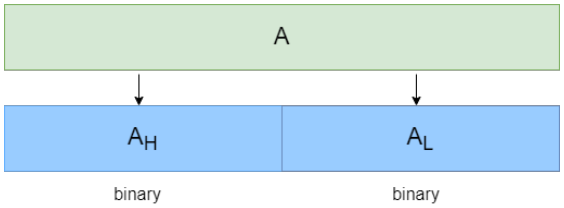


Figure 3.2: unsigned number decomposition

From the mathematical point of view, the situation can be generalized and expressed as follows:

$$AH = a_{n-1} * 2^{n-i} \tag{3.9}$$

$$AL = \sum_{i=0}^{n-2} ai * 2^{i} \tag{3.10}$$

$$A = AH + AL \tag{3.11}$$

To solve unsigned operations, a standard multiplier able to do multiplication NxN is implemented.
The multiplier used to implement the small multipliers is a standard multiplier, described in a high-level language(figure3.3).

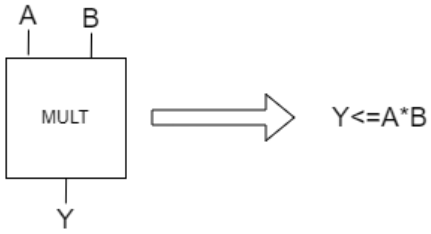


Figure 3.3: High-level multiplier

It does a multiplication NxN and the result is 2N-bit.
After the sub-products have been computed, it is necessary to extend them to a larger size. Before doing the addition, indeed, it is necessary to have numbers with the same size aligned.

3.4.1 Unsigned extension

In the unsigned case, each number extension can be easily done by adding zeros to the number. This is possible due to the absence of the sign. An unsigned number can be always considered as a binary number and, consequently, all sub-products will be unsigned. Let's consider a standard scenario, in which a multiplication of 2Nx2N has to be solved by using the sub-products mechanism. After having obtained the 2N-bit sub-products, the numbers have to be uniform to the same size. The situation is shown in the following picture:

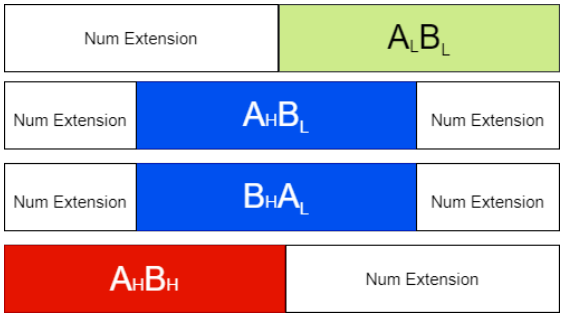


Figure 3.4: Unsigned addition

Each sub-product has been aligned and extended to allow the execution of the addition.

3.5 Signed

The goal and the field of application of the project require a multiplier for signed operations.
Dealing with signed numbers is different and requires more attention. In the previous case, each sub-numbers can be considered as a binary number and each sub-product has no sign. Therefore, the extensions and following steps in the execution were always the same.
Let's analyze the signed multiplication starting from the mathematical representation of a signed number:
In the signed case, considering a number A of 2n-bit and considering its decomposition into 2 sub-numbers of N-bit, the sign is situated in the higher part.

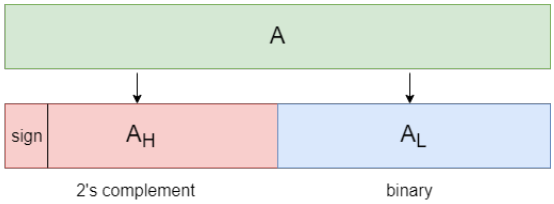


Figure 3.5: 2's complement number decomposition

The sub-number, which is the high part of the number, can be considered as a 2's complement number, while the lower part has no sign and can be considered as a binary number, as in the unsigned case. From the mathematical point of view, the situation can be generalized and expressed as follows:

$$AH = -a_{n-1} * 2^{n-i} \tag{3.12}$$

$$AL = \sum_{i=0}^{n-2} ai * 2^{i} \tag{3.13}$$

$$A = AH + AL \tag{3.14}$$

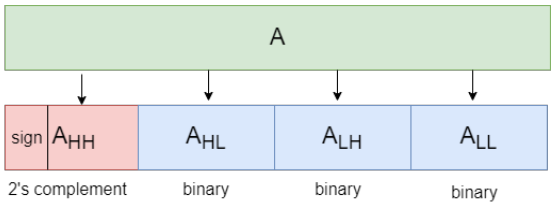
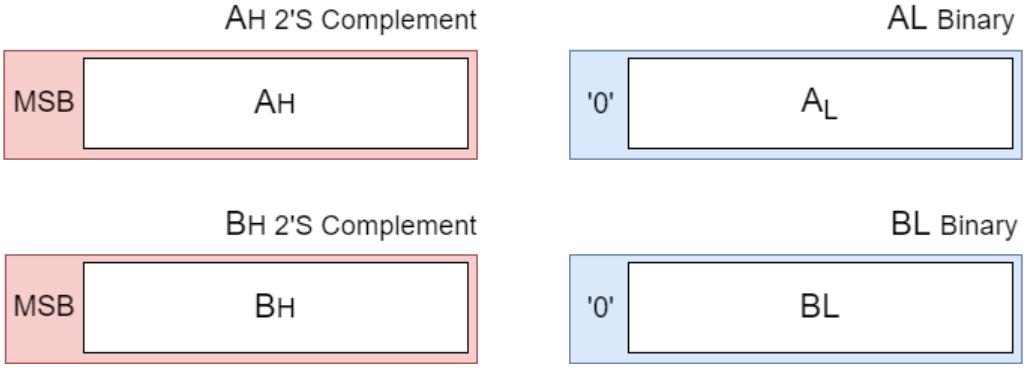


Figure 3.6: 2's complement number decomposition generalization

Clearly, this has to be taken into account in the execution process.
The multiplier used in the unsigned case was enough for executing unsigned multiplications, but it would not work in this case. Indeed, doing multiplications with a 2's complement numbers, means that each sub-product can be positive or negative depending on the sign of the factors. It is necessary to find a solution.
When a multiplication is done using the sub-products strategy, clearly, 3 different kind of multiplications have been executed: 2's complement x 2's complement, 2's complement x binary and binary x binary. This means that they can be done by utilizing : 1 multiplier 2Nx2N 2s complement, 1 NxN binary and 1 NxN 2s complement.
To enhance re-usability and to be able to implement 2's complement multipliers for extending the multiplier utilization to the signed number field, the following strategy has been adopted. The solution consists of adding an extra bit to each

sub-number and using a booth multiplier instead of a standard multiplier.
Booth's algorithm uses a procedure for multiplying binary integers in signed 2's complement representation efficiently way. It examines the multiplier bits and depending on the Booth encoding, the multiplicand may be added to the partial product, subtracted from the partial product, or left unchanged. The fulcrum of this method is the encoding mechanism and it requires numbers represented in 2's complement. For this reason, it is possible to get a signed multiplication by extending the sub-numbers with a bit and using the booth multiplier. The added bit allows the correct Booth encoding depending on the number's sign.



The higher part contains the sign which has to be preserved. To extend it, it is necessary to add an extra bit of the left part of the number which is exactly the MSB of the original number. By doing this, the number will be extended taking into account the sign.
The lower part, as said before, is a binary number. Therefore, in order to extend it, it is enough to add a zero on the left part of the number.
Obviously, due to the extension, each multiplier has to execute a multiplication (N+1)x(N+1). This is a requirement that imposes a hardware modification for all core multipliers.

3.5.1 Signed extension

Each sub-products has to be signed, which means that can be positive or negative depending on the sign of the respective operands. In this situation, the number extension has to take into account the sign of the number. Depending on it, it will consist of a series of ones or a series of zeros added to the number. Let's consider a standard scenario, in which a multiplication of 2Nx2N has to be solved by using the sub-products mechanism. After having obtained the 2N-bit sub-products, the numbers have to be uniform to the same size. The situation is shown in the following picture:

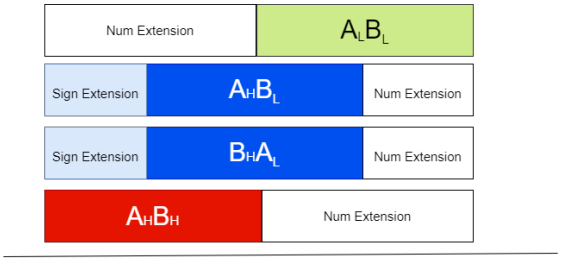


Figure 3.7: addition with signed sub-products

Each sub-product has been aligned and extended, depending on the sign of the sub-products, to allow the execution of the addition.

3.6 Mixed

The Mixed version is another architecture able to do both signed and unsigned multiplications, depending on a parameter that can be changed at run time.

Figure 3.8: bit extension

The multiplier is an extension of the signed version: when the "unsigned computation" is activated, it forces all the extension bits to zero. Therefore, in this case, before the multiplications and the size extension will be done by only adding zeros, as in the unsigned case. If the "unsigned computation is deactivated, the numbers will be extended with a zero or with MSB, depending on the type (high or low), and the size extension will depend on the sign of the sub-product.

3.7 The multiplier

In this chapter, the architectures developed are explained in detail. The goal is to illustrate the different multipliers that have been designed and analyze the functioning. For each architecture, there is an analysis of the different operating modes and the outputs provided.

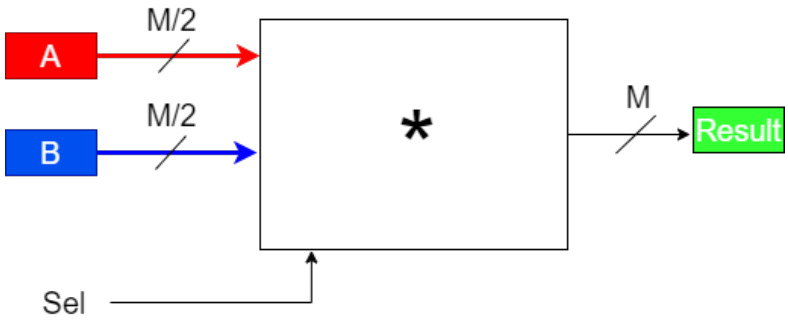


Figure 3.9: Multi-precision multiplier block diagram

The following examples show which are the different multiplications executed by the multi precision multiplier depending on the value of "SEL".

SEL	4X4	8X8	16X16
00	4		
01		2	
10			1
11	2	1	

Figure 3.10: EX M=32 N=4

SEL	4X4	8X8	16X16	32X32
000	8			
001		4		
011			2	
100				1
101		2	1	

Figure 3.11: EX M=64 N=4

3.8 Components analysis

In this section, the main components of the multiplier are analyzed. The multiplier, depicted in a different configuration in the previous sections, is mainly based on the following listed blocks: The logic block, the Core Multipliers, the adders, and the multiplexers

3.8.1 Logic Block

The logic block is situated before the small core multipliers. It is in charge of taking the inputs (A and B) and, depending on the "SEL" signal, providing the inputs for each core multiplier.

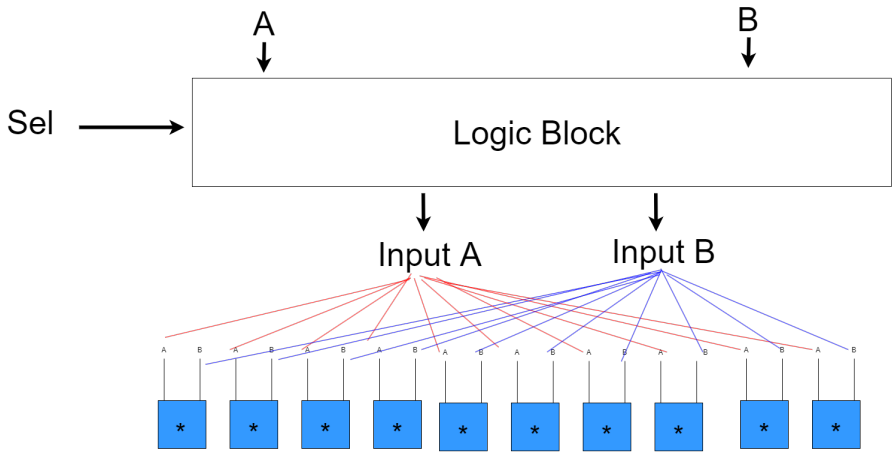


Figure 3.12: Logic Block

According to what was explained before, each multiplication is made of a sum of sub-products. The sub-products needed to change depending on the multiplication. The role of the Logic block is to provide the right inputs to each multiplier, to allow the correct execution of the computation. Here, in the signed and mixed version, the inputs of N-bit are converted into a bit of N+1 bit.
The logic block takes as inputs the two-bit vectors which are the main inputs of the entire circuits(A and B). It edits them and outputs two bits vector, which is the concatenation of the inputs of the core multipliers. In all the computations, except for the one which needs the use of all multipliers, there are some bits of the output equal to zero. For example, in the case of the computation of the 2 multiplications 8x8 which means that Sel=00, in the multiplier configuration with M=32 and N=8, the only multipliers which have to be used are the 2 on the left. The other 2 multipliers will receive all zeros as input as it is shown in the image.

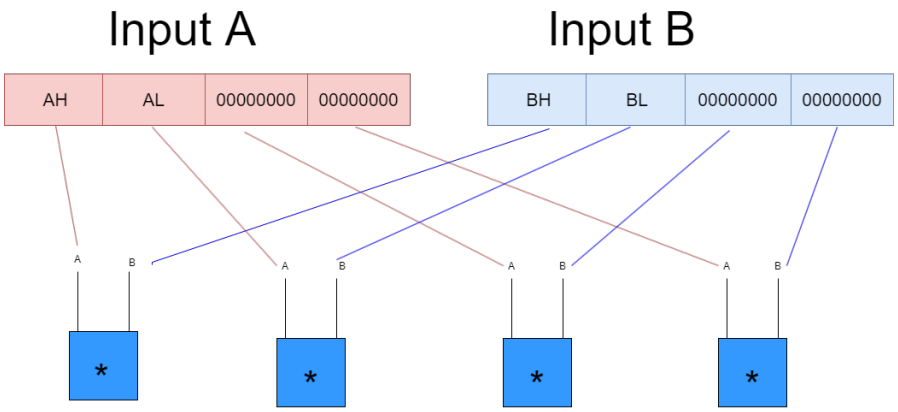


Figure 3.13: Logic block example

3.8.2 Core multipliers

The core multipliers are the all blocks in charge of doing a partial multiplication in the architecture. The number of multipliers depends on the configuration and they can be 4,16 or 64. They receive the inputs from the Logic block and their outputs are placed into the circuit to compute the multiplications.
In the Unsigned version, the core multipliers are standard multipliers described in the High-Level code. They execute multiplication NxN and their result is 2N-bit. The logic signals used inside are "std logic".
In the Signed and Mixed version the core multipliers have to execute multiplications of (N+1)x(N+1), due to the negative weight of the MSB in the 2's complement codification. These multipliers take as inputs two vectors of (N+1)-bit and they output a vector of 2N+1 bit.
There are two different versions of this multiplier: the Booth version and the "High Level" version HL.
The Booth version is based on a Booth multiplier Radix-4. Booth's Algorithm consists of the editing of multiplied numbers according to Booth's encoding. It is possible to reduce the number of partial products by half by using radix-4 Booth Encoding if compared to other encoding techniques.
The Booth multiplier's hardware block is shown in the image. It is composed of three main components: a multiplexer 5 to 1, an adder, and a three-bit encoder.

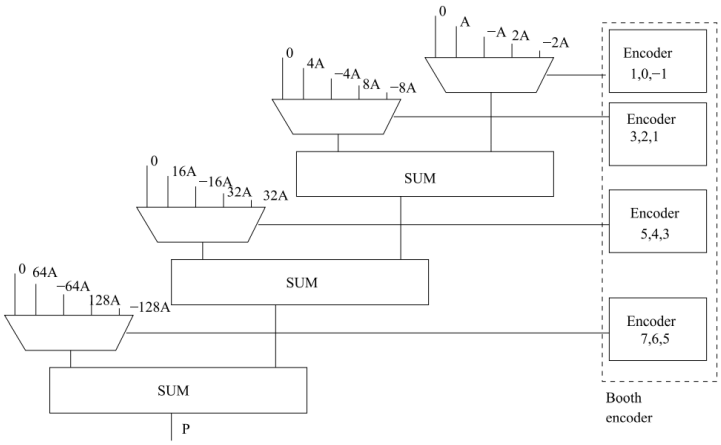


Figure 3.14: Booth multiplier block diagram

The numbers of the components and the hardware complexity of the multiplier vary according to the bit number, following the rules: the number of multiplexers is equal to N/2, the number of decoders is equal to N/2, and the number of adders is equal to N/2 -1. This implies that for larger multiplications, more adders are necessary and the critical path increases.
The HL multiplier is described in High-Level code. This means that implementation depends on the synthesizer. The logic signals used inside are "signed".
During the project stage, other types of multipliers have been tested. Radix-2, Carry sparse Tree, and Dadda.

3.8.3 Adders

The adders are in charge of executing the sums of sub-products. They are arranged on different levels and they execute additions of different sizes.
The first version of the adder was made starting from the theoretical explanation of the sub-products. According to the theory, previously analyzed, the adder has to compute a sum of four sub-products to compute the result. It takes as inputs, the four sub-products which have already been extended.

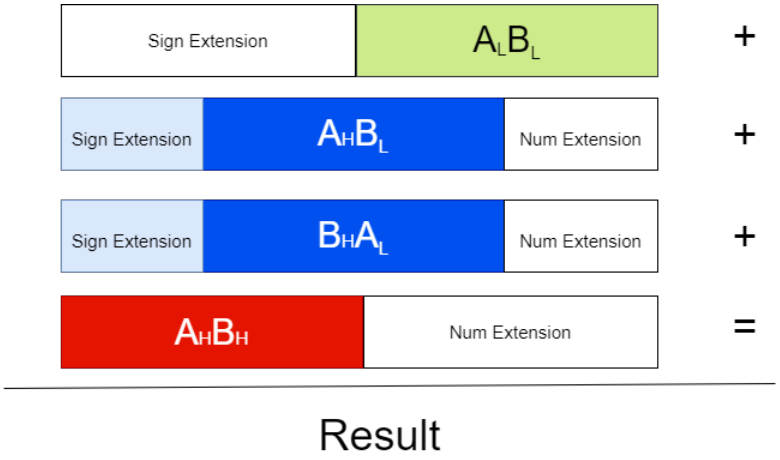


Figure 3.15: Addition representation

The adder, therefore, can be described in High-Level language as :

$$Result <= A + B + C + D; \tag{3.15}$$

From the perspective of synthesis, it is composed of four adders that do a standard addition. The partial result of each addition is summed to the next operand.

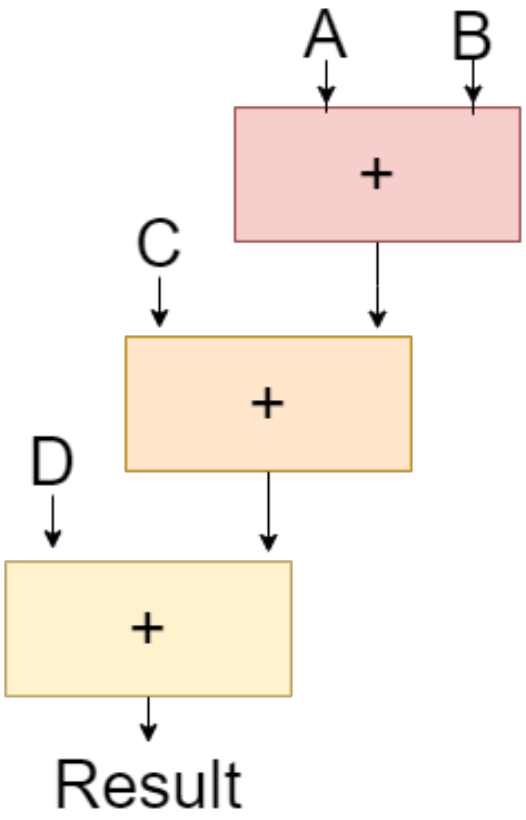


Figure 3.16: Adders chain

To improve the architecture and the performance, the following changes have been made. The addition has been reduced from 4 to 3, by taking advantage of the mathematical rules. Indeed, looking at the sub-products AH*BH and AL*BL, it is straightforward to understand that the two sub-products can be concatenated to form a single addend. This allows to decrease the critical path and save area, by cutting one adder and the hardware circuit in charge of extending the two sub-products.

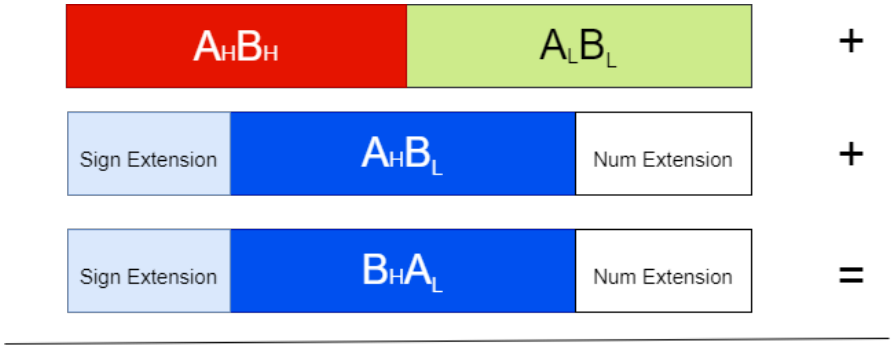


Figure 3.17: Addition improvement

After this improvement, there has been a focus on changing the addition strategy:

- One way consisted of trying to compute a smaller addition and then do a concatenation. Indeed, observing the addition representation, it is clear that, the least significant bits of the results, are precisely the lower part of the Sub-products AL*BL. Therefore, it is possible to compute a smaller size addition of 3/4 N Bits instead of N bits. After it, the lower part can be concatenated with the partial result to obtain the final result.

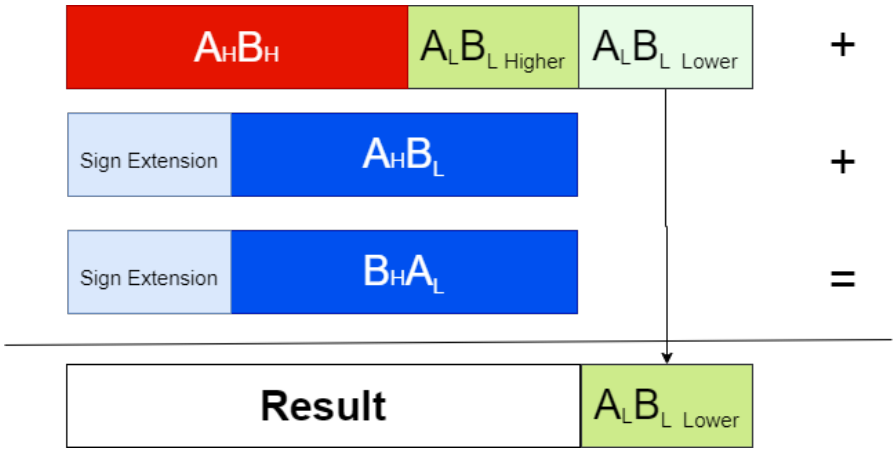


Figure 3.18: Addition improvement : concatenation

- Another way consisted of using a chain of Full Adders(FA in the figure) to reduce the 3 addends to 2 and then doing simple addition.

$$A = (a4, a3, a2, a1, a0) \tag{3.16}$$

$$B = (b4, b3, b2, b1, b0) \tag{3.17}$$

$$C = (c4, c3, c2, c1, c0) \tag{3.18}$$

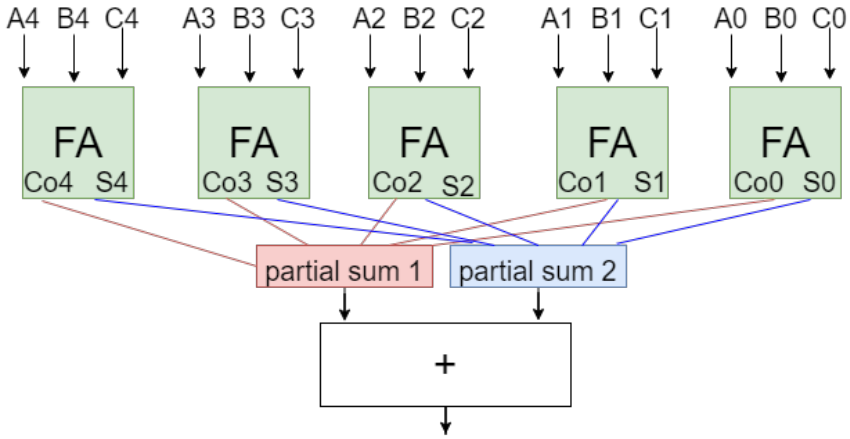


Figure 3.19: Addition improvement : FAs chain

- The last way was a mixture of the two previous ones. The addition executed is ¾ N bit and the lower part of the result is concatenated at the end. Furthermore, the three addends are reduced to 2 thanks to the Full Adders chain.

The second strategy showed the best result from latency and area points of view. Therefore, this one was integrated into every version of the multipliers

3.8.4 Multiplexers

The multiplexers are used to select the right result to be output by the multiplier. It outputs a different value depending on the "SEL" signal. The number of possible outputs depends on the configuration.

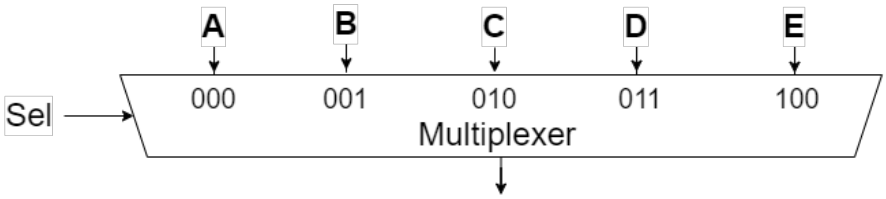


Figure 3.20: Multiplexer 5 to 1

In the configuration N=4 M=64 and N=8 M=128, the multiplexer 5 to 1 is implemented, as the possible results are:
N=8:

- 8 multiplications 8x8;
- 4 multiplications 16x16;
- 2 multiplications 32x32;
- 1 multiplication 64x64;
- 2 multiplications 16x16 and 1 32x32;

N=4

- 8 multiplications 4x4;
- 4 multiplications 8x8;
- 2 multiplications 16x16;
- 1 multiplication 32x32;
- 2 multiplications 8x8 and 1 16x16;

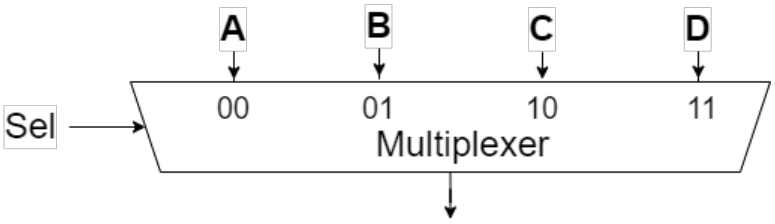


Figure 3.21: Multiplexer 4 to 1

In the configuration N=4 M=32 and N=8 M=64, the multiplexer 4 to 1 is implemented, as the possible results are:
N=8:

- 4 multiplications 8x8;
- 2 multiplications 16x16;
- 1 multiplications 32x32;
- 2 multiplications 8x8 and 1 16x16;

N=4

- 4 multiplications 4x4;
- 2 multiplications 8x8;
- 1 multiplications 16x16;
- 2 multiplications 4x4 and 1 8x8;

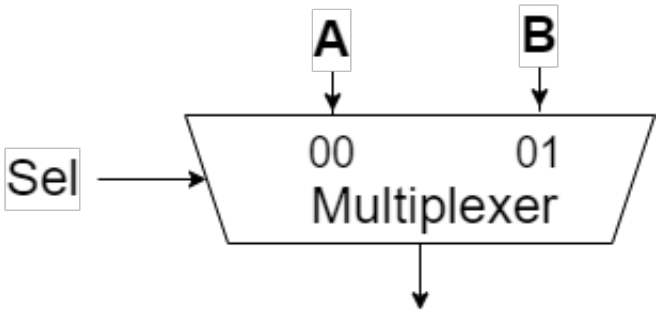


Figure 3.22: Multiplexer 2 to 1

In the configuration N=4 M=16 and N=8 M=32, the multiplexer 2 to 1 is implemented as the possible results are:
N=8:

- 4 multiplications 8x8;

- 1 multiplications 16x16;

N=4

- 4 multiplications 4x4;
- 2 multiplications 8x8;
- 1 multiplications 16x16;
- 2 multiplications 4x4 and 1 8x8;

3.8.5 Parameters

The different versions of the multiplier are distinguished by 2 main parameters: N and M.
N indicates the size of the multiplication executed by the core multipliers which compose the whole structure. If the "core multiplications" are 4x4, N is equal to 4. If the "core multiplications" are 8x8, N is equal to 8.
M indicates the number of bits at the multiplier's output. This parameter imposes a constraint on the multiplier's structure. Indeed, the number of multiplications of different sizes that can be performed in parallel is set by this parameter.
These two parameters were defined at the beginning of the project to be able to define the different configurations of the multiplier.
Let's do a simple example to clarify the meaning of the parameters: M is equal to 32, the number of the multiplications 8x8 executed in parallel will be 2, the number of multiplications 4x4 executed in parallel will be 4, and the number of multiplications 16x16 will be only 1. This is because the sum of the output bits has to be equal to 32 in all cases.

3.9 Configurations

In the following sections, the different configurations of the designed multiplier are listed and explained. They are categorized depending on the value of the parameters M and N, introduced in the previous section.
The goal is to show their structure and to explain the different working modes.

3.9.1 Multipliers with parameter M=16

The following configurations have the parameter M=16, therefore the output is 16-bit long.

Parameter N=4

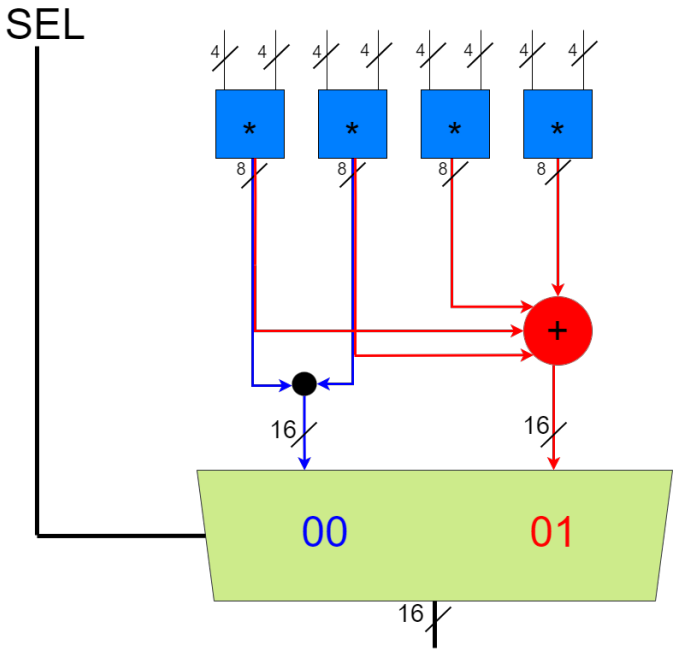
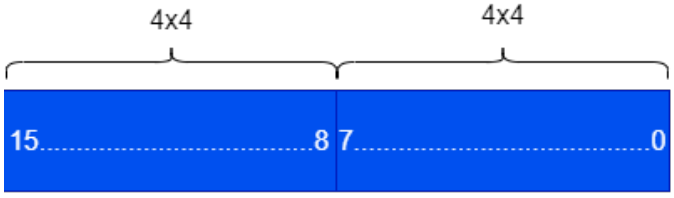


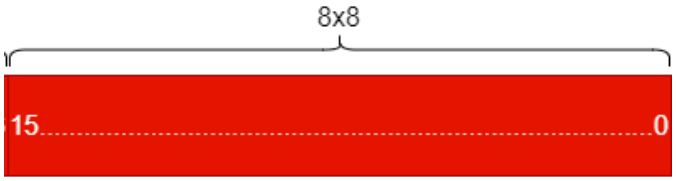
Figure 3.23: Multiplier M=16 N=4

This configuration's output is 16-bit and it is made of four multipliers 4x4, one adder, and one multiplexer 2 to 1. It can execute multiplication 4x4 and 8x8. Depending on the "SEL" signal is possible to obtain the following outputs:

- SEL = "00": 2 multiplications 4x4; In this case, the output will be the concatenation of the results of the 2 multipliers on the left in the picture (blue path). Each one is 8-bit long.



- SEL = "01": 1 multiplication 8x8 computed by using the four multipliers and the adder. The principle of the sub-products previously explained is used.



3.9.2 Multipliers with parameter M=32

The following configurations have the parameter M=16, therefore the output is 32-bit long. Parameter N=8

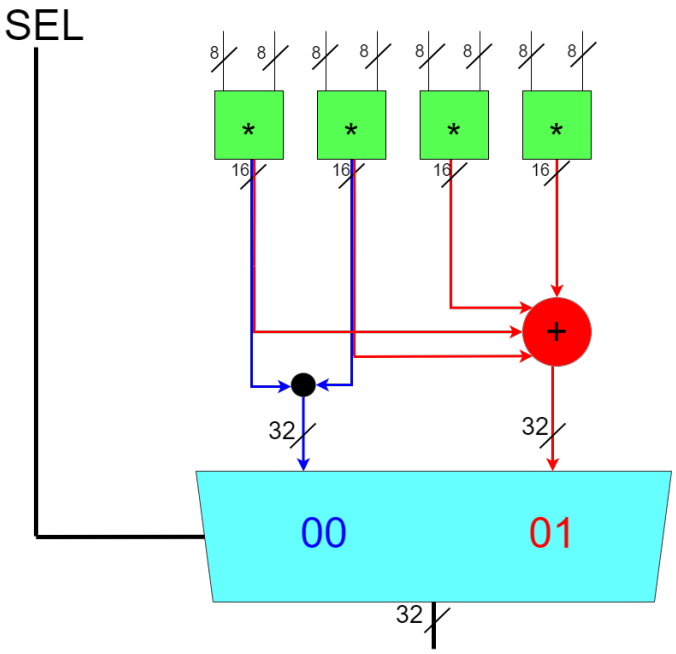
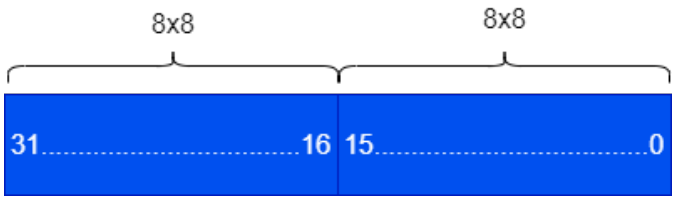


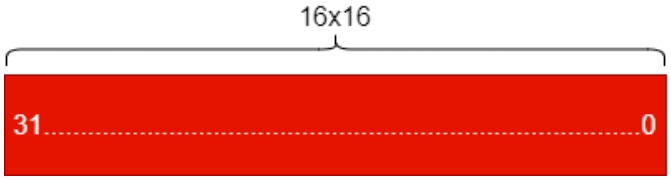
Figure 3.24: Multiplier M=32 N=8

In this multiplier's configuration, the output is 32-bit long and it is made of four multipliers 8x8, one adder, and one multiplexer 2 to 1. It can execute multiplication 8x8 and 16x16. Depending on the "SEL" signal is possible to compute the following outputs:

- SEL = "00": 2 multiplications 8x8; In this case, the output will be the concatenation of the results of the 2 multipliers on the left in the picture (blue path). Each one is 16-bit long.



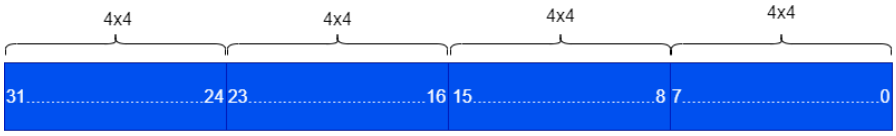
- SEL = "01": 1 multiplication 16x16 computed by using the four multipliers and the adder. In this case, the result will be the sum of the four sub-products (red path).



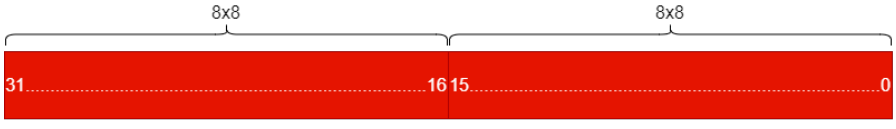
Parameter N=4

This configuration's output is 32-bit and it is made of 16 multipliers 4x4, 4 adders, and 1 multiplexer 4 to 1. It can execute multiplication 4x4,8x8 and 16x16. Depending on the "SEL" signal is possible to compute the following outputs:

- SEL = "00": 4 multiplications 4x4; The output is composed of the concatenation of the results on the 4 multiplications. Each one is 8 bits.



- SEL = "01": 2 multiplication 8x8 computed by using the 8 multipliers and 2 adders. Each one is 16 bits.



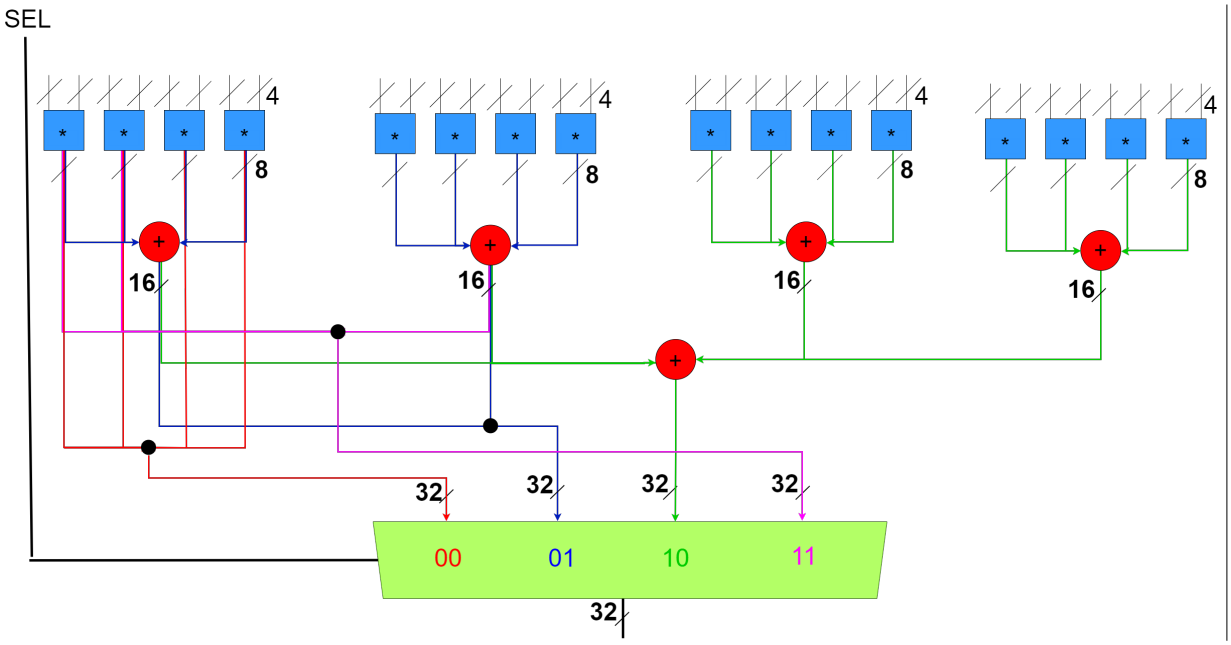
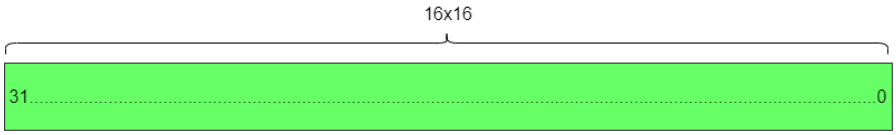
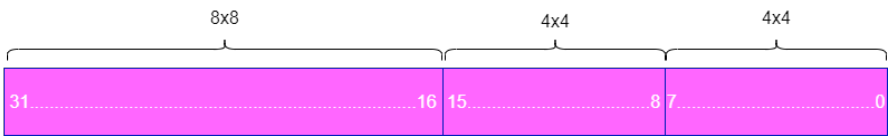


Figure 3.25: Multiplier M=32 N=4

- SEL = "10": 1 multiplication 16x16 computed by using 16 multipliers and 5 adders.1 multiplication 16x16 computed by using 16 multipliers and 5 adders. To compute this output all sub-products are used.



- SEL = "10": 1 multiplication 8x8 concatenated with 2 multiplication 4x4. This is the mixed precision case.



3.9.3 Multipliers with parameter M=64

Parameter N=8

This configuration's output is 64-bit. It is made of 16 multipliers 4x4, 4 adders, and one multiplexer 4 to 1. It can execute multiplication 8x8,16x16 and 32x32. Depending on the "SEL" signal is possible to compute the following outputs:

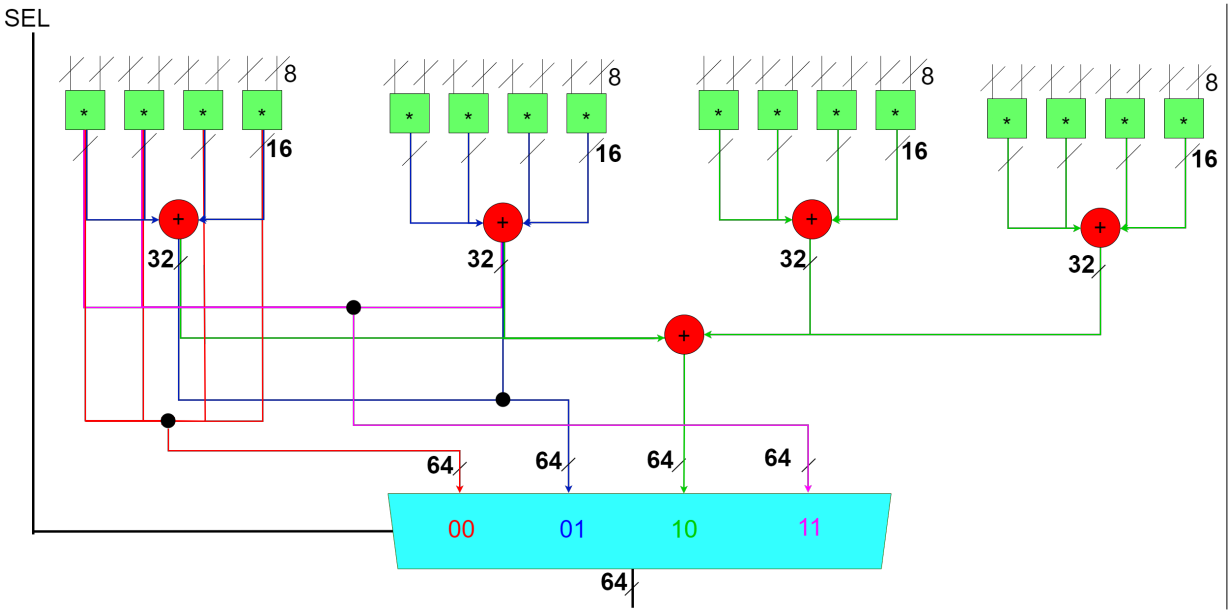
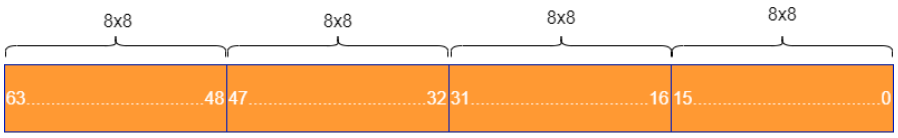
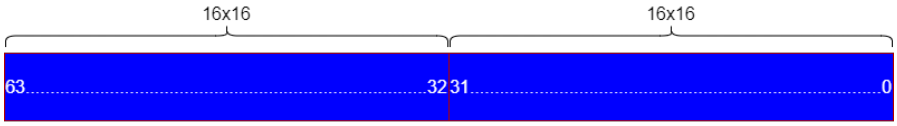


Figure 3.26: Multiplier M=64 N=8

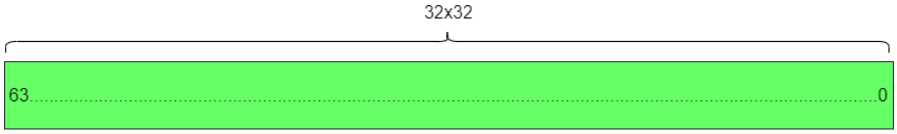
- SEL = "00": 4 multiplications 8x8; The output is composed of the concatenation of the results on the 4 multiplications. (orange path).



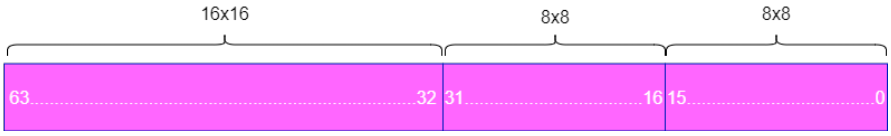
- SEL = "01": 2 multiplication 16x16 computed by using the 8 multipliers and 2 adders. In this case, the output is composed of the concatenation of the two results of the sub-products sum made by using the 2 adders on the left. (blue path)



- SEL = "10": 1 multiplication 32x32 computed by using 16 multipliers and 5 adders. This is the main multiplication and it is computed by calculating 16 sub-products and then by doing two levels of additions. The first level is composed of four adders, each one respectively is under four multipliers, and the second level is composed of the adder that is situated in the center. (green path)



- SEL = "11": 1 multiplication 16x16 and 2 8x8 computed by using 6 multipliers and 1 adder. This mixed case is the result of the concatenation of the multiplications of different precision.



Parameter N=4

This configuration's output is 64-bit. It is made of 64 multipliers 4x4, 17 adders, and 1 multiplexer 5 to 1. It can execute multiplication 4x4,8x8,16x16 and 32x32. Depending on the "SEL" signal is possible to compute the following outputs:

- SEL = "000": 8 multiplications 4x4;

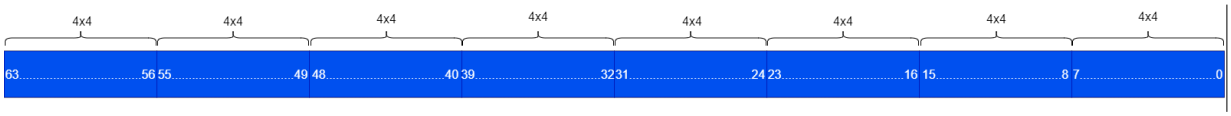
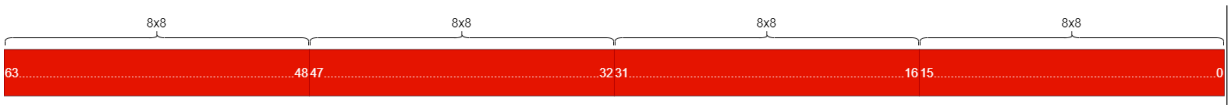
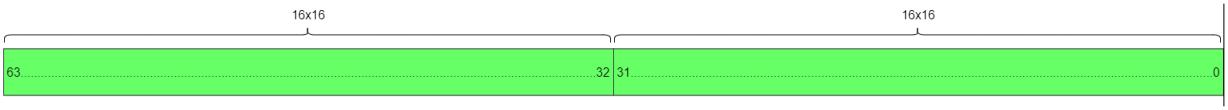


Figure 3.27: Multiplier M=64 N=4

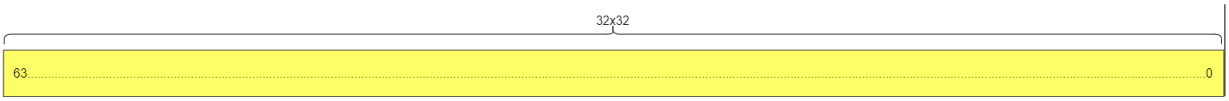
- SEL = "001": 4 multiplication 8x8 computed by using the 16 multipliers and 4 adders.



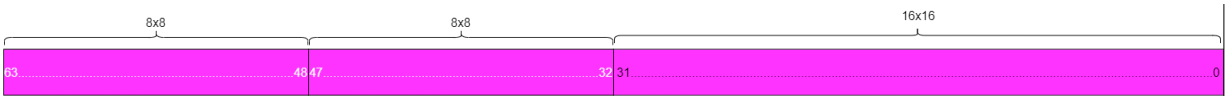
- SEL = "010": 2 multiplication 16x16 computed by using 32 multipliers and 6 adders.



- SEL = "011": 1 multiplication 32x32, to compute this output the all sub-products are used.



- SEL = "100":1 multiplication 16x16 concatenated with 2 multiplication 8x8. This is the mixed precision case.

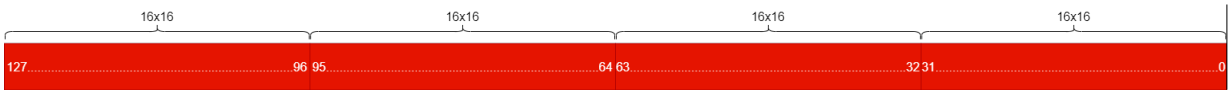


3.9.4 Multipliers with parameter M=128

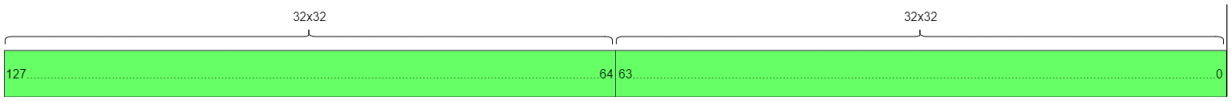
This configuration's output is 128-bit. It is made of 64 multipliers 4x4,17 adders and one multiplexer 5 to 1. Depending on the "SEL" signal is possible to compute the following outputs:

- SEL = "000": 8 multiplications 8x8; The output is composed of the concatenation of the results on the 8 multiplications. Each one is 8 bits.

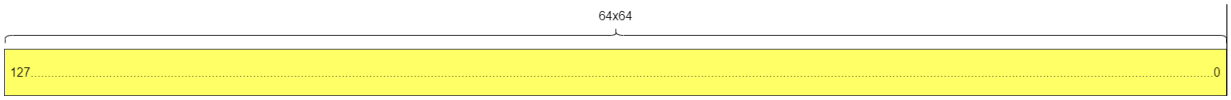
- SEL = "001": 4 multiplication 16x16 computed by using the 16 multipliers and 4 adders;



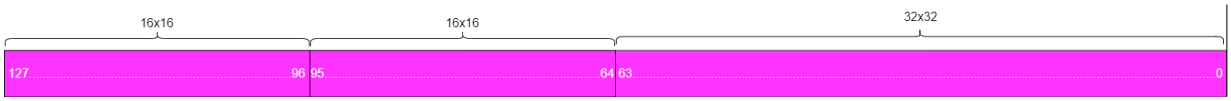
- SEL = "010": 2 multiplication 32x32 computed by using 32 multipliers and 6 adders;



- SEL = "011": 1 multiplication 64x64, to compute this output all sub-products are used;



- SEL = "100": 1 multiplication 32x32 concatenated with 2 multiplication 16x16. This is the mixed precision case.



Chapter 4

Results and analysis

In this chapter, the results obtained are analyzed and commented on.

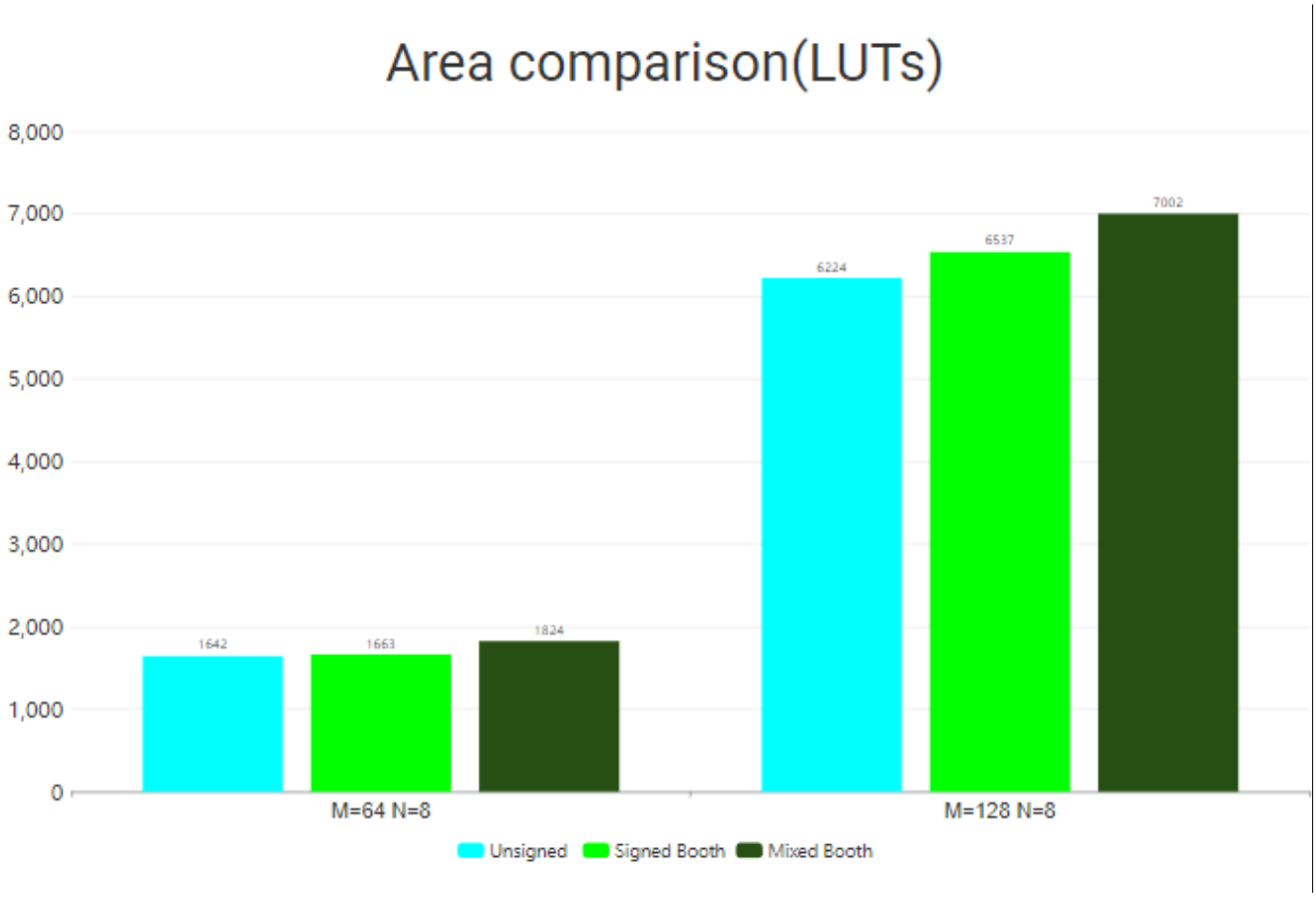
4.1 Experiment description

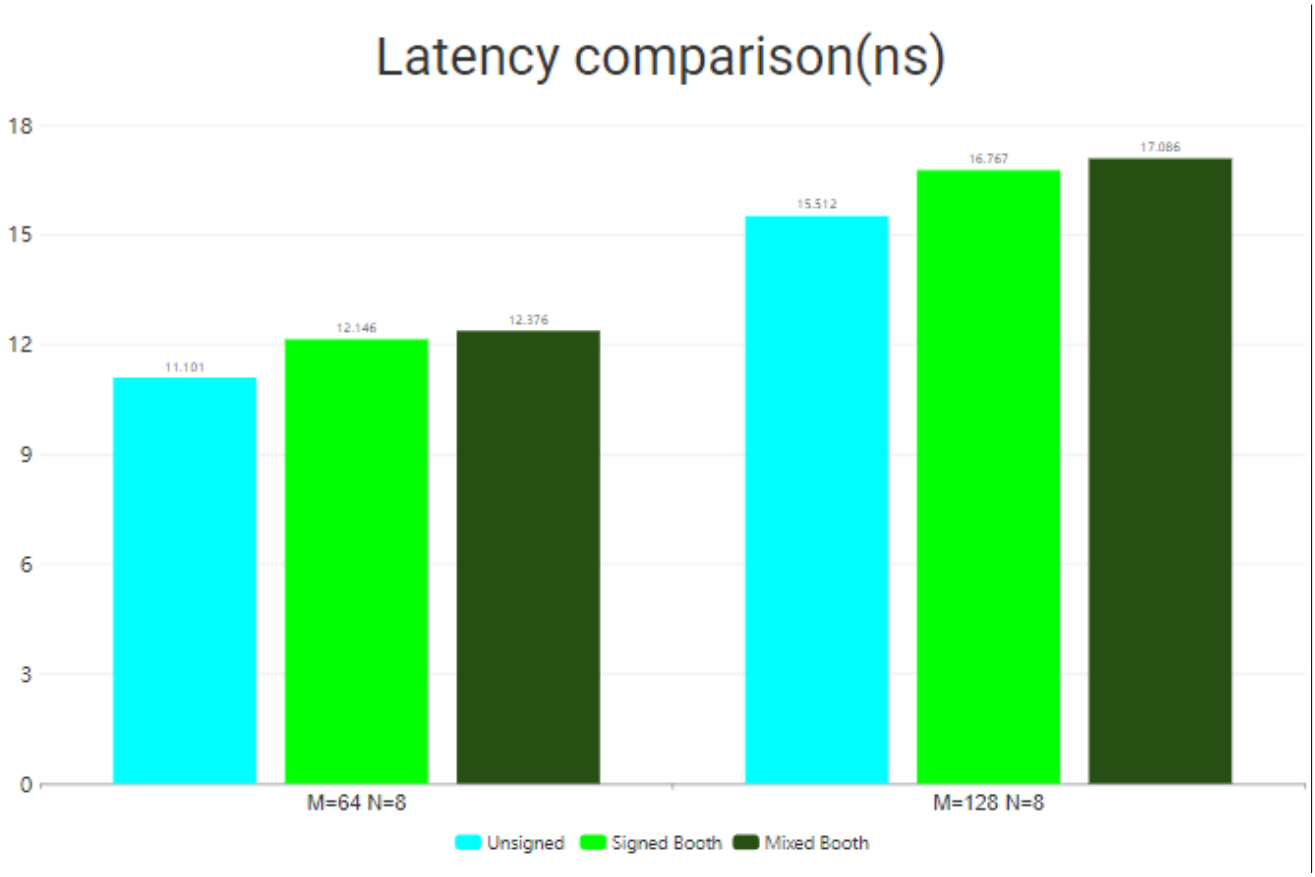
All configurations created have been simulated using the software Xilinx Vivado Logic Simulator . In particular, they were simulated in an FPGA zynq-7000 clg848. In the following paragraphs, the different configurations of the multiplier explained in the previous chapters, are compared.
For each configuration there are the 3 versions of the multiplier: unsigned, signed, and mixed. Furthermore, There are two versions of the signed and mixed version: one is made with the implementation of Booth multipliers, and the other is made with a generic multiplier. described in High-Level Language using the signal type "signed". Both of them use the same mechanism for solving sign operations, explained in the third chapter.
The versions to be compared are:

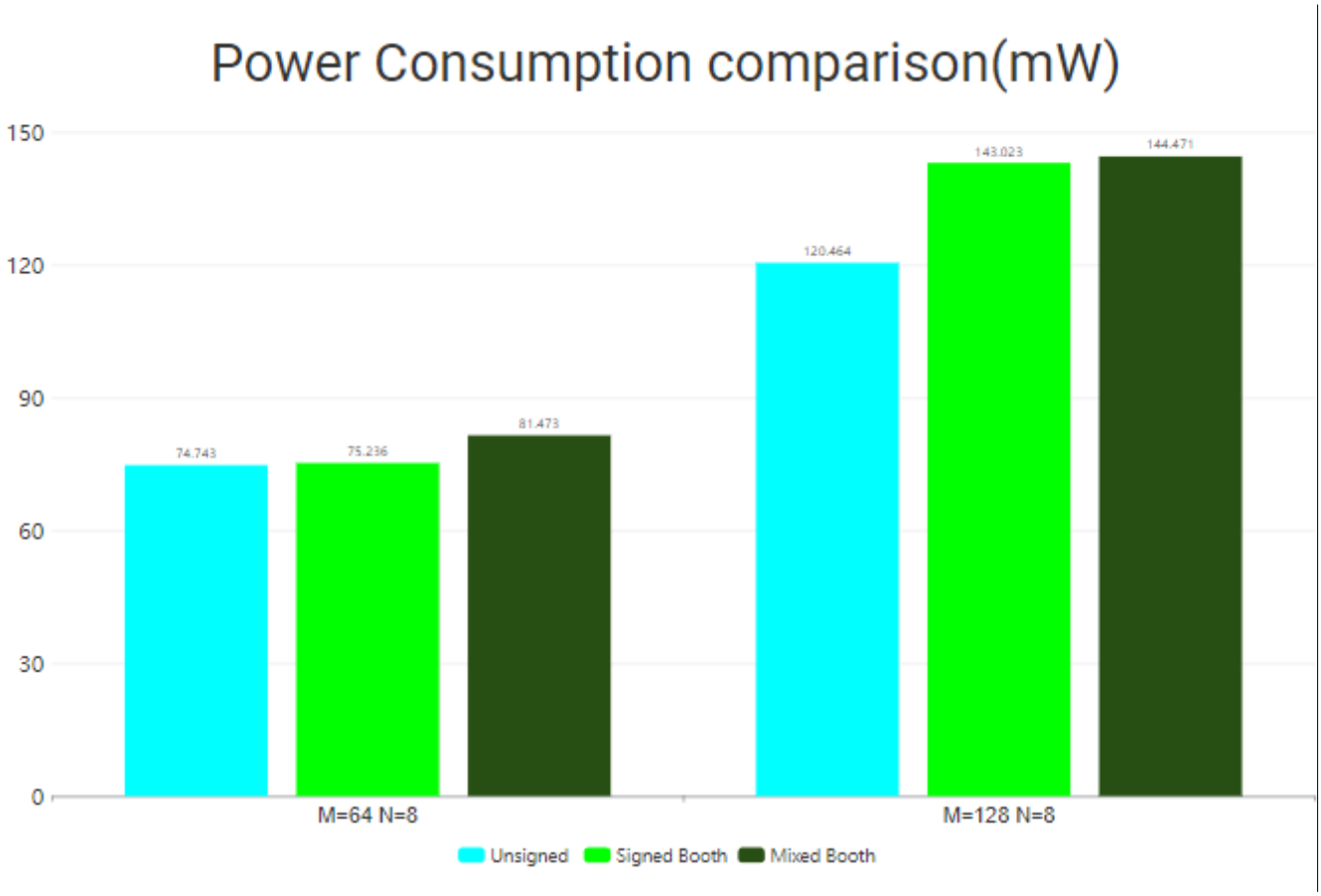
- 1)Unsigned version: it is made with High-level multipliers with "std logic" signals;
- 2)Signed Booth version: it is made with booth multipliers;
- 3)Signed HLM version: it is made with multiplier described in High-Level way;
- 4)Mixed Booth version: it is made with booth multipliers;
- 5)Mixed HLM version: it is made with multiplier described in High-Level way;

4.1.1 Unsigned, Signed, Mixed

The three versions of the multiplier, as it was expected, are different in terms of Area, Latency, and power consumption. The Unsigned version shows the best results from every perspective. It turns out to have the lowest latency, the smallest area needed, and it consumes less than the other two versions, both with Booth and HLM multiplier. This is in line with expectations since the unsigned version is composed of core multipliers that operate only with N bits, while in the other versions the core multipliers operate with N+1 bits.




Latency comparison(ns)
18
15
12
9
6
3
0
11.101
12.146
12.376
15.512
16.767
17.086
M=64 N=8
M=128 N=8
Unsigned
Signed Booth
Mixed Booth



In addition, the sub-products extension, in the unsigned version, consists of only adding "zeros", while in the other versions it is necessary to extend the number according to its sign. Therefore the circuit's block in charge of doing it is simpler and, consequently, smaller and faster compared to the other versions. This trend is observable in all the configurations, and it is more visible in the bigger ones.
The mixed version contains a more complex logic than the signed version. Indeed, the mixed version's logic block is able to provide the input for both unsigned and signed computation, this requires a more complex logic. Indeed, it has an additional block in charge of handling the sign bits depending on the selected mode. It follows then that, for both the Booth version and HLM version, the mixed multiplier has a larger area, higher latency, and higher power consumption.

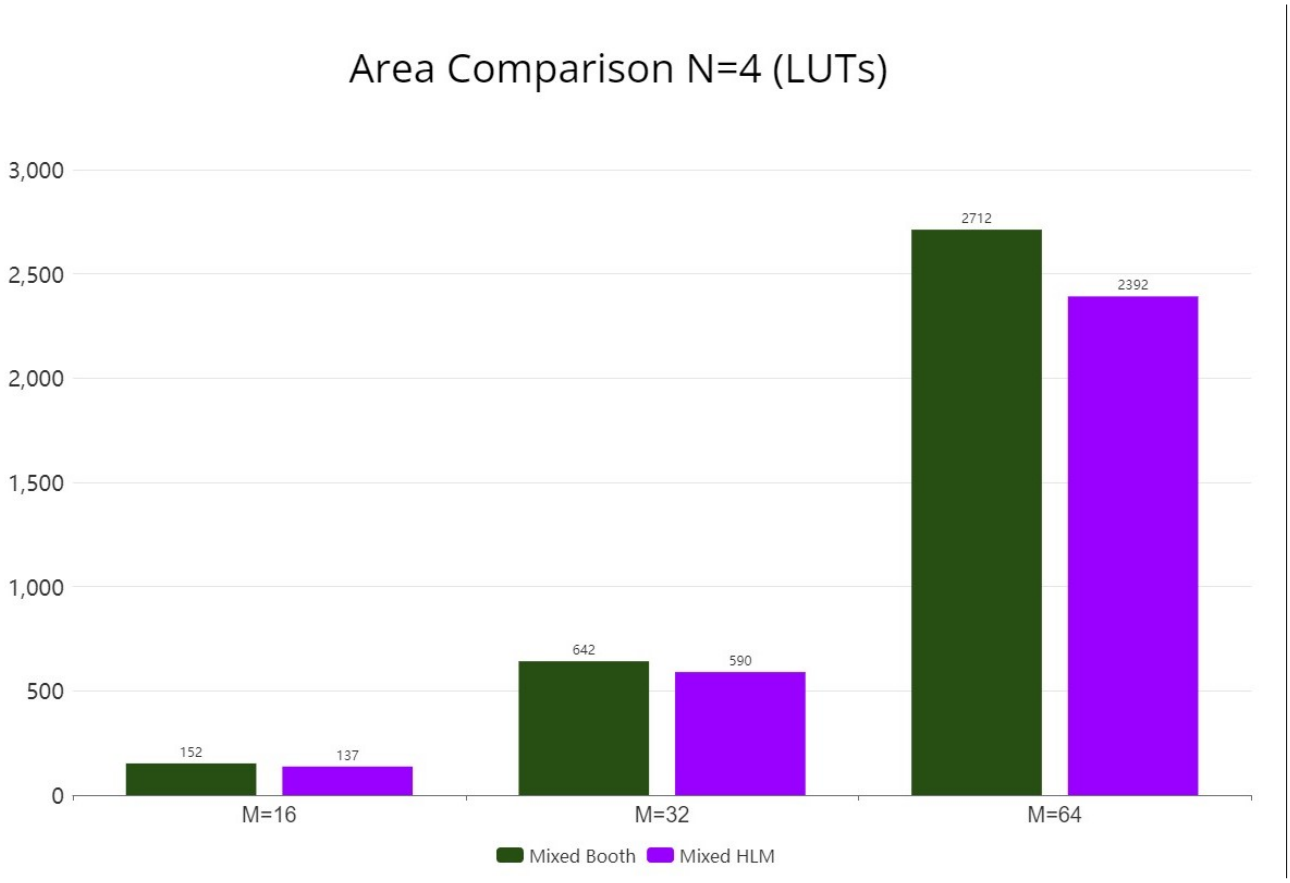
4.1.2 N=4 vs N=8

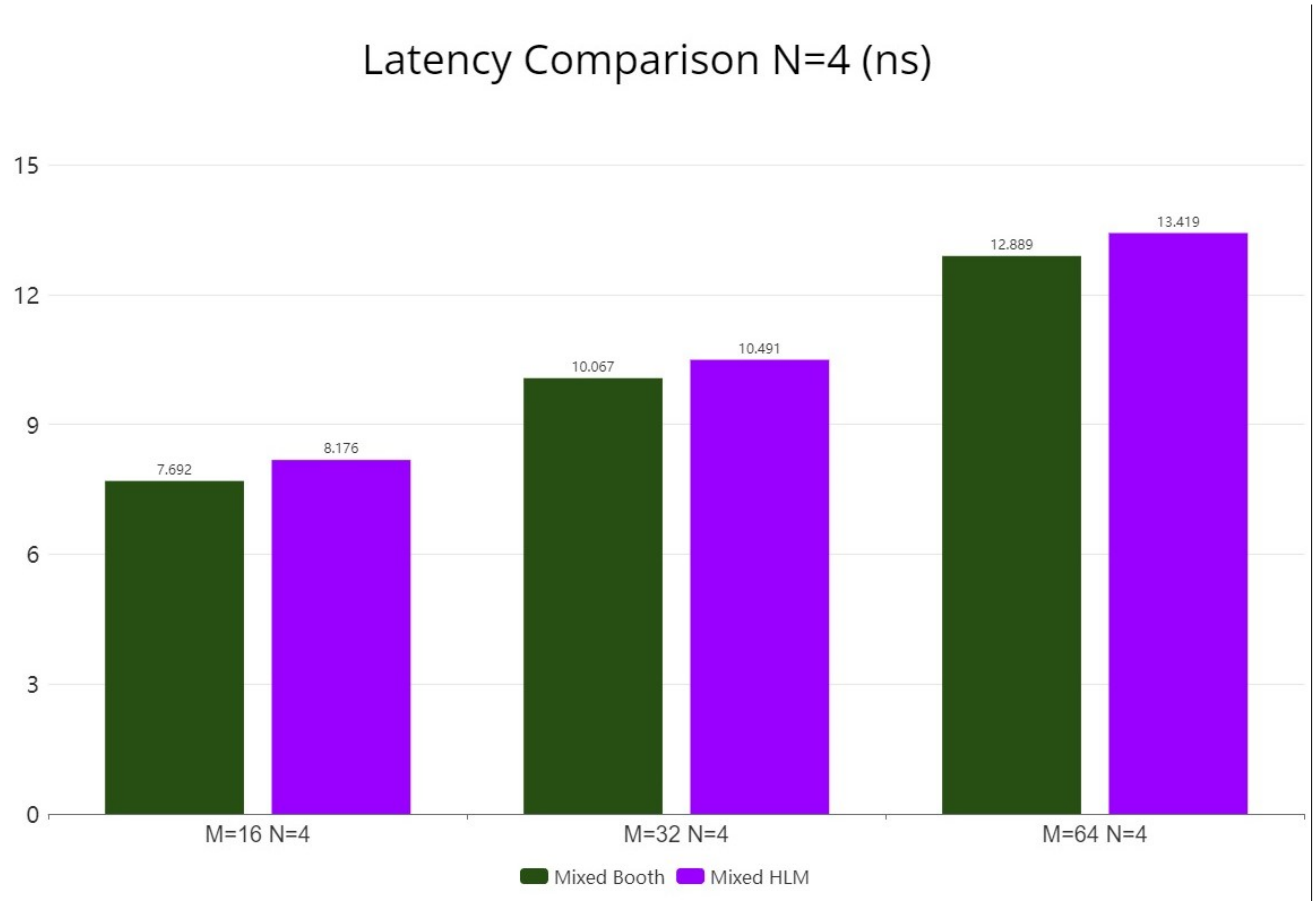
The different configurations have been designed with N=4 multipliers and N=8 multipliers. In particular, the configurations M=32 and M=64 have been implemented by using both of them.

The 2 configurations are both able to execute a 16x16 multiplication, with M=32, and 32x32 multiplication, with M=64. The version with N=4 can also execute 4x4 and 8x8 while, the N=8 version, can only execute 8x8.
The results show that the N=8 versions are faster and they need less area. This is in line with expectations since the N=4 version uses more hardware to compute the same precision multiplication. The M=32 N=4 multiplier contains 16 multipliers and 5 adders arranged on two levels, while the M=32 N=8 multiplier contains only 4 multipliers and 1 adder. Consequently, the critical path of the N=4 version is longer and the necessary area is larger. However, it should be kept into account that the N=4 version can perform more types of multiplication.
It is a different matter for power consumption. The N=4 configurations consume less than the equivalent with N=8. The 8-bit core multipliers consume a lot more than the 4-bit core multipliers and this is visible experimentally.
Finally, comparing the multiplier version with the same parameter M, the N=4 version has the ability to perform a wider range of multi-precision multiplications but it has worse performance in terms of area and latency, however, it has a lower power consumption.

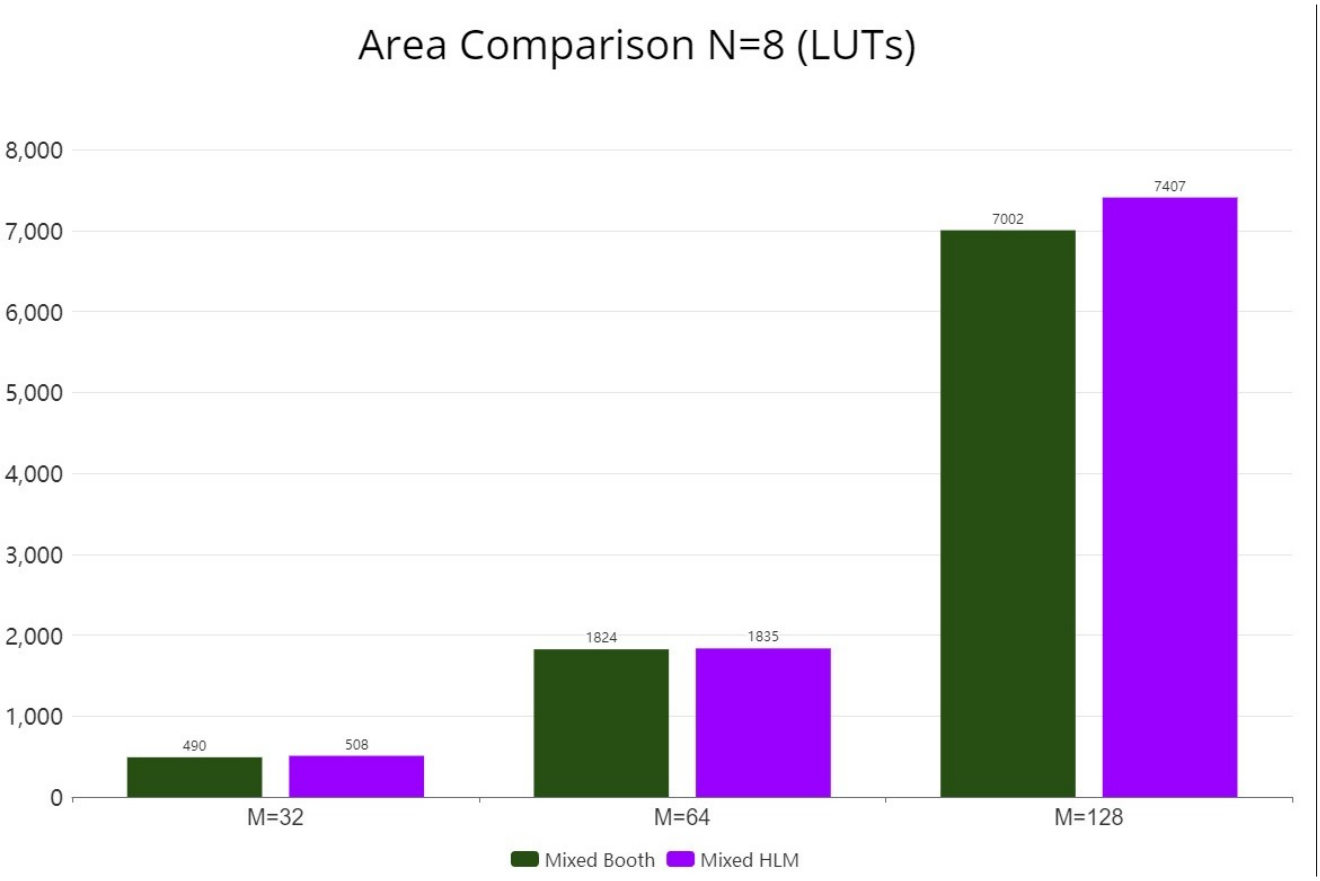
4.1.3 Booth vs HLM

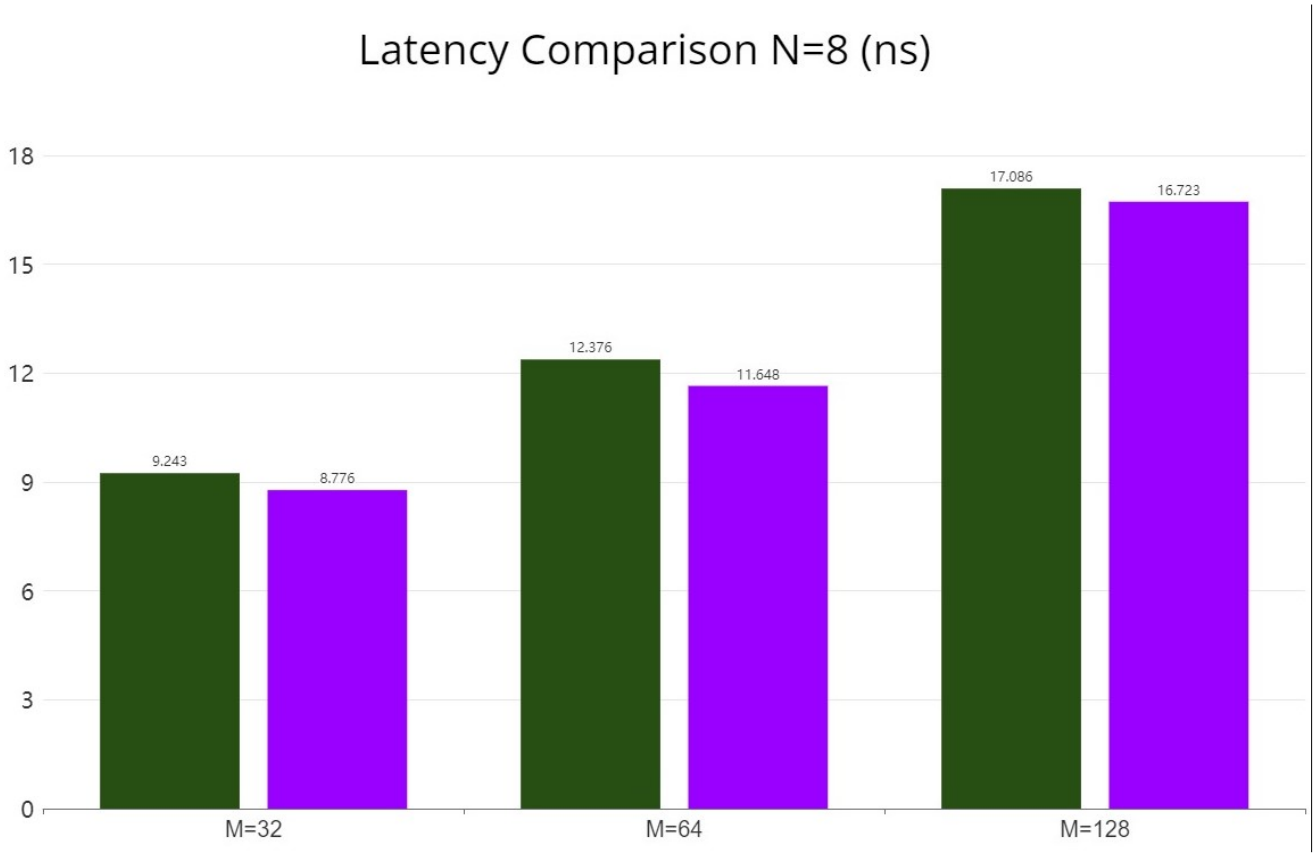
For each configuration, either a version with Booth multipliers, either one with HLM multipliers has been done. The two versions show different results depending on the parameter N.


Area Comparison N=4 (LUTs)
3,000
2,500
2,000
1,500
1,000
500
0
152
137
642
590
2712
2392
M=16
M=32
M=64
Mixed Booth
Mixed HLM

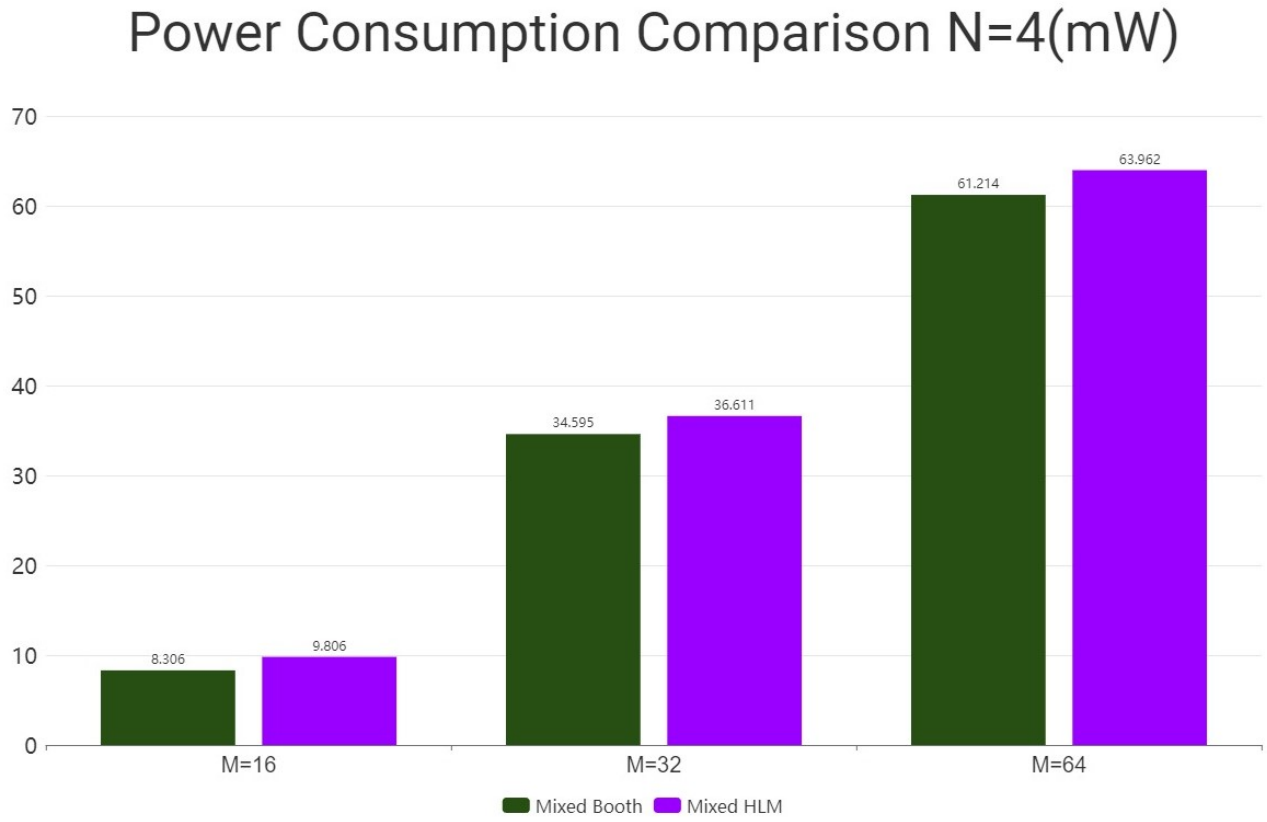


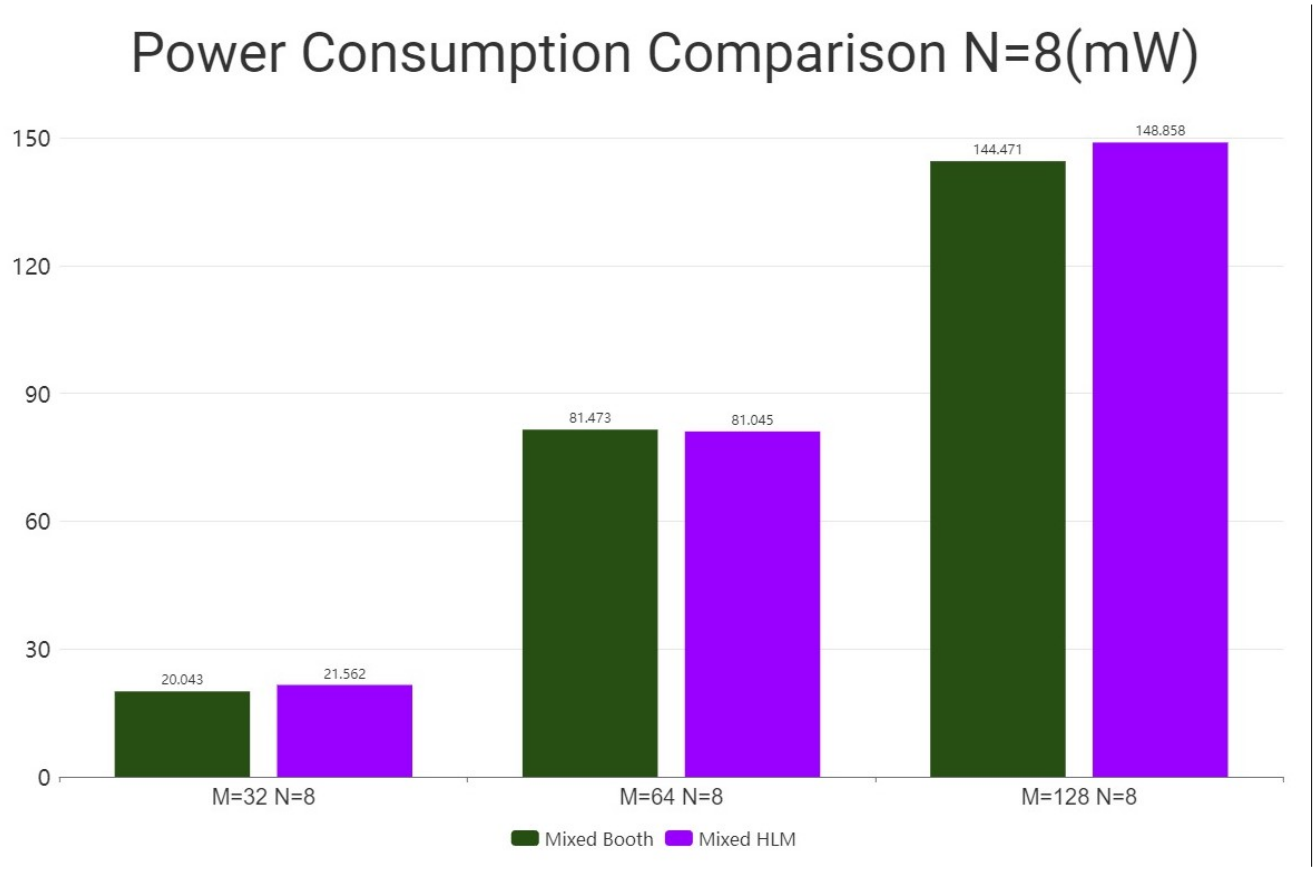
Analyzing the configurations with N=4, the Booth multiplier version turns out to have lower latency, but a larger area than the version made with HLM multipliers. This trend is visible in all the configurations M=16, M=32, and M=64.


Area Comparison N=8 (LUTs)
8,000
7,000
6,000
5,000
4,000
3,000
2,000
1,000
0
490
508
1824
1835
7002
7407
M=32
M=64
M=128
Mixed Booth
Mixed HLM



Analyzing the configurations with N=8, the opposite situation is observed. The HLM multiplier, synthesized by Xilinx, turns out to have lower latency, but a larger area than the multiplier made with HLM. Once again, this trend is observable in all the configurations M=16, M=32, and M=64.
The Booth multiplier's architecture, explained in the previous chapter, is made of a chain of adders, where each one is in charge of doing a partial sum. The number of adders grows linearly with the number of bits and, as a result, the critical path increases. The critical path's increment of the HLM turns out to be less fast compared to the BOOTH. However, the Area shows the opposite trend. The HLM Area, in version N=8, is larger and the area's increment appears faster.


Power Consumption Comparison N=4(mW)
70
60
50
40
30
20
10
0
8.306
9.806
34.595
36.611
61.214
63.962
M=16
M=32
M=64
Mixed Booth
Mixed HLM

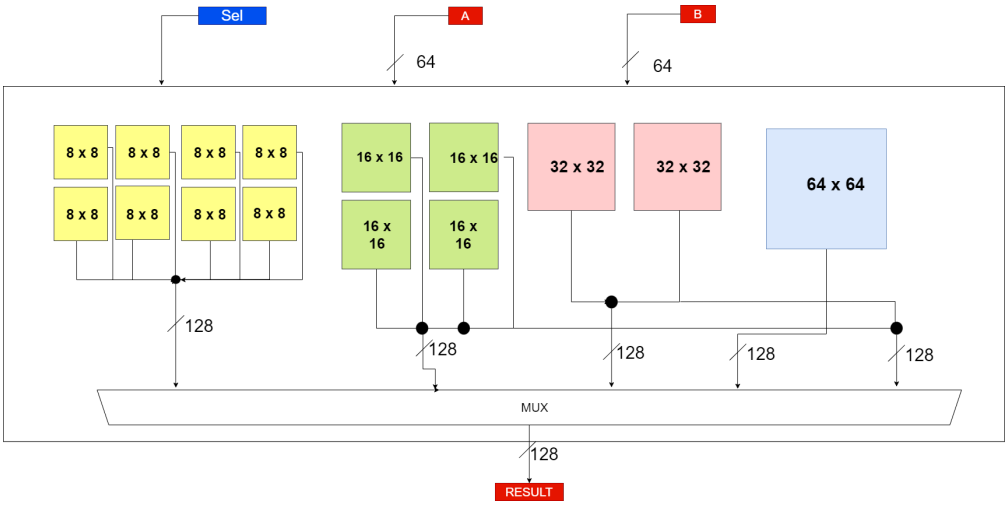


From the perspective of power consumption, the Booth version appears to be the best option in every configuration. The Booth version consumes less than the signed version in both N=4 and N=8, and with all the values of M., This is valid for the Mixed case as well.
In many cases, even the Mixed Booth multiplier consumes less than the signed HLM multiplier.

4.2 Comparison with previous work for Hardware Accelerator

In the state of art other solutions have been implemented for this specific application. In particular, in [7], in the PEs is present a multi-precision multiplier. It is made of different multipliers, able to execute multiplication of different sizes, which work in parallel to compute matrix product and multi-precision results.
To make a comparison between this architecture and the designed multi-precision multiplier, a multi-precision parallel multiplier (MPPM) has been designed. This

multiplier is able to execute signed multi-precision multiplications. An equivalent configuration of the MPPM has been designed for every configuration of our architecture.
Subsequently, each configuration has been synthesized under the same conditions to get results about the area, latency, and power consumption.



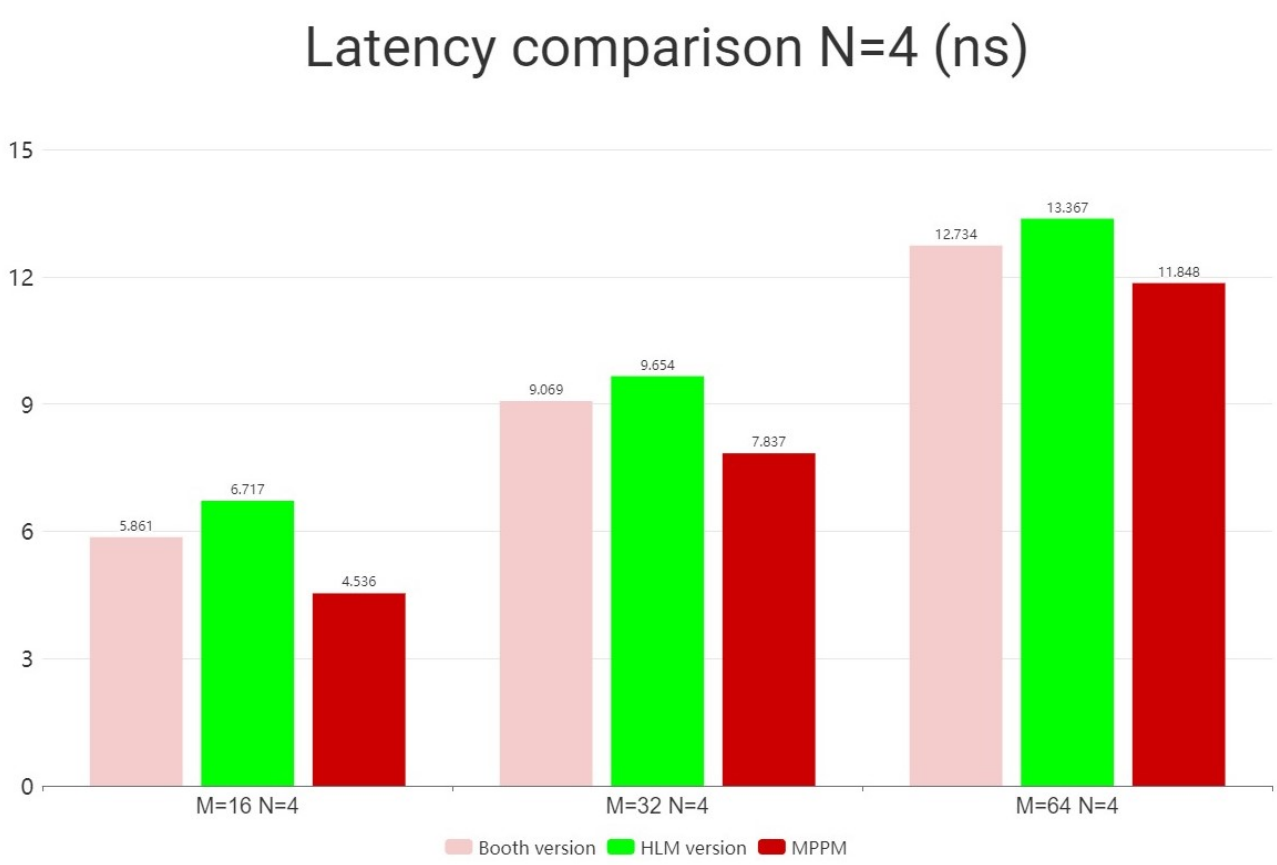
In this section, the following are analyzed and compared:

- 1)Booth version: signed Booth version;

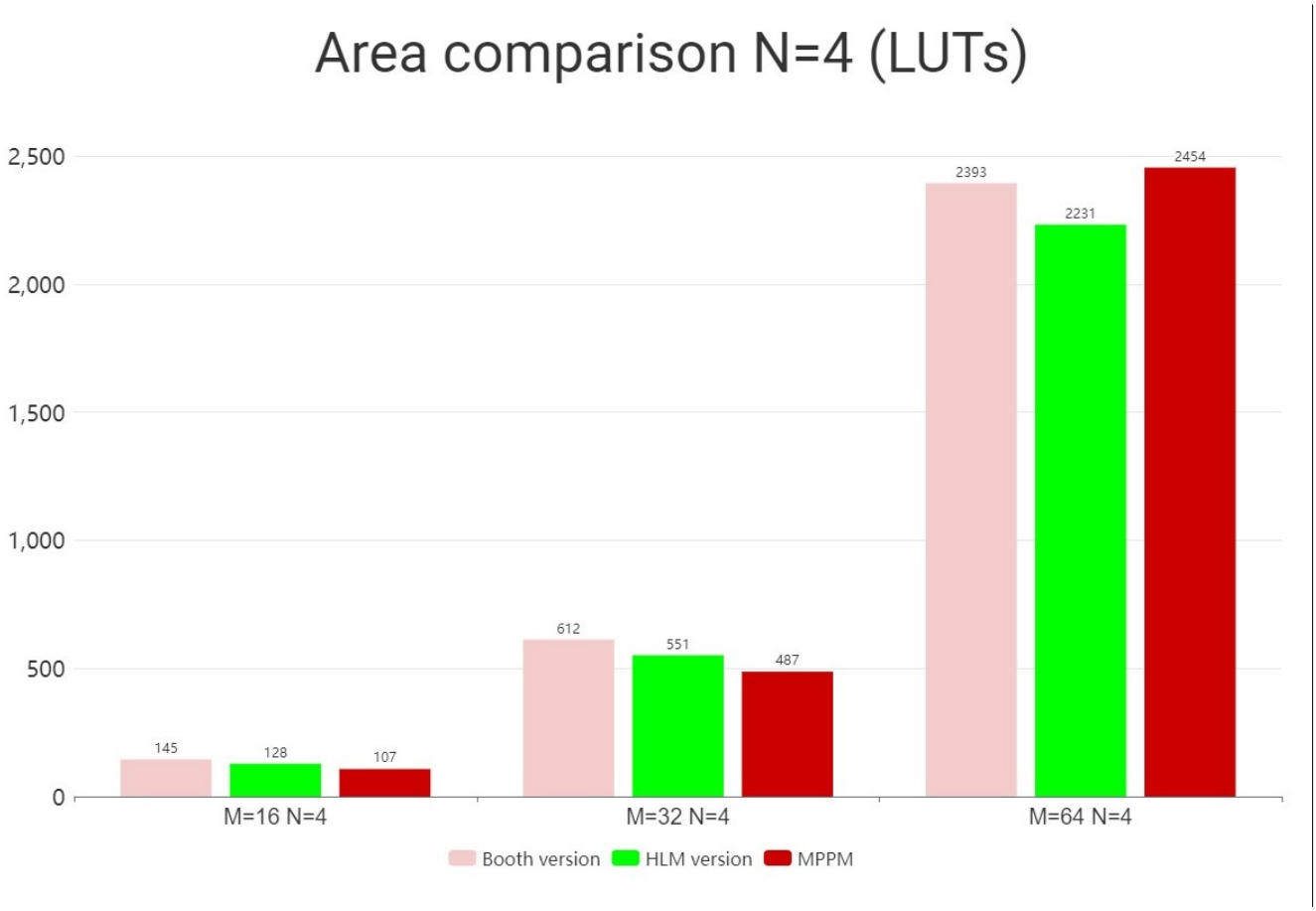
- 2)HLM version: signed HLM version;

- 3)MPPM: multi-precision parallel multiplier from previous work;

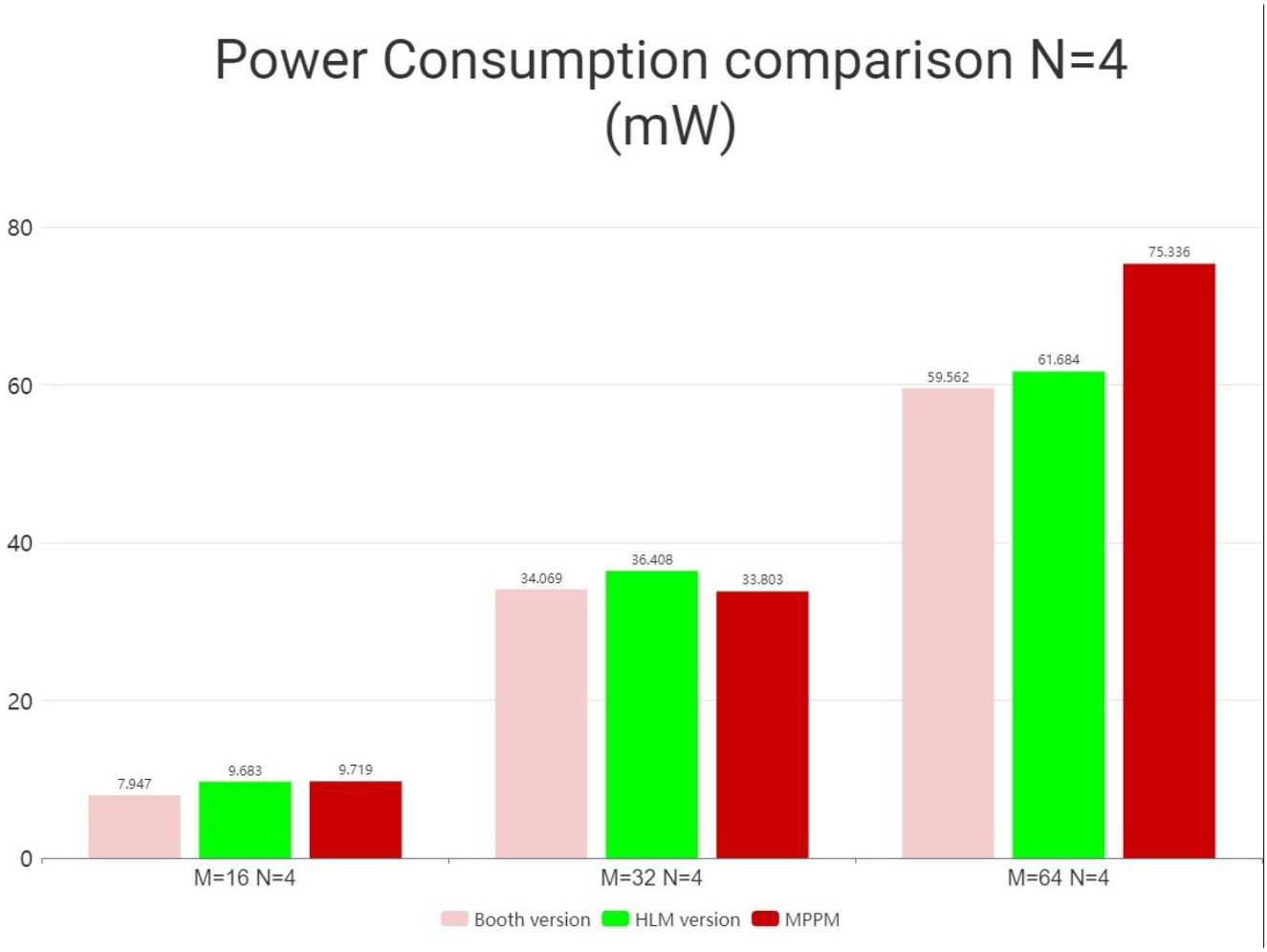
4.2.1 N=4



The MPPM multiplier turns out to have a lower latency in every configuration compared to both Booth and HLM versions. This is in line with the expectations since the MPPM's critical path is represented by the core multiplier able to execute the largest multiplication(32x32 in this case), while our multiplier's critical path is longer and composed of the core multiplier and the adders.

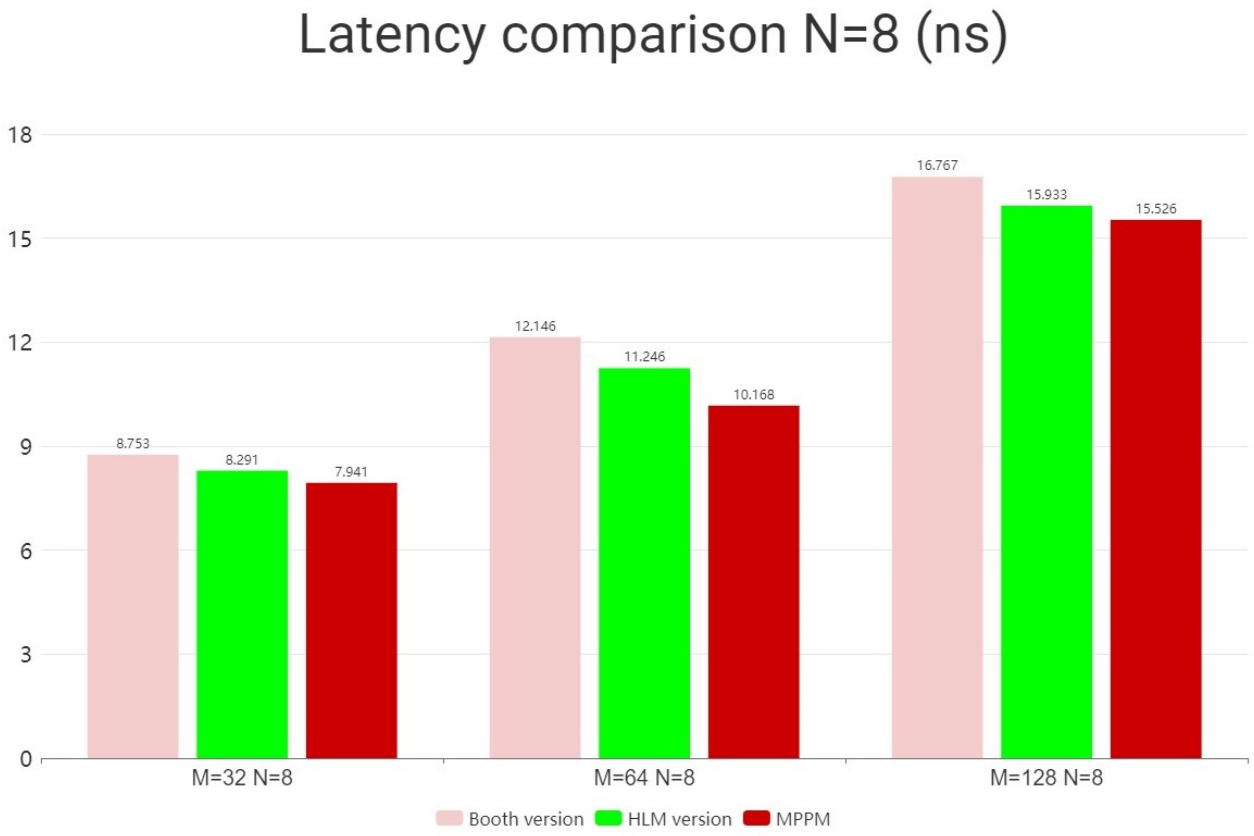


From the area perspective, the MPPM is smaller than both Booth and HLM versions in the configurations M=16 and M=32, while it is larger in the configuration M=64. In the biggest configuration M=64, the hardware re-utilization, in both versions of the multiplier, leads to a better result than the MPPM, whose area increases due to the core multiplier 32x32. In this case, the Area of Booth(2393) and HLM(2231 LUTs) versions are 2.4% and 9.9% less than MPPM(2454).

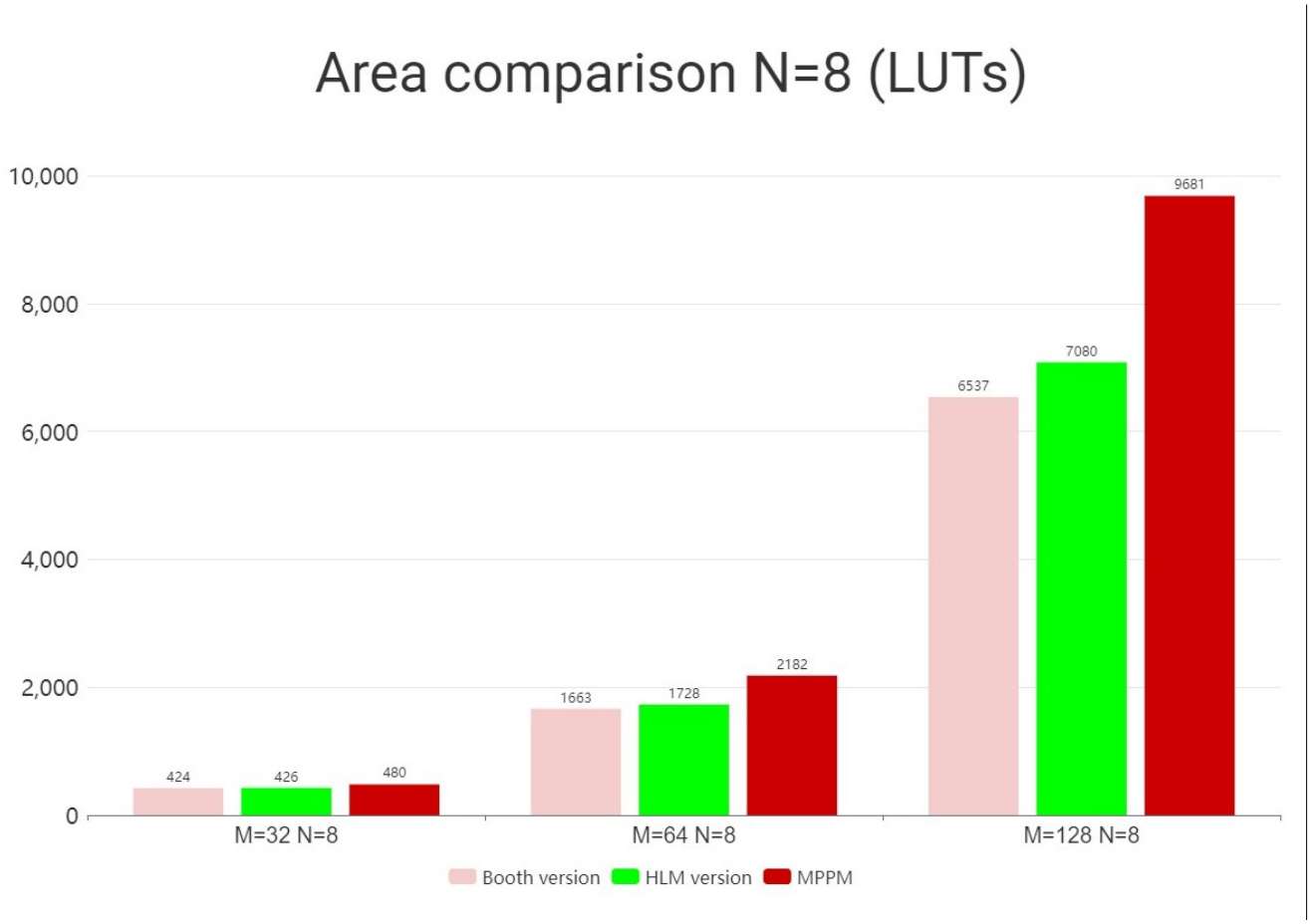


The power consumption of the different architectures appears to be comparable for the configuration M=16 and M=32, while, in the configuration with M=64, the power consumption of the MPPM(75.336mW) is 20.9% larger than Booth(59.562mW) version and 18.1% larger than HLM(61.684) version.

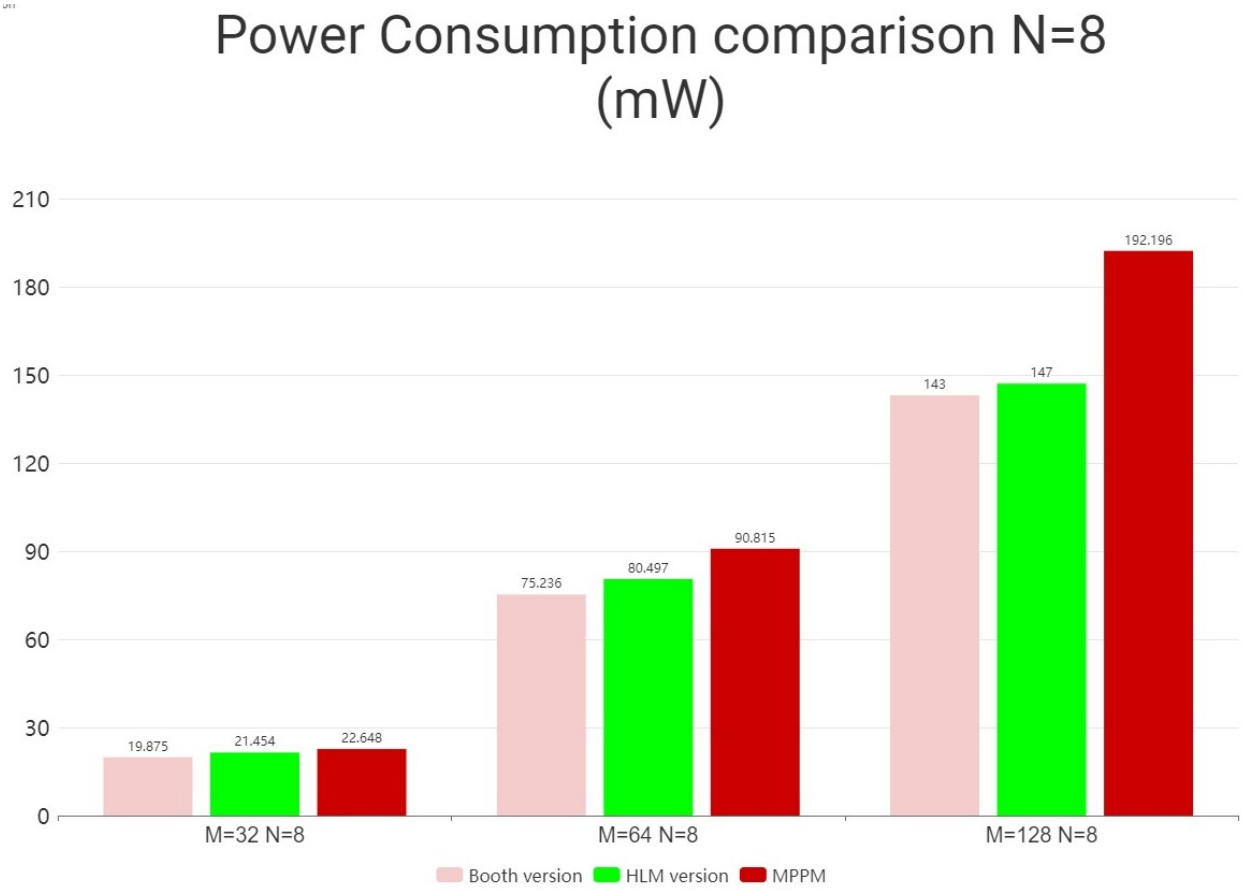
4.2.2 N=8



Comparing the N=8 multiplier version to MMPM, it is observable that the MPPM latency is lower in all the configurations, as in the N=4 version. However, the discrepancy between the latency results is lower. In the configuration M=128, the MMPM latency is 15.526ns, and it turns out to be, respectively, 7.9% and 3.4% more than Booth(16.767ns) and HLM(15.933ns).



About Area, both Booth and HLM versions result in smaller in all the configurations. The difference increases with increasing parameter M: in the configuration with M=32 Booth(424LUTs) is 11.6% and HLM(426LUTs) 12.5% smaller than MPPM(480LUTs), in the configuration with M=64 Booth(1663LUTs) is 23.7% and HLM(1728LUTs) 20.8% smaller than MPPM(2182LUTs) and in the M=128 Booth(6537LUTs) is 32.4% and HLM(7080LUTs) 26.8% smaller than MPPM(9681LUTs).



The power consumption of the different architectures appears to be very similar for the configuration M=16, while, in the configuration with M=64, the power consumption of the MPPM(90.815mW) is 17.1% larger than Booth(75.236mW) version and 11.3% larger than HLM(80.497) version and in the configuration with M=128 MPPM(192.196mW) is 34.4% larger than Booth(143mW) version and 30.7% larger than HLM(147mW).

Chapter 5

Conclusions

In the state of art, several solutions for multi-precision multipliers have been already proposed. The two main design trends are “low power” and “High-performance” which, respectively, aim to keep stable the throughput and decrease the power consumption, and increase the throughput while keeping constant the power consumption. The solution explained in .. represents a viable road. It is an architecture that exploits the theory of “sub-products” sum to compute multi-precision multiplications. Its hierarchical structure, based on the hardware re-utilization mechanism, is promising from the point of view of area and power consumption.
Starting from this knowledge, multi-precision multiplier has been designed which can be a viable and efficient solution in deep learning applications. It has been designed in different configurations and versions to be able to study its behavior and draw conclusions.
The multi-precision multiplier is based on the sums of sub-products, it can execute multi-precision multiplication, matrix product, and mixed precision multiplications. The different blocks have been designed from scratch and they have been customized to this specific application.
It has been developed in three different versions: unsigned, signed, and mixed. The unsigned version can execute only unsigned multiplication, it is made with HL core multipliers and its logic is simpler than the other versions.
The signed version is able to execute signed multiplications. To achieve this, the core multiplier has been replaced with the Booth multiplier or signed HL multiplier.
The mixed version can execute both signed and unsigned multiplications. The working mode can be changed at a run time thanks to an appropriate signal which can be modified.
Comparing the three versions it has been observed that the area, the latency, and the power consumption grow with the increasing complexity of the circuit. The unsigned version has less complex logic and its operations require fewer bits,

consequently, it is the fastest, the smallest and it consumes less. The mixed and signed version is more similar, but due to the changing needed in the mixed logic, the signed version shows better performance from every perspective.
For each version, different configurations based on two parameters: M the output size, and N the core multipliers multiplications size, have been designed. By analyzing them, it has been observed that two solutions with the same output size (M has the same value) achieve different results depending on N. The version with N=4, despite having a higher granularity of precision, has a larger area and a higher latency. Nevertheless, its power consumption is lower.
Furthermore, every multiplier configuration in signed and mixed versions has been studied with both Booth and HLM core multipliers. By evaluating the results, it was possible to conclude that the Booth version performs better in terms of latency in the configurations with N=4, while it has a larger area than the HLM. In contrast, the opposite situation is observable in the configurations with N=8, in which the HLM has a lower latency but it has a larger area. Finally, in every configuration, independently on the value of the parameters N and M, the Booth version consumes less power than the HLM version.
In order to estimate the proposed solution, a similar architecture, which is supposed to be applied in a hardware accelerator, has been designed. This architecture is present in[7]. The multi-precision parallel multiplier is an alternative with redundant logic instead of hardware reuse. The comparison between the two different solutions has been made differentiating between N=4 configurations and N=8. In the first case, the MPPM shows better results except for the configuration M=64, in which the proposed work needs less Area and consumes less, despite the higher latency.
In the configurations with N=8, the proposed multiplier achieves a better result than the MPPM from Area and Power perspective. The multiplier turns out to be really smaller than the alternative and to consume less power. Furthermore, its latency appears comparable to the alternative. This result is promising and could be the way to increase the PEs of a hardware accelerator, for the same area, keeping power consumption constant but increasing throughput. It could represent a great opportunity to improve the efficiency of the entire architecture.

Bibliography

[1] Jie Sun Zulong Lai Liping Di Member IEEE Ziheng Sun Jianbin Tao and Yonglin Shen. «Multi-level deep learning network for county-level corn yield estimation in the U.S. Corn Belt». In: (2003), pp. 569–571 (cit. on p. 1).

[2] Nosherwan Ijaz;Yuehua Wang. «Automatic Steering Angle and Direction Prediction for Autonomous Driving Using Deep Learning». In: (2021) (cit. on p. 1).

[3] Hayato Matsumoto;Soh Yoshida;Mitsuji Muneyasu. «propagation-Based Fake News Detection Using Graph Neural Networks with Transformer». In: (2021) (cit. on p. 1).

[4] Nosherwan Ijaz;Yuehua Wang. «On Application of Natural Language Processing in Machine Translation». In: (2018) (cit. on p. 1).

[5] Sarfaraz Hussein; Pujan Kandel; Candice W. Bolan; Michael B. Wallace; «Lung and Pancreatic Tumor Characterization in the Deep Learning Era: Novel Supervised and Unsupervised Learning Approaches». In: (2017) (cit. on p. 1).

[6] MAURIZIO CAPRA; BEATRICE BUSSOLINO; ALBERTO MARCHISIO;GUIDO MASERA;MAURIZIO MARTINA;MUHAMMAD SHAFIQUE. «Hardware and Software Optimizations for Accelerating Deep Neural Networks: Survey of Current Trends;Challenges; and the Road Ahead». In: (2020) (cit. on pp. 1, 2).

[7] Swagath Venkataramani; Vijayalakshmi Srinivasan ; Wei Wang ; Sanchari Sen ; Jintao Zhang; Ankur Agrawal; Monodeep Kar; Shubham Jain; Alberto Mannari; Hoang Tran; Yulong Li; Eri Ogawa; Kazuaki Ishizaki; Hiroshi Inoue; Marcel Schaal;Mauricio Serrano; Jungwook Choi; Xiao Sun; Naigang Wang; Chia-Yu Chen; Allison Allain; James Bonano; Nianzheng Cao; Robert Casatutak; Matthew Cohen; Bruce Fleischer; Michael Guillorn; Howard Haynie; Jinwook Jung; Mingu Kang; Kyu-hyoun Kim; Siyu Koswatta; Saekyu Lee; Martin Lutz;Silvia Mueller; Jinwook Oh; Ashish Ranjan; Zhibin Ren;Scot Rider; Kerstin Schelm; Michael Scheuermann ; Joel Silberman; Jie Yang;

Vidhi Zalani; Xin Zhang; Ching Zhou; Matt Ziegler; Vinay Shah ; Moriyoshi Ohara; Pong-Fei Lu; Brian Curran; Sunil Shukla; Leland Chang and Kailash Gopalakrishnan IBM Research. «RaPiD: AI Accelerator for Ultra-low Precision Training and Inference». In: (2021) (cit. on pp. 2, 52, 61).

[8] Yu-Hsin Chen; Joel Emer and Vivienne Sze. «Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks». In: (2016) (cit. on p. 2).

[9] Xiaoxiao Zhang; Student Member; IEEE; Farid Boussaid; Senior Member; IEEE; and Amine Bermak; Fellow; IEEE. «32 Bit×32 Bit Multiprecision Razor-Based Dynamic Voltage Scaling Multiplier With Operands Scheduler». In: (2014) (cit. on p. 4).

[10] Shen-Fu Hsiao; Yu-Chang Chen; Yu-Che Yen. «Efficient Quantization and Multi-Precision Design of Arithmetic Components for Deep Learning». In: (2021) (cit. on p. 5).

[11] D. Jackuline Moni and P. Eben Sophia. «DESIGN OF LOW POWER AND HIGH SPEED CONFIGURABLE BOOTH MULTIPLIER». In: (2011) (cit. on p. 6).

[12] R. Ramya and S. Moorthi. «Design and Implementation of Accuracy Configurable Multi-Precision Multiplier Architecture for Signal Processing Applications». In: (2018) (cit. on p. 7).

[13] Claudio Brunelli; Perttu Salmela; Jarmo Takala and Jari Nurmi. «A Flexible Multiplier for Media Processing». In: (2005) (cit. on p. 8).

[14] Kai Li1; Wei Mao1; Xinang Xie1; Quan Cheng1; Huan Xie2; Zhenjiang Dong2 and Hao Yu. «Multiple-Precision Floating-Point Dot Product Unit for Efficient Convolution Computation». In: (2021) (cit. on p. 9).

[15] Manish Kumar Jaiswal and Hayden K.H S. «Dual-Mode Double Precision / Two-Parallel Single Precision Floating Point Multiplier Architecture». In: (2021) (cit. on p. 10).

[16] Ivan A. Belyaev; Andrey A. Belyaev; Tatiana V. Solokhina; Yaroslav Y.Petrichkovich «A High-perfomance Multi-format SIMD Multiplier for Digital Signal Processors». In: (2020) (cit. on pp. 11, 13–15).

[17] Vaijyanath Kunchigi; Linganagouda Kulkarni; Subhash Kulkarni. «High Speed and Area Efficient Vedic Multiplier». In: (2017) (cit. on p. 13).

[18] A. Karatsuba; Yu. Ofman. «Multiplication of many digital numbers by automatic computers». In: (1962) (cit. on p. 14).