Error-tolerant multiplier for high speed application

Khaing, Yin Kyaw

2011

Khaing, Y. K. (2011). Error-tolerant multiplier for high speed application. Master’ s thesis, Nanyang Technological University, Singapore.

https://hdl.handle.net/10356/45661

https://doi.org/10.32657/10356/45661

NANYANG TECHNOLOGICAL UNIVERSITY

ERROR-TOLERANT MULTIPLIER FOR HIGH SPEED APPLICATION

KHAING YIN KYAW

SCHOOL OF ELECTRICAL AND ELECTRONIC ENGINEERING

2011

ERROR-TOLERANT MULTIPLIER FOR HIGH SPEED APPLICATION

KHAING YIN KYAW

School of Electrical and Electronic Engineering

A thesis submitted to the Nanyang Technological University in partial fulfilment of the requirement for the degree of Master of Engineering

2011

TABLE OF CONTENTS

Abstract

With the advent of hand held computing devices that require functionality rivaling the desktop, low-power and high-performance systems have become very important. The transistor network contributes mostly to the overall power dissipation and is becoming a major obstacle in implementing those systems. Hence, the need for high performance basic sequential element with low-power dissipation is steadily growing. The aim of this project is to develop a new type of multiplier to fulfill this need.

In this report, for the first time, a multiplier design concept that engages accuracy as a design parameter is proposed. By introducing accuracy as a design parameter, we can break-through the bottleneck of conventional digital IC design techniques to improve on the performances of power consumption and speed. The two dimensional trade-off between power and speed becomes three-dimensional, i.e. power-speed-accuracy. To realize the design concept, digital multiplier circuits were studied and a novel mechanism is proposed in this work. The new type of multiplier adopting the proposed mechanism is named Error-Tolerant Multiplier (also called ETM). As illustration, the designs of 8-bit and 12-bit Error-Tolerant Multiplier, taken as examples, are described to elaborate on the design process and detailed circuit implementation of an ETM.

ETM is a novel design of low-power and high-performance multiplier based on the technique of statistically analysis on the error compensation for truncated partial products. It offers significant reduction in power consumption as well as for the improvement of circuit delays with small gate area usage. Both the output stage and performance of the new ETM was studied and compared with existing standard multipliers. All the tests performed in this research were conducted using the SPECTRE simulation tool of the CADENCE software. With this tool, substantial four to six times improvement was noted especially in the power-delay-product (PDP), when compared to conventional multiplier configurations.

Acknowledgement

I would like to take this opportunity to express my gratitude towards my supervisor, Associate Professor Goh Wang Ling for her constant guidance, assistance, encouragement and comments during the entire period of my postgraduate studies.

Special thanks also to my project co-supervisor, Professor Yeo Kiat Seng, for his invaluable advice, patience and guidance, for the successful completion of the project.

In particular, I like to thank Zhu Ning for his suggestions and time in supporting me when required.

Last but not least, I extend my gratitude to all the staffs of the Integrated Circuit and System (CICS) lab, and many others who facilitated smooth running of my project.

With all the helps, my experience in MEng studies has been the most rewarding and pleasant.

Chapter 1 Introduction

1.1 Background and Motivation

Multiplication is an important function in fundamental arithmetic operations. They are the most important elements in many Digital Signal-Processing (DSP) applications, e.g., convolution, Fast Fourier Transform (FFT) and arithmetic based systems. However, multiplier circuits are expensive and slow. The performance of many computational problems is often dominated by the speed at which multiplication can be executed [1]. Since multiplication dominates the execution time of most DSP algorithms, a high-speed multiplier is very desirable. Currently, the multiplication time is still the dominant factor in the determination of the instruction cycle time of a DSP chip.

With the ever-increasing quest for greater computing power on battery-operated mobile devices, there is a migration of design emphasis from conventional delay and area optimization to power dissipation minimization, while preserving the desired performance. A low-power design allows portable devices to operate a longer time with the same amount of battery. Besides the power dissipation concern, the operating frequency of the digital system is also a very important factor since it relates directly to the speed of a digital device. The study for the microprocessor shows that it has grown in performance and complexity at a steady and predictable pace in the early of 1970s but in the late 1980s, clock frequencies double every three years in the past decade and have reached into the GHz range [1]. The microprocessor speed for personal computer has moved from 0.1 MHz to over 4GHz over the past few years [2]. This shows that there is a great demand in the market towards high speed and low power circuit design.

One common technique for energy efficiency CMOS circuits is to reduce the supply voltage. It not only reduces the energy consumed per transition in a quadratic way but also reduces the leakage current. However, the delay of CMOS gates increases inversely with supply voltage [1]. The other drawback is the degradation of noise immunity of the circuits [3]. Hence the increasing noise sensitivity has become an important issue in the design of devices, circuits and systems [4]. Conventionally, one usually assumes that a usable circuit/system should always provide definite result. But in fact, such perfect operations are seldom needed in our non-digital world experiences. The data processed by many digital systems may already contained errors [5]. Because of advances in VLSI scaling and the near emergence of billion transistor chips, noise, process variations and spot defects will dictate that few such chips will be error-free. Hence the need for the error-tolerant circuit was foretold [6] in the 2003 International Technology Roadmap for Semiconductors (ITRS) [8].

Hence, unlike the conventional method, a new design for multiplier is proposed in this report. In addition to the power consumption and speed, *accuracy* is used as a parameter for the upcoming nano regime. The term *accuracy* is used to indicate how correct the output result that a multiplier can generate. This implies that the output of a multiplier is not necessary to be always correct.

When incorporates the error-tolerant circuit, a digital system is no longer totally "correct". Instead, certain errors may be generated in the output. This "imperfect" result seems to be accepted by end-users in the real non-digital world. This is attributed to "analog computation", which generates "good enough" results rather than totally accurate results [6]. It might be better for users to be more "generous" to accept certain amount of errors. Furthermore, in [8], it was quoted that "Relaxing the requirement of 100% correctness (also called as accuracy) in both transient and permanent failure of signals, logic values, devices, or interconnects may reduce the cost of manufacturing, verification and testing."

With an innovative and novel multiplication method adopted, the Error Tolerant Multiplier (ETM) is introduced. As the accuracy is used as a design metric, it would be traded for the improvement on speed and power consumption. The new ETM can achieve enormous improvements on speed performance and power consumption. As a trade-off, it cannot always maintain 100% accuracy as the conventional multipliers do. However, because of its outstanding advantages in power consumption and speed performance, this ETM has many potential applications in the domains where ultra-low power and/or super-high speed is required while the accuracy is not the main concern [6] [7]. Other applications include audio/video decoding, synthetic aperture radar (SAR) imaging, and so on.

1.2 Objective

This report aims to introduce a new digital design technique that is leading to an ETM, which takes accuracy into consideration in its design. Then the performance of the new multiplier technique will be analyzed and its simulation results will be compared with the conventional multipliers reported.

1.3 Organization of Report

The report is organized in the following manner. A brief literature review of conventional multiplier designs is provided in Chapter 2. Chapter 3 presents the new design technique of ETM on both the theoretical and circuit aspect. In Chapter 4, the performance of a new 8-bit multiplier is simulated and compared with the standard parallel multiplier reported in Chapter 2. The quality measurements described in Chapter 2 have been adopted to analyze all the multipliers discussed. It also presents a 12-bit multiplier using the proposed method. Simulated results are compared against with its 12-bit conventional counterpart and their performances are tabulated. In Chapter 5, the conclusions of this project are drawn and some recommendations are presented for future work.

Chapter 2 Overview of Multiplication

Multiplication can be considered as a series of repeated additions. For inputs that are M and N bits wide, the multiplication takes M cycles, using an N-bit adder. This shift-and-add algorithm for multiplication adds together M partial products [1]. The basic operations involved in multiplication are the generation and accumulation/addition of the partial products. As a consequence, these two major steps have to be optimized to speed up the entire multiplication process. The two main categories of binary arithmetic multiplication are the computation of unsigned and signed numbers.

2.1 Unsigned Multiplication

Real-time computer applications require fast multiplication. By utilizing AND gates and full adders, multiplication can be implemented on the processor in much the same way as hand multiplication: multiply each digit of the multiplier by the multiplicand to generate partial products, and sum up the respective partial products to generate the final result. Assume that X and Y are M and N bits wide respectively, where X is the multiplicand and Y is the multiplier. They can be expressed as [1]:

$$X = \sum_{i=0}^{M-1} X_i 2^i \tag{2.1}$$

$$Y = \sum_{j=0}^{N-1} Y_j 2^j \tag{2.2}$$

The product of X and Y is P and it can be written in the following form:

$$P = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} X_i Y_j \, 2^{(i+j)} \qquad (2.3)$$

Table 2.1 - General multiplication algorithm

				X_3	X_2	X_1	X_0	= X (multiplicand)
				Y_3	Y_2	Y_1	Y_0	= Y (multiplier)
				X_3Y_0	X_2Y_0	X_1Y_0	X_0Y_0	Partial Product 0
			X_3Y_1	X_2Y_1	X_1Y_1	X_0Y_1		Partial Product 1
		X_3Y_2	X_2Y_2	X_1Y_2	X_0Y_2			Partial Product 2
+	X_3Y_3	X_2Y_3	X_1Y_3	X_0Y_3				Partial Product 3
P_7	P_6	P_5	P_4	P_3	P_2	P_1	P_0	P = X * Y

To illustrate further, the multiplicand X and multiplier Y can be represented as:

X	Multiplicand	$x_{M-1}\ x_{M-2}\ x_{M-3} \ldots\ldots\ldots x_1\ x_0$
Y	Multiplier	$y_{N-1}\ y_{N-2}\ y_{N-3} \ldots\ldots\ldots y_1\ y_0$
P	Product (X*Y)	$p_{2N-1}\ p_{2N-2}\ p_{2N-3} \ldots\ldots p_1\ p_0$

Table 2.1 shows the process of multiplying two unsigned Binary-Coded Decimal (BCD) using the pencil-and-paper method [9] [10]. Each partial product has to be stored in the Arithmetic Logic Unit register, which occupies memory space until the final partial product is obtained. All the partial products are then added up to generate the final product. The partial storing of each partial product and the addition process involved make this method considerably inefficient.

2.2 Multiplication of Signed Numbers

The above procedure for multiplication works well for unsigned integers or unsigned fixed-point numbers. Generally for the multiplication of signed numbers, the negative number is first converted to its 2's complement representation. This is to make sure that all the partial products are positive.

Consider the multiplication of an n-bit X and −Y. Converting negative number of Y to its 2's complement format results in $(2^n - Y)$. The product in 2's complement context based on direct multiplication is

$$P' = X(-Y) = X(2^n - Y) = 2^n X - XY \tag{2.4}$$

This product differs from the expected result

$$P = -XY = 2^{2n} - XY \tag{2.5}$$

$(2^{2n} - XY)$ is the 2's complement representation of −XY. Therefore, P' deviates from the actual result P, by

$$P - P' = 2^{2n} - 2^n X = 2^n(2^n - X) \tag{2.6}$$

where $2^n(2^n - X)$ is the correction factor to be added. This is done by just taking the 2's complement of X and then shifting it to the left by n positions. This factor is then added to the pre-computed product of P'.

If both the multiplicand and multiplier are negative, their 2's complements will be multiplied. Consider that an n-bit −X is to be multiplied with an n-bit −Y. The product P' is given by

$$P' = (2^n - X)(2^n - Y) = 2^{2n} - 2^n X - 2^n Y + XY \tag{2.7}$$

However, the expected result is

$$P = XY \tag{2.8}$$

The difference is

$$P - P' = -2^{2n} + 2^n X + 2^n Y \tag{2.9}$$

The term 2^{2n} denotes a carry-out bit from the Most Significant Bit (MSB) that can be ignored. To get the correct result, correction factors for both multiplier and multiplicand should be added.

2.3 Types of Multiplier Architectures

The multiplier architecture can be generally classified into the following categories, namely serial, parallel and serial-parallel architectures.

2.3.1 Serial Multipliers

The serial multiplier uses a successive addition algorithm. It is simple in structure because both operands are entered in a serial manner. Therefore, the physical circuit requires less hardware and minimum chip area. However, the speed performance of the serial multiplier is poor due to the sequentially entered operands.

2.3.2 Parallel Multipliers

Three important criteria to be considered in the design of multipliers are the chip area, speed of computation and power dissipation. Most advanced digital systems incorporate a parallel multiplication unit to carry out high-speed mathematical operations. A microprocessor requires multipliers in its arithmetic logic unit and a digital signal processing system requires multipliers to implement algorithms such as convolution and filtering.

Today, high-speed parallel multipliers with much larger area and higher complexity are used extensively in Reduced Instruction Set Computers (RISC), Digital Signal Processing (DSP), and graphics accelerators. Some examples of the parallel multiplier are the array multipliers such as the Braun Multiplier [9] and Baugh-Wooley Multiplier [15], as well as the tree multipliers like the Wallace Multiplier [1]. Array multipliers have a more regular layout while tree multipliers generally are faster. Parallel multiplier presents a high-speed performance but it is expensive in terms of silicon area as well as power consumption. This is because for parallel multipliers, both operands are input to the multiplier in a parallel manner. As a result, the circuitry is much larger in area and more complex as compared to serial multipliers.

2.3.3 Serial-Parallel Multipliers

The serial-parallel multiplier serves as a good trade-off between the time consuming serial multiplier and area consuming parallel multipliers. These multipliers are used when there is a demand for both high speed and small area. In a device using the serial-parallel multiplier, one factor is entered serially and the other is stored in parallel with a fixed number of bits. The resultant enhancement in the processing speed and the chip area will become more significant when a large number of independent operations are performed.

A tidy breakdown of different types of digital multipliers is portrayed in Figure 2.1. Contemporary digital signal processing algorithms for image processing and telecommunications applications are increasingly dependent on matrix- and vector-like arithmetic. In addition, given the exponentially rising processor performance requirement and the arithmetic-intensive nature of many applications such as speech and image processing, waveform shaping, infinite impulse response digital filtering, channel equalization, networking, multimedia and computer vision, high-speed multipliers will continue to be in high demand. Thus, in the following sections, only different types of parallel multipliers will be discussed in details.

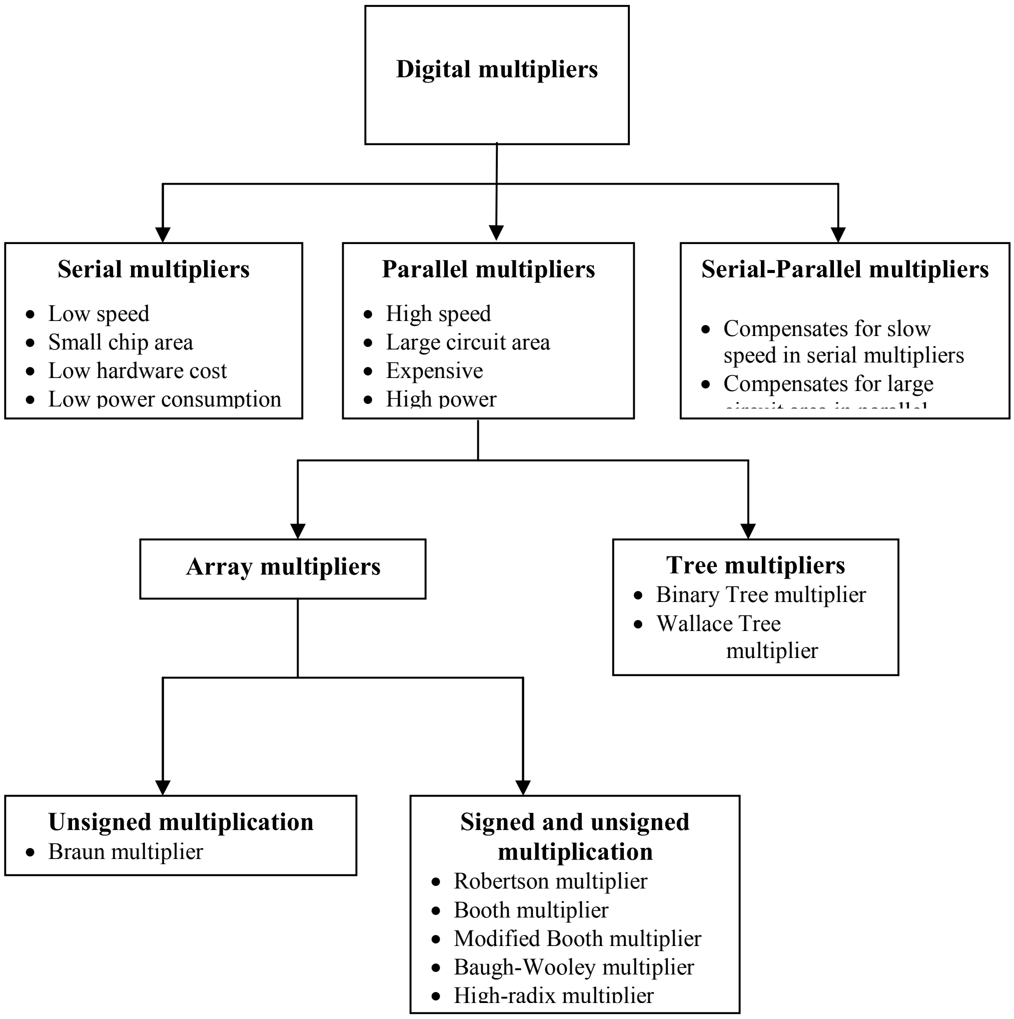


Figure 2.1 Types of digital multipliers.

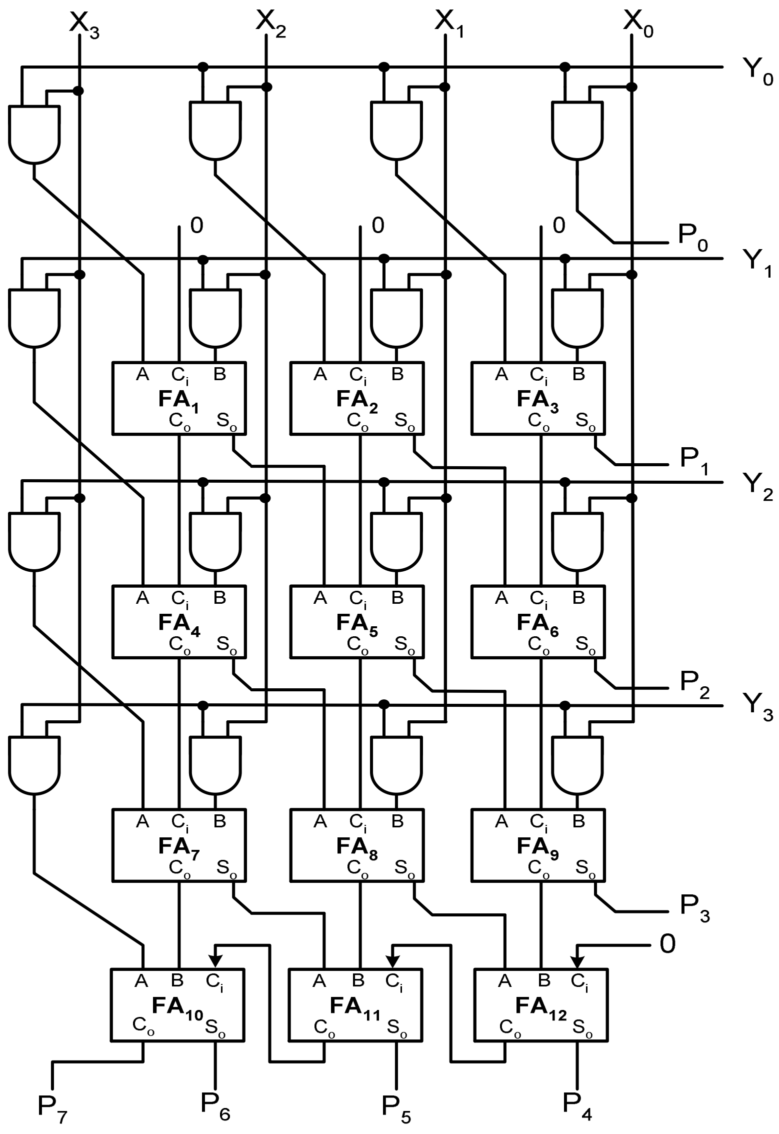
2.4 Braun Multiplier

Braun Edward Louis first proposed the Braun multiplier in 1963 [9]. It is a simple parallel multiplier that is commonly known as the Carry Save Array Multiplier. This multiplier is restricted to performing multiplication of two unsigned numbers. It consists of an array of AND gates and adders arranged in an iterative structure that does not require logic registers. This is also known as the non-additive multiplier since it does not add an additional operand to the result of the multiplication.

2.4.1 Architecture of Braun Multiplier

An $n \times n$-bit Braun multiplier requires $n(n - 1)$ adders and n^2 AND gates [9] [12]. An efficient implementation of the Braun multiplier is a regular layout of the adder array as shown in Figure 2.2 and the internal structure of the full-adder is depicted in Figure 2.3 [11]. This makes Braun multipliers ideal for Very Large Scale Integration (VLSI) and Application Specific Integrated Circuit (ASIC) realization.

Each of the X_iY_j product bits is generated in parallel with the AND gates [9]. Each partial product can be added to the previous sum of partial products by using a row of adders. The carry-out signals are shifted 1 bit to the left and added to the sums of the first adder and the new partial product. The shifting of the carry-out bits to the left is done by a Carry Save Adder (CSA). As the carry bits are passed diagonally downward to the next adder stage, there is no horizontal carry propagation for the first four rows. Instead, the respective carry bit is ‘saved’ for the subsequent adder stage. Ripple Carry Adders (RCA) are used at the final stage of the array to output the final result.



X: 4-bit multiplicand

Y: 4-bit multiplier

P: 8-bit product of X and Y

$P_n = X_iY_j$ is a product bit

Figure 2.2 Schematic diagram of a 4 × 4-bit Braun multiplier.

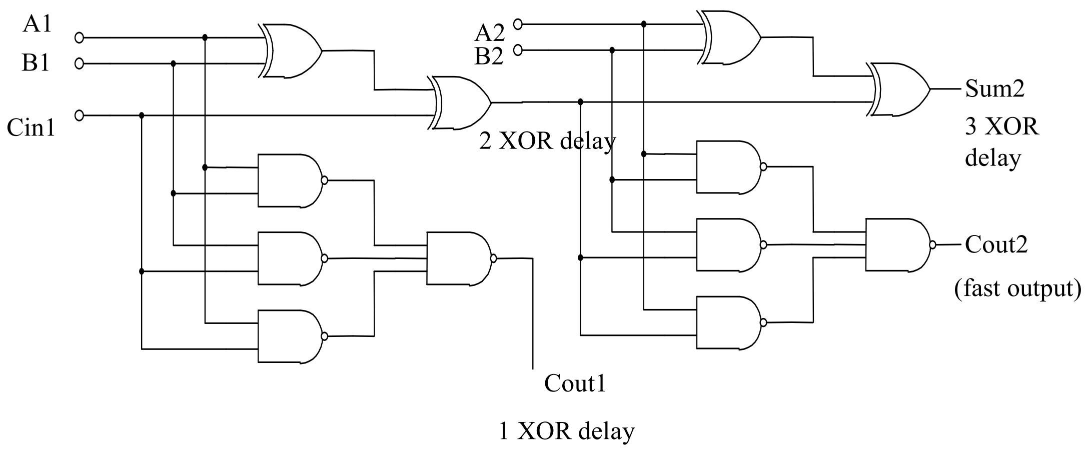


Figure 2.3 Two optimally interconnected full adders.

2.4.2 Performance of Braun Multiplier

The Braun multiplier performs well in terms of speed, power and area for unsigned operands, which are less than 16 bits. Besides, it has a simple and regular structure as compared to other multiplier schemes. However, the number of components required building the Braun multiplier increases quadratically with the number of bits. This makes the Braun multiplier inefficient and hence it is rarely employed while handling large operands. Another pitfall of the Braun multiplier is its potential susceptibility to glitching problems at the last stage of the full adders due to the exploitation of the Ripple-Carry-Adders (RCA).

2.4.3 Speed Consideration

The delay of the Braun multiplier is dependent on the delay of the full adder cell and also the final adder in the last row. In the multiplier array, a full adder with balanced carry and sum delays is desirable because the sum and carry signals are both in the critical path. The speed and power of the full adder are very important for large arrays.

The worst-case multiplication time of a Braun multiplier can be expressed as [12].

$$t_{Braun} = (n - 1)t_{carry\text{-}save} + t_{AND} + (n - 1)t_{Ripple\text{-}Carry} \qquad (2.10)$$

where $t_{carry\text{-}save}$ is the time required to generate Carry-out (C_{out}) or Sum (S_{out}) at the output after the inputs are supplied to a CSA; $t_{Ripple\text{-}Carry}$ is the time taken for the Carry-out (C_{out}) or Sum (S_{out}) to be generated at the output after the inputs are supplied to a RCA; and t_{AND} is the delay of an AND gate.

2.5 Baugh-Wooley Multiplier

The Baugh-Wooley multiplier is an enhanced version of the Braun multiplier. It is designed to cater for the multiplication of both signed and unsigned operands, which are represented in 2's complement number system [13]. The partial products are adjusted so that the negative signs are moved to the last steps, which in turn maximize the regularity of multiplication array.

2.5.1 Architecture of Baugh-Wooley Multiplier

The architecture of the Baugh-Wooley multiplier is also based on the carry-save algorithm. It inherits the regular and repeating structure of the array multiplier. The structure of a 4 × 4-bit 2's complement multiplier is shown in Figure 2.4, with the cell number representing the type of basic cell [9] [14] in Figure 2.5.

2.5.2 Performance Consideration

The area and power consumption of a number of multiplier structures vary with the number of bit of operands and also the layout strategies. Increasing regularity and locality at the silicon level reduces the power consumption in a standard-cell based design flow.

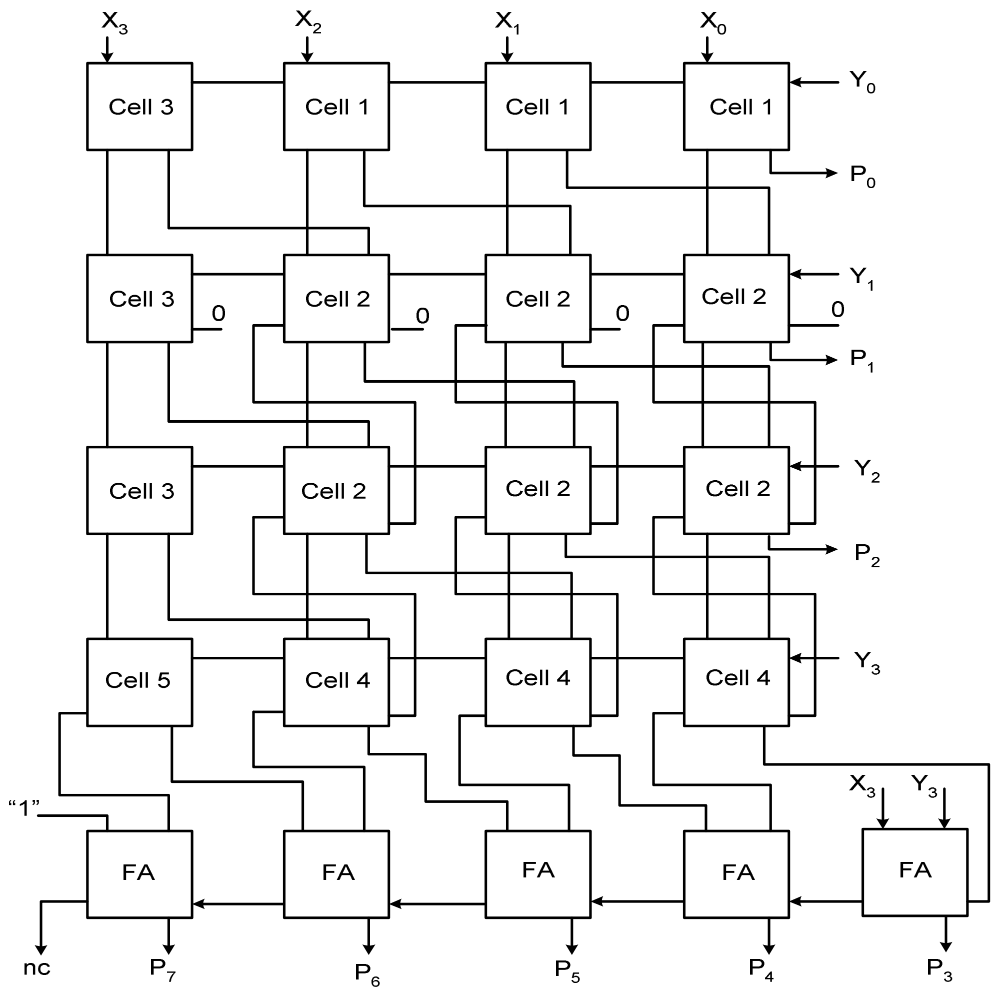


Figure 2.4 Schematic circuit of a 4 × 4-bit Baugh-Wooley multiplier.

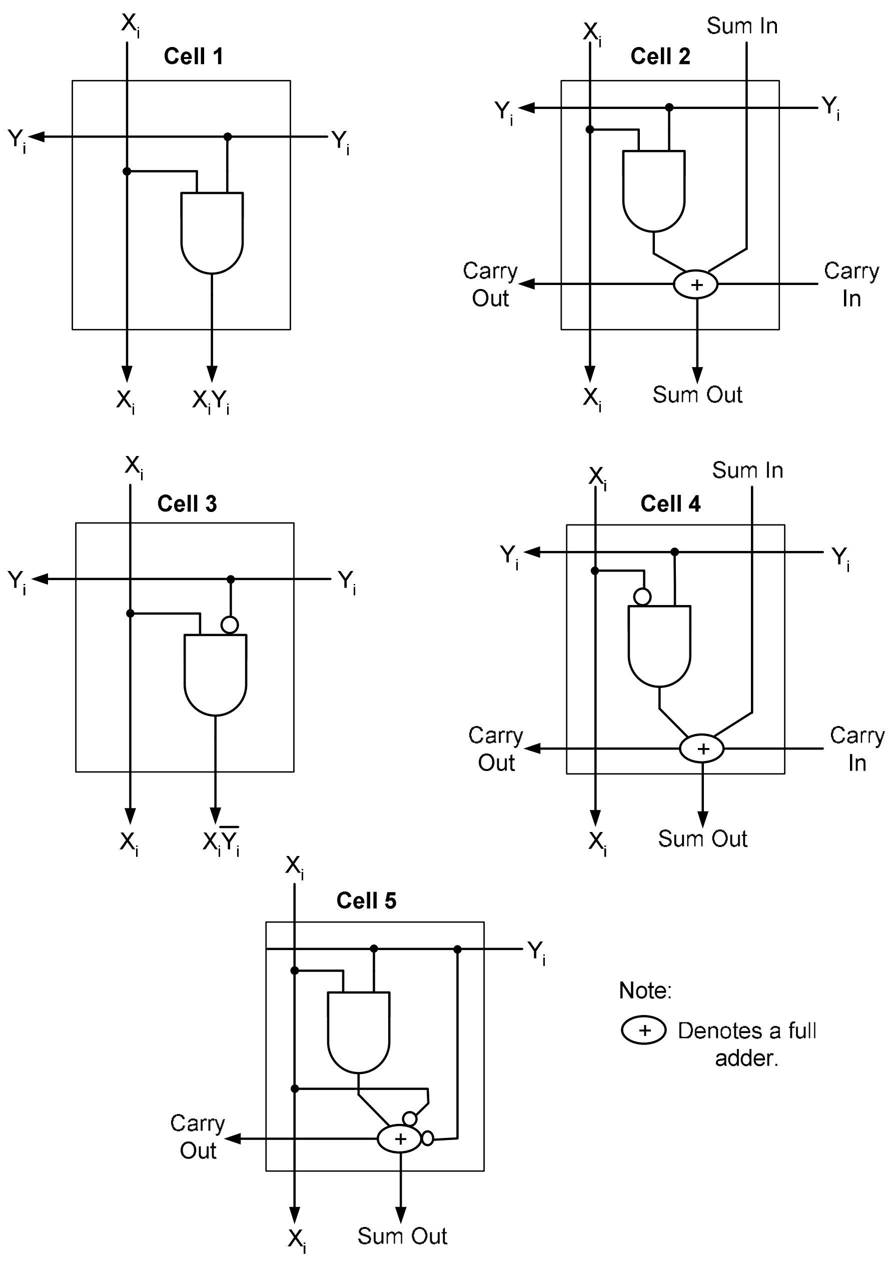


Figure 2.5 The five basic building blocks of the Baugh-Wooley multiplier.

2.6 Booth Multiplier

Area-efficient and fast multipliers are the essential blocks for high-performance computing. Therefore, multipliers should be small enough so that a larger number of them can be integrated on a single chip.

Conventional array multipliers like Braun multiplier and Baugh-Wooley multiplier achieve comparatively good performance but require large silicon area, which is in contrast with the add-shift algorithms that require less hardware with poorer performance [1]. The Booth multiplier makes use of the Booth encoding algorithm to reduce the number of partial products by considering two bits of the multiplier at a time, thereby achieving a speed advantage over other multiplier architectures. This algorithm is valid for both signed and unsigned operands.

2.6.1 Booth's Algorithm

A.D. Booth proposed the Booth algorithm (also known as radix-2 algorithm) in 1951 for multiplication that accepts numbers in 2's-complement form based on radix-2 computation. It can handle signed binary multiplication by using 2's complement representation [15]. This increases the complexity in the form of the storage of signs of operands in auxiliary circuits.

2.6.2 Standard Radix-2 Booth Multiplication Rules

The rules for a standard radix-2 Booth recoding are as follow [15]:

a) Append a zero to the right of the Least Significant Bit (LSB) of the multiplier;

b) Inspect groups of two adjacent bits of the multiplier, starting with the LSB and the appended zero;

- If the pair is 00 or 11, then shift the partial product one bit to the right.

- If the pair is 01, then add the multiplicand to the partial product and shift the partial product one bit to the right.
- If the pair is 10, subtract the multiplicand from the partial product and shift the new partial product to the right by one bit.

c) Proceed with overlapping pairs of bits such that the MSB of a pair becomes the LSB of the next pair. In this manner, one bit of the multiplier number is eliminated in each pass through the algorithm.

d) When the last pair of bits is examined, the partial product is updated following the rules except that no shift is performed.

2.6.3 Booth Encoder

The Booth encoder implements the Booth encoding of three bits of multiplier as well as handling the sign extension logic. Each encoder is dedicated to one partial product in the array. Since there is a circuit for each of the five possible generated partial product signals, one and only one signal is high during the steady state operation. The carry propagation circuits are independent of the partial product circuits and they do not share any inputs.

In the conventional modified Booth algorithm encoder, three signals X_j, $2X_j$ and M_j, are generated from three adjacent bits, b_{j-1}, b_j and b_{j+1} for selecting a partial product that is 0, +A, −A, +2A, or −2A. In this case, A is a multiplicand of n-bit width. The X_j and $2X_j$ signals show whether the partial product is doubled and an active M_j means that the negative partial product should be used. The structure of Booth encoder is shown in Figure 2.6 and the partial product generator for this implementation is depicted in Figure 2.7 [16].

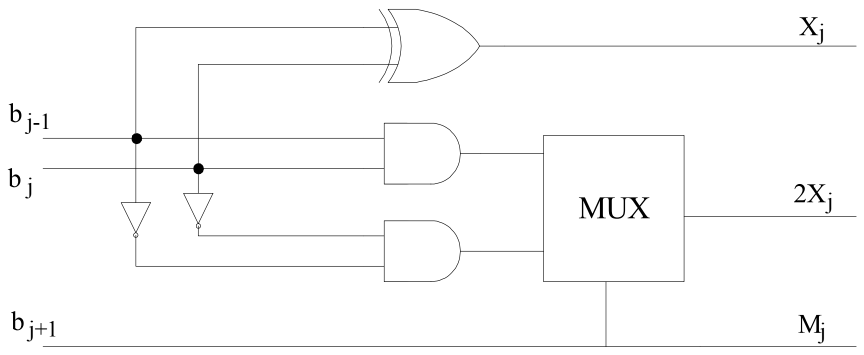


Figure 2.6 Structure of Booth encoder.

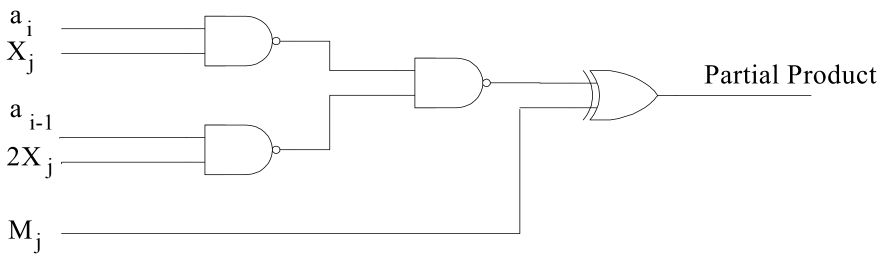


Figure 2.7 Conventional partial product generator.

2.7 Wallace Tree Multiplier

Booth's algorithm effectively reduces the number of partial products to half. However, for large-operand multipliers such as 32-bit and above, the partial products are more than 16 bits and are still unacceptably large in number. The number of partial products can be reduced with Wallace tree multiplication algorithm. It employs multiple input compressors, which are able to accumulate several numbers of partial products concurrently [17].

C.S. Wallace proposed the Wallace Tree multiplier in 1964, which can handle the multiplication process for large operands. This is achieved by minimizing the number of partial product bits in a fast and efficient way by means of a carry-save adder tree constructed from one-bit full adders.

The main disadvantage of Wallace tree algorithm is that the architecture exhibits some irregularities in the layout since it has a relatively complicated interconnection scheme. In general, its multiplication process can be summarized as follows:

a) After generating the partial products, a set of counters reduces the partial product matrix but it does not propagate the carries;
b) The resulting matrix is composed of the sums and carries of the counters;
c) Another set of counters then reduces this matrix and the whole process continues until a two-row matrix is generated; and
d) The two rows summed up with a final adder, preferably a carry propagate adder. This method takes the advantage of the carry save architecture to avoid the carry propagation until the final adder. In this scheme, the number of levels is crucial and determines the speed of the multiplier.

2.7.1 4:2 Compressors

Effective reduction of the propagation stage can be achieved by employing 4:2 compressors instead of the 3:2 compressors as in the conventional Wallace tree algorithm. The advantage of tree multipliers is that their speed increases in a logarithmic scale with the operand length, as opposed to the case of iterative arrays, where the speed increases linearly with the size of the operands. This idea was originated from A. Weinberger of IBM in 1981. The 4:2 compressor is able to yield a much more regular structure than the 3:2 counter because it can reduce four inputs of the same weight to two.

The 4:2 compressor and its equivalent building blocks are depicted in Figures 2.8 (a) and 2.8 (b), respectively [9]. It receives four numbers (In_1, In_2, In_3, and In_4) and a carry-in (C_{in}), compresses them and then generates two numbers (S and C) and a carry-out (C_{out}). The 4:2 compressor is also dubbed the 5:3 compressors if C_{in} and C_{out} are taken into account. C_{in} is the carry input from the lower bit and C_{out} is the carry output to the higher bit. C_{in} and C_{out} have the same weight and C_{out} is independent of C_{in}. Hence, carry propagation does not occur.

The sum is $S = In_1 \oplus In_2 \oplus In_3 \oplus In_4 \oplus C_{in}$ (2.11)

The carry C is related to the sum of the first adder, In_4 and C_{in}. The C_{out} is the carry signal of the addition of In_1, In_2 and In_3. By this arrangement, the sum can be obtained via four XOR gate delays ($S = [\ [\ (In_1 \oplus In_2) \oplus In_3\ ] \oplus In_4\ ] \oplus C_{in}$), which is identical to the result in the Wallace tree structure using 2-layer carry save adders. Therefore, it can be rearranged to be

$$S = [(In_1 \oplus In_2) \oplus (In_4 \oplus C_{in})] \oplus C_{in} \quad (2.12)$$

With this arrangement, three XOR gate delays are involved. Hence, the speed advantage can be obtained. The use of the 4:2 compressor permits the reduction of vertical critical path while the path involving the carry propagation (horizontal path) is not changed. However, horizontal propagation is faster and limited to one bit per level.

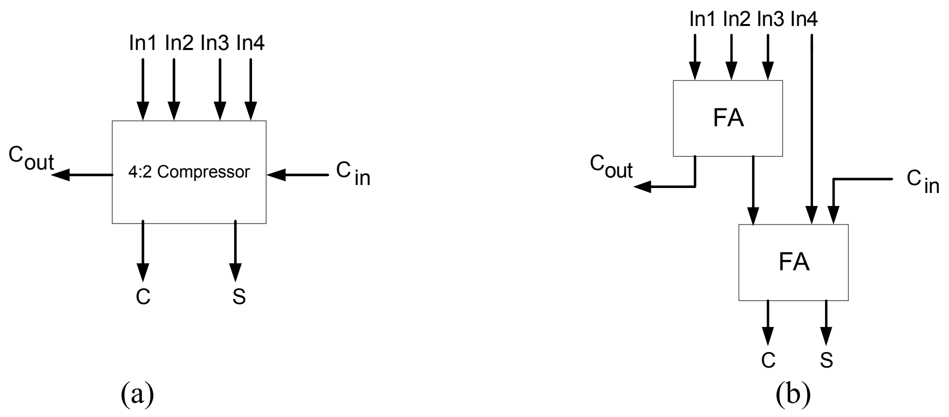


Figure 2.8 A 4:2 compressor: (a) schematic diagram and (b) equivalent circuit.

2.7.2 Wallace Tree Construction

The Wallace tree can be constructed in many ways. Basically, the Wallace tree uses 4:2 compressors, 3:2 compressors, full adders and half adders to compress a 12-partial product tree. Each 4:2 compressor takes in four bits from the same position ‘j’, one bit from the previous position ‘j–1’ (which is the carry-out of the compressor in the previous position) and outputs one sum bit in the position ‘j’ and two carry-outs to the next position ‘j+1’.

The partial-sum adders are arranged in a treelike fashion, reducing both critical part and the number of adder cells needed. Figure 2.9 illustrates how the 4 bits x 4 bits Wallace tree multiplier is implemented.

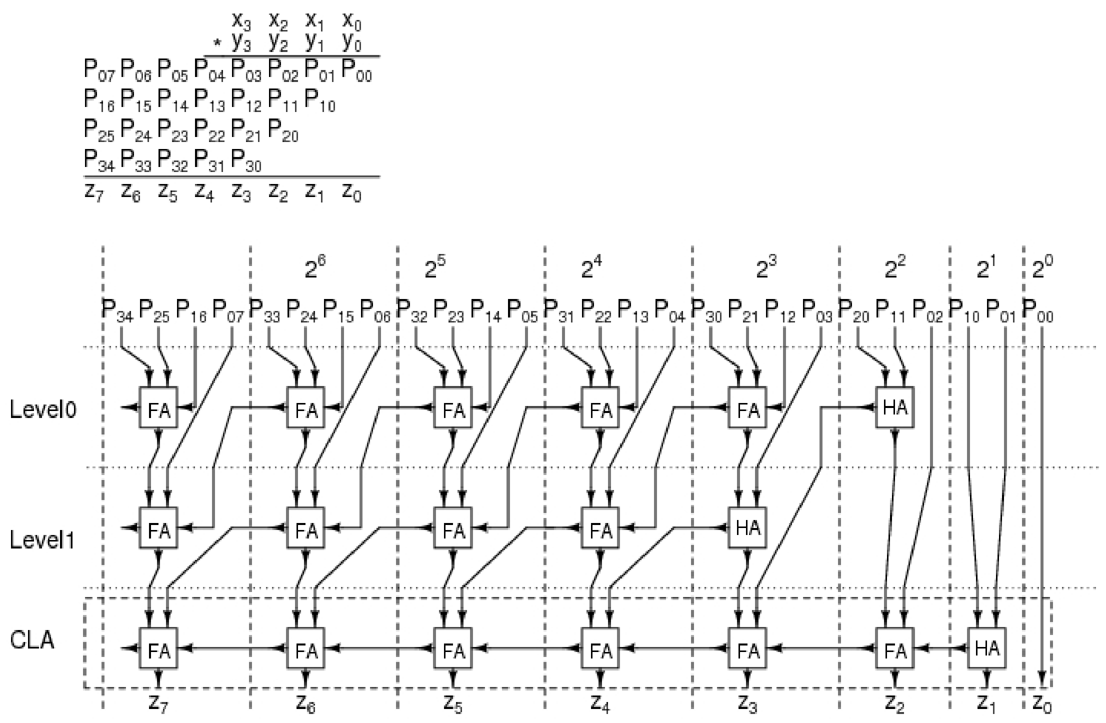


Figure 2.9 Transforming a partial-product tree into a Wallace tree for a 4 x 4 multiplier.

The challenge is to realize the complete multiplier with a minimum depth and a minimum number of adder elements. There are two types of adder used here: Full-adder which takes three inputs and produces two outputs and half-adder which takes two input bits in a column and produces two outputs. The tree multiplier realizes substantial hardware savings for larger multipliers. The propagation delay is reduced as well [1]. In Figure 2.9, the Wallace tree only needs 18 adders which comprise of 15 full-adders and 3 half-adders. The typical array multiplier might need 64 adders for signed multiplication.

2.8 Summary of conventional multipliers

This chapter provides a glimpse of the multiplication techniques, types of multiplier architectures and a wide-ranging review of different multiplication algorithms. A myriad of eminent architectures of multipliers, including Braun multiplier, Baugh-Wooley multiplier, Booth's multiplier and Wallace tree multiplier have been covered.

Braun multiplier is useful for applications where the area is of major concern and the importance of high-speed is downplayed. This multiplier offers the advantage of high regularity in its structure. However, its drawback is that it is only meant for unsigned multiplication. In addition, its hardware requirement is more than the other multiplier algorithms when handling operands that are greater than 16 bits.

The main motivation behind the realization of the Baugh-Wooley multiplier is to multiply negative numbers based on the 2's complement number representation. The Baugh-Wooley multiplier resembles the Braun multiplier in layout regularity and enhancement methods. Like the Braun multiplier, it is very efficient in handling multiplicands of less than 16 bits but becomes less efficient as the operands get larger.

Ever since the Booth's algorithm is introduced, it has always been used and developed because it renders the advantage of reducing the number of partial products. This algorithm is useful for applications where the hardware cost is of major concern. It can reduce the delay and implementation cost by a factor of two.

The Wallace tree multiplier is a fast multiplier with irregular routing. Wallace tree multiplier is widely used for high-speed applications. Since the wiring of Wallace tree is considerably complex, it may add to a significant wiring overhead, which may cause an increase in the design time.

Adopting hybrid architectures such a combination of modified Booth recoding technique and Wallace tree structure can further enhance the multiplier speed. The modified Booth recoding technique is capable of reducing the number of partial products to half and the Wallace tree speeds up the process of accumulating parallel partial products in a tree-like fashion. Table 2.2 shows the summary and comparison of the performance of conventional multipliers described in previous sections.

Table 2.2 - Summary of conventional multipliers' performance comparison

Braun multiplier	Baugh -Wooley multiplier	Booth's multiplier	Wallace-tree multiplier
Efficient for operands less than 16 bits.	Efficient for operands less than 16 bits.	Able to handle the large multipliers by reducing the number of partial products, the delay and implementation cost by a factor of two.	Efficient for operands less than 32 bits.
Simple and regular layout structure.	Simple and regular layout structure.	Increase the complexity due to the storage of signs of operands in auxiliary circuits.	Irregular layout, wasted chip area and complex wiring. Increase design time.
Useful when area is major concern and high speed is required.	Enhanced version of Braun multiplier.	Useful for applications where the hardware cost is of major concern.	Widely use for high speed applications and substantial hardware saving in large multipliers.
Only meant for unsigned multiplication.	Able to do 2's complement number representation.	Valid for both signed and unsigned operands.	Valid for both signed and unsigned operands.

2.9 Quality Measures

Factors which are desirable in many kinds of digital circuits can be spelled follows:

- High speed
- Low power consumption
- Robustness and noise stability
- Small area and less number of transistors
- Supply voltage scalability
- Low glitch probability
- Large internal race immunity
- Insensitivity to clock edge
- Insensitivity to process variables
- Less internal activity when data activity is low

According to the requirements of the system, the designer has to consider all parameters, which decide on a suitable multiplier structure. What makes this decision even harder is that usually most of these parameters are not independent from each other. Trade-off between desired parameters makes these decision a multi-dimensional optimization problem for high performance systems [17]. A multi-dimensional optimization problem for a non-linear system that usually has hundreds of variables is unfortunately impossible to solve within the limited design time.

In this project, which is the design of high performance and low power digital circuit, the quality measures are rise time, fall time, propagation delay, power dissipation, power-delay product and the number of transistors which can be translated into silicon area.

2.9.1 Propagation Delay (PD)

Propagation delay (input-to-output) is the time delay after the propagation of the signal whereby the output is considered stable. It is equals to the time it takes for the output to change before the occurrence of the next input. Hence, the propagation delay time is measured from input to output Q.

The propagation delay differs for the low-high transitions and the high-low transitions. So propagation delay of the multiplier is defined by the maximum value of the two delays:

$$\mathbf{t_{PD} = Maximum\ (t_{phl}, t_{plh})} \tag{2.13}$$

2.9.2 Power Dissipation

Power dissipation is one of the primary considerations in electronic circuit design. This is because high power consumption will result in a low battery life, low reliability, and high cost due to additional necessary cooling mechanism.

The power dissipation in any CMOS digital circuit is given by

$$\mathbf{P_{average} = P_{static} + P_{sc} + P_{dynamic}} \tag{2.14}$$

There are three main sources in the average power consumption of a circuit [18].

- Static power dissipation – Contains of two main components: power dissipation due to leakage currents in parasitic diodes in CMOS structures and power dissipation due to sub-threshold currents when the input is not a complete low (for NMOS transistors) or a complete high (for PMOS transistors).
- Short circuit power dissipation - This occurs due to the finite slope of the input where the PMOS and NMOS sections are on simultaneously. The short-circuit power dissipation is about 20% to the total power dissipation.
- Dynamic power dissipation – Power dissipation due to the charging and discharging of the output load and the internal capacitances.

$$\mathbf{P_{dynamic} = \alpha * V_{dd}^{2} * C_L * f} \tag{2.15}$$

where C_L is the capacitance of the load and f is the frequency of the clock. The factor α is the average switching activities of the input relative to the clock.

The dynamic power dissipation is the largest contributor, i.e. 90% to the overall power dissipation. Therefore, reducing the power dissipation of basic digital circuit can significantly reduce the overall power dissipation of a VLSI system [1]. It should be noted that the power dissipation of a multiplier depends not only on the supply voltage, clocking frequency, switching nodes, but also on the input sequence. The power dissipation is therefore measured with different input sequences that give different output switching activities, α. The data activity rate α represents the average number of output transitions per cycle [19].

Two extreme cases of the maximum and minimum power dissipation are when $\alpha = 1$ and $\alpha = 0$. The power dissipation at $\alpha = 1$ is when the input, switches at every phase. In the case of $\alpha = 0$, the power dissipation is measured under two conditions: one where the input stays at 'high' logic level and the other at 'low' logic level. In most design, although $\alpha = 1$ reflects the maximum power dissipation, $\alpha = 0$ can also dissipate more power, depending on the design structure, contributing by the static power and short circuit power dissipation.

2.9.3 Power-Delay-Product (PDP)

In digital circuit design, there is always the trade-off between power dissipation and propagation delay. Normally higher power is required to achieve higher speed and vice versa. Therefore, the power-delay-product (PDP) is used as a Figure-of-Merit measurement.

The lowest PDP gives the optimal energy utilization at a given clock frequency. It is obtained as a product of averaging the power consumption and the propagation delay (D-Q Delay) as in equation shown.

$$\mathbf{PDP = P_{average} * PD} \qquad (2.16)$$

The power dissipation of all data circuits is taken into account when calculating the average power consumption. PDP sensitivity is, very often, influenced by scaling of supply. Generally, although the dynamic power is largely dependent on the supply voltage, stray capacitance as well as the frequency of operation, the overall supply voltage has the largest effect. Therefore, with the overall supply voltage lowered, the power dissipation of the circuits can be largely reduced. However, there are various problems associated with lowering the voltage.

During the scaling process, the supply voltage has to decrease to limit the field strength in the insulator of the CMOS as well as to relax the electric field from the reliability point of view. This leads to a tremendous increase in the propagation delay of the BiCMOS gates, especially at the very low supply voltage. Also, scaling down the supply voltage causes the output voltage swing of the BiCMOS circuits to decrease. Moreover, external noise does not scale down as the device features size reduces, giving rise to adverse effects on the circuit performance and reliability.

Another major device problem associated with the simple scaling lies in the threshold voltage restriction. In CMOS circuitry, the drivability of MOSFETs will decrease, signals will become smaller, and the threshold voltage variations will become more limiting. Since the reduction of the threshold voltage causes a drastic increase in the cut-off current, the lower limit of the threshold voltage should be considered by taking into account the stability of the circuit operation and power dissipation. Hence, a compromise is needed between the threshold voltage and supply voltage in order to have both low-power and high-speed operation [8].

2.9.4 Area

The silicon area is also a key parameter as it involves cost and packaging density. Circuits that occupy a large silicon area will incur more cost and hence not suitable for portable systems. The main figures of merit are the number of transistors (NT), and the sum of the areas of all the transistors. In this project, the total area measures are estimated by the summation of the product of the individual transistors' gate lengths and widths used by the multipliers in Cadence environment.

Chapter 3 Proposed Error-Tolerant Multiplication technique

3.1 Introduction

Technology scaling-includes feature size, voltage, and clock frequency scaling. It has brought tremendous improvements in performance over the past several decades. Unfortunately, these make digital VLSI design /system significantly more susceptible to hardware faults in the future, resulting in reduced system reliability. In anticipation of the reduced reliability that further technology scaling will bring, designers and architects have recently focused on several important fault tolerance issues. Areas of focus include characterizing error susceptibility [20], and developing low-cost error detection [21] [22] and recovery techniques [23]. Fundamental to all such reliability research is the definition of correctness of the circuit/system [24]. It is a system that is able to continue operating properly in the event of failure(s) of some of its components, at the cost of hardware, time or information quality [25], which, in some cases, cannot be justified [8]. Relaxing the requirement of 100% correctness for devices and interconnections may dramatically increase the effective yields and bring down the costs of manufacturing by recovering the chips that were previously deemed useless [6] [26].

In the past, researchers have made very strict assumptions about the design correctness. Traditionally, a usable system/circuit is said to be correct only if the obtained result and output-state is numerically perfect. In such case, correctness requires precise numerical integrity at the architecture/design level, a fairly strict requirement. An interesting question is that must we require strict numerical correctness for circuit design to be correct? In many systems, even if execution is not 100% numerically correct, it can still function correctly from the user's perspective. Although such numerically error does not pass the muster of design-level correctness, they may be completely acceptable at the user or application level [24]. Hence, whether an error is intolerable or benign may depend on the level of a system.

Case Study I - Testing Methodology for Error-tolerant based Systems

Although the testing methodology is not the main concern of our work, the ideas and analysis methods proposed and studied in [6] give a better view of error-tolerant digital integrated circuit design which is the main contribution of this thesis.

In conventional digital circuit testing techniques, it targets to test all the possible faults that may occur in the circuit. However, in the error-tolerance circuitry, the coverage of testing is reduced to only unacceptable faults that are predetermined by designer/user base on the system/application intended be used in. For every error-tolerance supported system, there is a maximum acceptable error-rate specified by the designer/user. Those faults whose error-rates are higher than the maximum acceptable system error-rate are considered as unacceptable accuracy while the rest faults are expected to be tolerable by the system.

In [6], it presents a case study of the applicability of the Error-Tolerance technique. It is illustrated with respect to a digital telephone-answering device (DTAD). There are two main components, the microcontroller and the flash memory which is assumed to be defective. When the user listens to the recorded speech, the microcontroller extracts the encoded data stored in the memory, decodes the data and finally recovers the speech. However, the quality of output of the system is degraded because the flash memory employed in DTAD is defective. If such an “imperfect” output is acceptable to the user according to certain measurement standard, this system can be regarded as an error-tolerant system.

The fault model considered in [6] is the multiple stuck-at fault models. The erroneous bits in the memory are either stuck-at-1 or stuck-at-0. Faults are randomly allocated through the memory based on the uniform distribution. According to [6], twenty different fault densities between 0% and 1% are simulated.

It is also presented about the relationships between the defect density (error-rate), the acceptable performance and the effective yields. The defect density is defined as the ratio between the number of faults and the size of the flash memory. The acceptable performance is referred to the performance (subjective or objective) that is acceptable to the user according to certain measurement standard. The effective yield represents the yield in manufacturing process due to the employment of Error-Tolerant technique. The yield-factor (*d*) is the expected percentage of faulty flash memories whose test scores are greater than *T,* a threshold value to partition the acceptable memories from unacceptable ones.

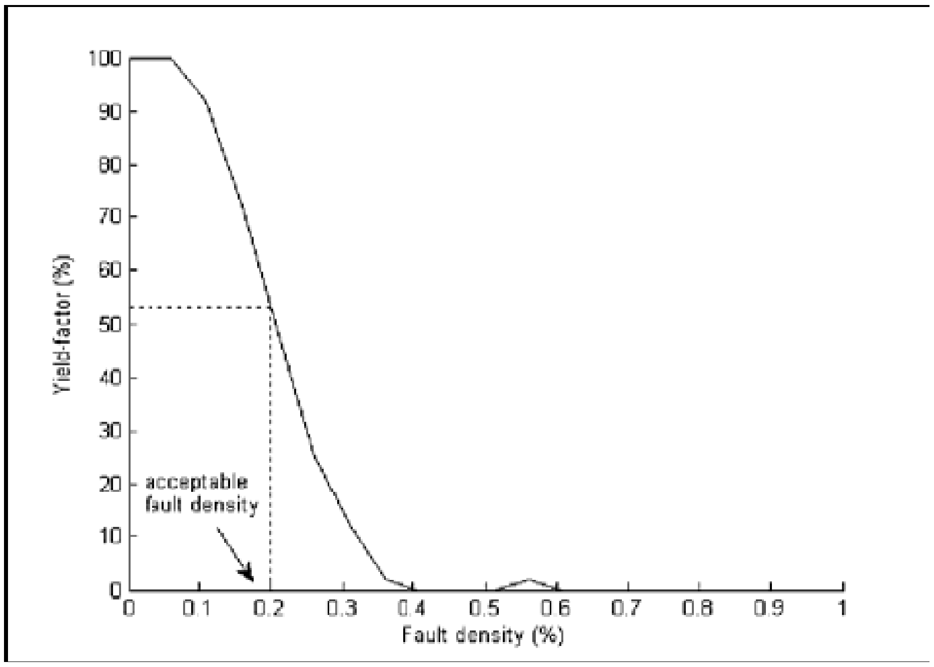


Figure 3.1 Yield factor Vs fault density (%) [6].

In figure 3.1, it is clearly seen that the yield-factor decreases as fault density increases. At the fault density of 0.2%, the yield-factor is about 53%. To measure the quality of the performance of the DTAT, a kind of test whose guidelines from a mean opinion score (MOS) [27] is conducted to get the simulation results. The qualitative interpretations of the MOS are 1 (bad), 2 (poor), 3 (fair), 4 (good), 5 (excellent). If the acceptable threshold value, T is set to 3 (fair) as the lowest acceptable MOS, the corresponding acceptable fault density for the DTAT is 0.20%. That means when 0.20% of all the bits in the flash memory are defective, the whole system is still in an acceptable and tolerable range for the end-user.

In general, more errors are acceptable when handling large domain such as multi-media systems that process content consumed by human senses, such as sound, images, smells and touch. Much of the raw data in these domains is sampled, digitized, quantized, encoded via a lousy compression algorithm, processed and transmitted. Eventually this data is decoded and transformed into analog signals that seem to be quite acceptable to most people. Examples of ET circuits are portions of MPEG and JPEG processors [6].

Compared with many other arithmetic operations, multiplication is a time consuming and power hungry [28]. The critical paths dominated by digital multipliers often impose a speed limit on the entire design. Hence, VLSI design of high-speed multipliers, with low energy dissipation, is still a popular research subject [29]. The proposed Error-Tolerant Multiplier (ETM) is a digital multiplier that does not always yield correct results but is still usable in some circuit systems by generating "acceptable" result. According to this definition, there can be numerous ways to implement an ETM. In this chapter, a methodology that serves to provide an investigation in the emerging research area is presented.

Prior to discussing the ETM, the exact definitions and explanation of some commonly used terminologies are given as follows [5]:

- *Overall error (OE)*. It is defined as the difference between the correct result and obtained result. It can be obtained by using the following equation: $OE = |R_c - R_e|$, where R_c denotes the correct result and R_e is the result obtained by the proposed multiplier (all the results are represented as decimal numbers).

- *Accuracy (ACC)* – In the scenario of the error-tolerant design, the accuracy of a multiplier is used to indicate how "correct" the output of a multiplier is for a particular input. It is defined as: $ACC = (1 - OE / R_c) \times 100\%$. Its value ranges from 0% to 100%, only if $R_c > 0$ and $OE > 0$. According to the mathematical expression, it can be seen that the accuracy of multiplier is depending on the output result that is not a constant. As an appropriate correctness, it should accommodate all valid numerical outputs. At the same time, it is important to recognize that not all valid outputs are of equal value; instead, there are varying degrees of solution quality across the designs' output [30]. Hence, the accuracy of a multiplier can be regarded as a variable with respect to the output/input pattern.

- *Minimum acceptance accuracy (MAA).* Although some errors are allowed to exist in the output of an ETM, the accuracy of an acceptance output should be "good enough" (higher than a threshold value) to meet the requirement of the whole design. Minimum acceptance accuracy is the threshold value for the obtained results of the proposed multiplier. If they are higher than the minimum acceptance accuracy (*MAA*), they are called accepted results. The value of minimum acceptance, in other words, the threshold value, is often defined by the customers/designers according to the specific applications.

- *Acceptance probability (AP).* It is the probability that the accuracy of a multiplier is higher than minimum acceptance accuracy (*MAA*). It can be expressed as $AP = P(ACC > MAA)$ and its value ranges from 0 to 1. Since the input patterns of a digital system are often regarded as random signals, the accuracy of the multiplier can also be taken as a random variable. This parameter is usually used as an important metric indicating the accuracy performance of an ETM.

In the past few years, significant reduction in the chip area and an associated increase in the speed of operation of the multipliers have been achieved through increased device density by taking advantage of advancements in VLSI technology. Unfortunately, this approach has reached a stage of diminishing returns and further improvements in the design of parallel multipliers will occur only if major breakthroughs are achieved in the technology [31]. Hence, new method can be envisioned by allowing the tradeoff between minimizing the use of costly device density with relaxing the result accuracy.

In order to ensure the performance matrices, a new method is proposed whereby n x n bit multiplier process the multiplication only for N bits of most significant products and truncate least significant M bits of the partial products in order to produce a final product with reduced precision. This is illustrated in the next section. In order to achieve a lower average error with the truncation of the partial product bits, an error compensation circuit, in which area overhead is much lower than the truncated part is usually added [26]. It would greatly reduce the hardware cost without sacrificing much accuracy. It is widely used in current digital signal processing applications such as finite-impulse response (FIR) filtering and discrete cosine transform (DCT) transforms [31], [32], [33], [34], and [35]. Due to the error compensation value (ECV), two categories can be classified: constant [30] [31] [32] and adaptive [33] [34] [35] [36]. The constant ECV for the truncated partial product is generated independent of truncated digits and is fixed for a given accuracy. Thus the implementation of this method is simple with the penalty of large truncation errors. On the contrary, the adaptive ECV is determined by the most significant bits of the truncated partial products.

Very often in DSP applications, fixed-point arithmetic is used and typically N-bit signals are multiplied by N-bit coefficients. Since a uniform word length is required, the 2N-bit products obtained must be quantized to *N* bits by eliminating the *N* least-significant bits (LSB) by the newly proposed scheme. Close examination of the standard parallel multiplier reveals that approximately 50% of the chip area is needed for the generation of the *N*- LSB. It could be achieved by omitting about half the adder cells but significant error in the required product would be introduced. The design strategy is to omit about half the adder cells, as suggested, but introduce appropriate compensation to the retained adder cells. In this way, an area-efficient parallel multiplier, referred to as truncated multiplier, is obtained in which the error involved is kept to a minimum.

3.2 Proposed Multiplication Algorithm

There are three major steps to any multiplication. In the first step, the partial products are generated. In the second step, the partial products are reduced to one row of final sums and one row of carries. In the third step, the final sums and carries are added to obtain the result. The delay is mainly attributed to generation of the partial product rows and the carry propagation chain along the critical path from the Least Significant Bit (LSB) to the Most Significant Bit (MSB). Moreover, a significant proportion of the power consumption of a multiplier is due to the glitches that are also caused by the carry propagation [1]. Therefore, if the partial product array can be reduced rapidly and the carry propagation can be eliminated or curtailed, a great improvement in both the speed performance and power consumption can be achieved. In this section and for the first time, innovative and novel multiplication algorithm that can attain faster speed and lower power consumption is proposed. This new multiplication algorithm can be illustrated via an example shown in Figure 3.2.

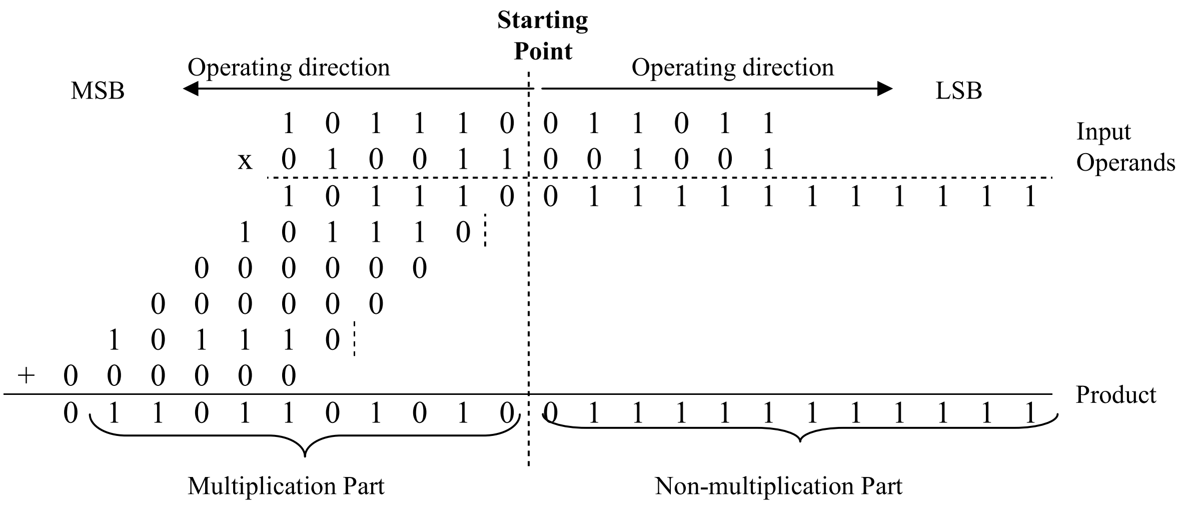


Figure 3.2 Multiplication algorithms for ETM.

First, the input operands are split into two parts: a multiplication part that includes a number of higher order bits and a non-multiplication part that is made up of the remaining lower order bits. The length of each part needs not necessary to be equal. The multiplication process starts from the starting point shown in Figure 3.2 (joining part of the two parts) towards the two opposite directions simultaneously. In the example, the two 12-bit input operands, the multiplicand “1011 1001 1011” (2971) and the multiplier “0100 1100 1001” (1225), are divided into two equal-sized parts, each of which contains 6 input bits.

For the higher order bits of the input operands that fall into the multiplication part, the operation is formed from right to left (LSB to MSB) and normal multiplication method is applied. For the example shown in Figure 3.2, assuming that the joining point of the part is in the middle, the higher order operands (6 MSB bits) follow the conventional multiplication algorithm. Hence the product of “101110” with “010011” gives the result “01 10110 1010”.

For the lower order bits of the input operands that fall into the non-multiplication part, a special mechanism is applied. In this part, no partial product rows will be generated or the carry propagation path no longer exists. To reduce the overall error caused by eliminating the partial-product rows, a special strategy is adopted and its operational process is described as follows: check every bit position from left to right (MSB to LSB) of the non-multiplication part, if either of the two operand bits is “1” or both operand bits are “1”, the checking process is brought to an end and from this bit onwards, all the bit positions are set to “1”. In the event that both operand bits are “0”, the corresponding product bit is set to “0”. In this way, the overall error generated due to the elimination of partial-products can be reduced to minimum. In the example, at the sixth position, the two input bits are both equal to “0”. Hence the corresponding result bit is set to “0”. At the fifth LSB bit (2nd position from the starting point), as the multiplicand bit is “1”, the corresponding result bit is set to “1” and all the remaining results on its right are also set to “1”. The product generated in the non-multiplication part is therefore “0111 1111 1111”, which belongs to incorrect part.

The final result of the complete multiplication is equal to “011 0110 1010 0111 1111 1111” (3581951). This is the result obtained using the proposed multiplication algorithm. On the other hand, the correct result of this multiplication, which can be derived using the normal multiplication process, is “011 0111 1000 1000 1011 0011” (3639475). Thus the accuracy of the multiplier with respect to these two input operands is:

$$ACC = \{1- \{(3639475 - 3581951)/ 3639475\}\}* 100\% = 98.42\%.$$

In this new method, the partial product rows and carry propagation exist only in the multiplication part. The multiplication part is constructed in the conventional way because the higher order bits of a result need to be made as accurate as possible, as they play a more important role (have higher weights) than the lower order bits. In conventional method, there should be more than 6 partial-product rows that would lead to more XOR delays due to its lower order bits. This is the fact that one partial-product row would bring relatively higher delay and larger silicon area, that would result in an extra hardware cost.

Many of the recent papers [37], [38], [39], [40], [41], [42] have been focusing on rapidly reducing the partial-product arrays by using some kind of circuit optimization and identifying the critical paths and signal races. In other words, the goals have been to optimize the partial-product rows down to the final sums and carries for the final accumulation. In our proposed algorithm, by eliminating the partial products and carry propagation path in the non-multiplication part (LSBs) and performing the multiplication for MSBs simultaneously, the overall delay time is significantly reduced and so is the power consumption as compared with conventional approaches, which will be discussed in more details in chapter 4.

3.2.1 Relationships between *AP, MAA,* Dividing Strategy and Size of the Multiplier

As mentioned in section 3.1, there is a minimum acceptable accuracy (*MAA*) associated with an ETM. If a result obtained by the multiplier has an accuracy that is higher than the *MAA*, this result is taken as the acceptable result. Upon further evaluation of the proposed multiplication algorithm, it can be seen that the accuracy of the ETM is closely related to the number of bits to be used and its input pattern. Assume that the inputs of ETM are random numbers; there exists a probability of obtaining an acceptable results (i.e., the *AP*). The dividing strategy is the strategy for deciding the sizes of both the multiplication and non-multiplication parts. In this section, relationships between the *MAA*, the *AP*, the dividing strategy, and the size of multipliers are investigated.

To implement ETM in real application, dividing strategy (i.e. starting point) needs to be chosen such that accuracy is within the specification, .i.e. it has to satisfy AP, MAA, etc. A MATLAB program is engaged to simulate the proposed multiplication algorithm. By checking the output results, the relationship between *MAA* and *AP* can be derived as depicted in Figure 3.3. In this study, simulation of multiplier with different dividing strategies was performed. In Figure 3.3, the 3 curves represent 3 different dividing strategies for 8-bit multiplier, each of which has been assigned a name “N-M” where “N” denotes the size of the accurate part and “M” is for the size of the inaccurate part. For example, “6-2” means the size of the accurate part of the operands is 6-bit and that of inaccurate part is 2-bit. For the input patterns, over 6500 inputs were randomly selected from all possible input patterns.

It can be deduced from Figure 3.3 that the different dividing strategy leads to different accuracy performance for 8-bit multiplier. When the size of the accurate part is made larger, the AP of the multiplier will increase. But the delay path will be too long since the sum and carry generators will take place.

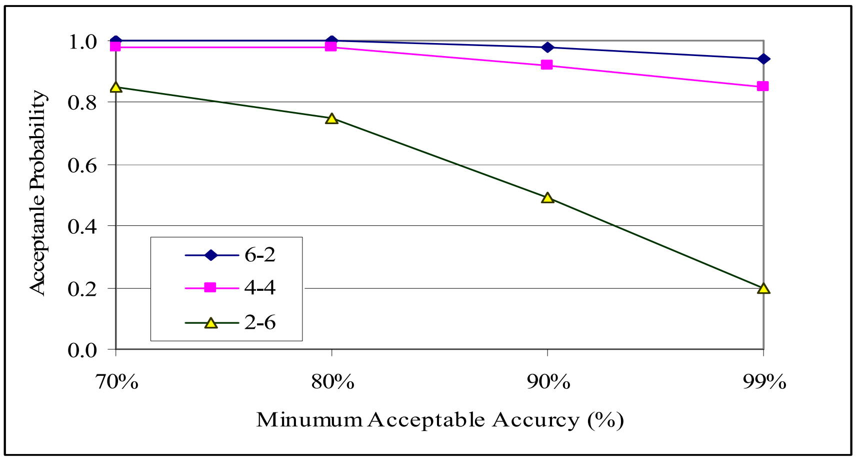
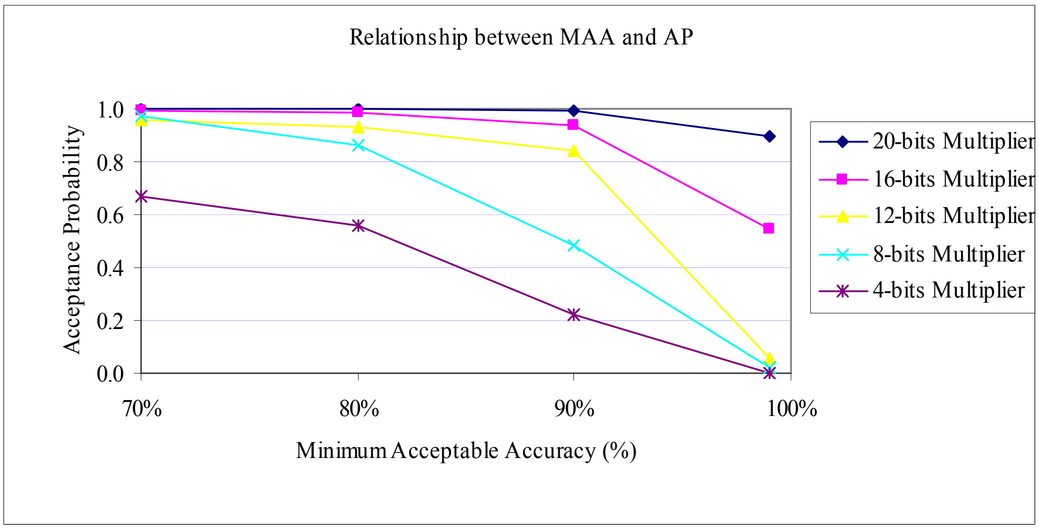
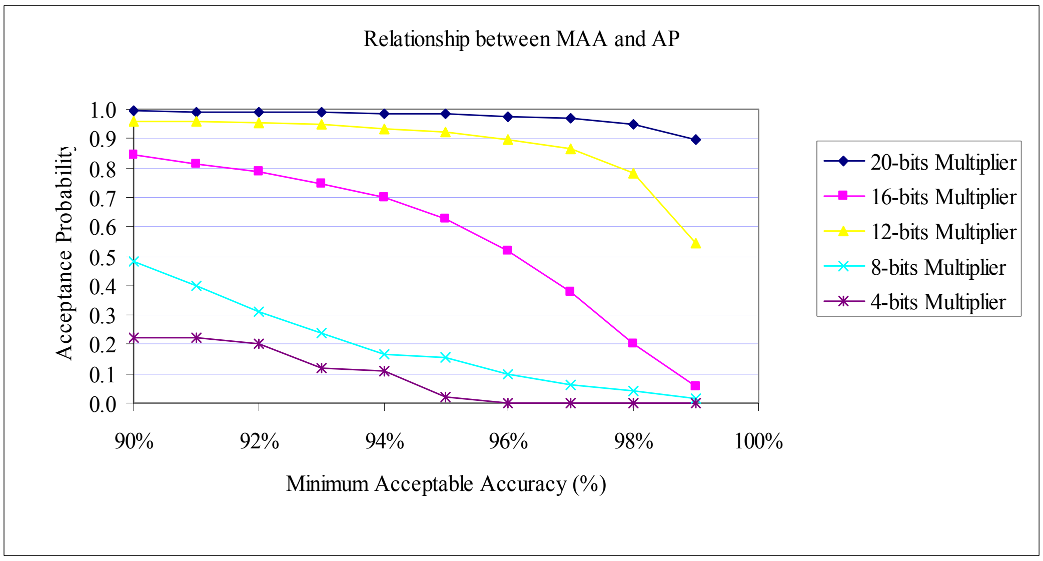


Figure 3.3 Relationship between AP and MAA for an 8-bit multiplier.

Situations where the requirement on accuracy is somewhat relaxed are also investigated. In Figure 3.4, the 5 curves represent 5 different sizes of multipliers and they are associated with different *MAA*'s, 70%, 80%, 90%, 99%, respectively. It can be deduced that the lower the *MAA* set, the higher the *AP* for the multiplier. This means that if some degree of error can be permitted, the chances of getting acceptable results will be very high and the likelihood increases when the size of the multiplier increases. When the size of the multiplier becomes larger, the *AP* value will also increase. Therefore the proposed ETM works better in larger multipliers especially those of 16-bit, 20-bit or even higher, which are used in certain multimedia and audio processing applications.



(a)



(b)

Figure 3.4 Comparison of multipliers with different sizes adopting proposed algorithm: (a) the *AP* comparison for *MAA* 70%, 80%, 90% and 99%; (b) the *AP* comparison for *MAA* 90% to 99%.

In this study, all the sizes of the multipliers have been divided into two equal-sized parts. Hence, the multiplication part contains 6 most significant bits (MSB) and non-multiplication part has 6 least significant bits (LSB) of the inputs for a 12-bit multiplier. For the 20-bit multiplier, the 10 higher order bits of both multiplier and multiplicand will fall into the multiplication part and the remaining 10 lower order bits (LSB) will fall into the non-multiplication part. It should be noted that those unacceptable results often occur when the input patterns are of small numbers in most significant bits (MSB). This is because small numbers will be calculated only in the non-multiplication part of the multiplier. The proposed ETM is thus especially suitable for large input patterns such as 10-bit or more multipliers. Over 65,000 inputs were randomly selected from all possible input patterns in the range of 0 ~ (2^{20} – 1) for the evaluation of the 20-bit multipliers. As for the rest of the multipliers, over 6,500 input patterns were also selected randomly.

Case Study II - How much more fault resilient are acceptable at the application level?

The answer to this question is application dependent, and primarily depends on how numerically exact a program's outputs need to be. In [23], it investigates and conducts the detailed study about the definition of program correctness viewed from the application's standpoint. For instance, applications that process human sensory and perception information are highly fault resilient at the system level. Other examples are multimedia applications, artificial intelligence programs (e.g., Inference, machine learning), which have become increasingly important recently and applications that process human sensory and perception information. Traditionally, it is said to be correct only if architectural state is numerically perfect on a cycle-by-cycle or module-by-module basis. Although such numerically faulty executions do not fulfill the architecture-level correctness, they may be completely acceptable at the user or application level. Hence, whether a fault is intolerable or benign may depend on the level of abstraction at which correctness is evaluated. In general, more faults are acceptable at higher levels of abstraction, i.e. at the application level, compared to lower levels of abstraction, i.e. at the architecture level.

The degree of error tolerance varies across the application levels. In [23], a special program called "soft computation" is used to study the outputs that occurred frequently and interpreted qualitatively by the application users. It also explores the soft computation output with 9 different bench tests listed in Table 3.1. There are three tests for use in the multimedia domain, three for the artificial intelligence (Al) domain, and three other tests that belong to the SPECInt CPU2000 benchmarks, in which integer performance testing component of the SPEC (Standard Performance Evaluation Corporation) test suite are being used for audio, image, and video decompression, respectively.

Table 3.1 - Numerical and qualitative outputs computed in nine benchmarks. The fidelity metrics in the last column is used to quantify solution quality [23]

Bench	Num. out	Qual. Out	Fidelity Metric
Multimedia			
G.721-D	Decompressed audio data file	Perceived audio	Segmental Signal-to-Noise Ratio (SNRseg)
JPEG-D	Decompressed image data file	Perceived image	Peak Signal-to-Noise Ratio (PSNR)
MPEG-D	Decompressed video data file	Perceived video	Peak Signal-to-Noise Ratio (PSNR)
Artificial Intelligence			
LBP	Network believe values	Web Page Class Types	% Classification Change
SVM-L	Support Vector Model	Test Data Class Types	% Classification Change
GA	Thread Schedule	-	% Schedule Length Change
SPECInt CPU2000			
164.gzip	Compressed file	-	Compression Ratio
256.bzip2	Compressed file	-	Compression Ratio
175.vpr	Cell placement	-	Consistency Check

In [23], it is also conducted the fault injection experiments to study the detailed fault susceptibility at the application level as compared with the architecture level. In the multimedia and Al benchmarks, 45.8% of fault injections that lead to architecturally incorrect execution are found correct under application-level correctness. In the SPECInt CPU2000 benchmarks, 17.6% of architecturally incorrect faults produce the acceptable results at the application level. Based on those experiments, it is concluded that a significant number of faults were previously thought to cause erroneous execution are in fact completely acceptable to the user. Besides, application-level correctness can be defined by choosing the minimum fidelity given in Table 3.1.

Besides the “soft computation”, it has also studied the outcome of recovery experiences in the program output. In Figure 3.5, it breaks down the outcome of all architecturally visible fault injections when they are simulated to program completion. For each benchmark, the faults are injected into the physical register file, fetch queue, and IQ separately in a group of 3 bars labeled "R," "F," and "I," respectively. Each bar contains 6 categories. The first category, labeled "Architecture," indicates the program outputs that pass architecture-level correctness. The next two categories, labeled "Application-High" and "Application-Good," indicate the additional program outputs that are acceptable under application-level correctness only, assuming the "high" and "good" thresholds described in Table 3.1.

The category labeled "Incorrect" indicate outcomes that are either invalid or unacceptable under both architecture and application-level correctness. Finally, the last two categories indicate experiments that fail to complete during functional simulation due to an exception or a program lockup (labeled "Crash") or early program exit with an error (labeled "Terminate"). In Figure 3.5, it is also reported the last 3 groups of bars which are the average breakdowns for the multimedia, Al, and SPEC benchmarks, respectively [23].

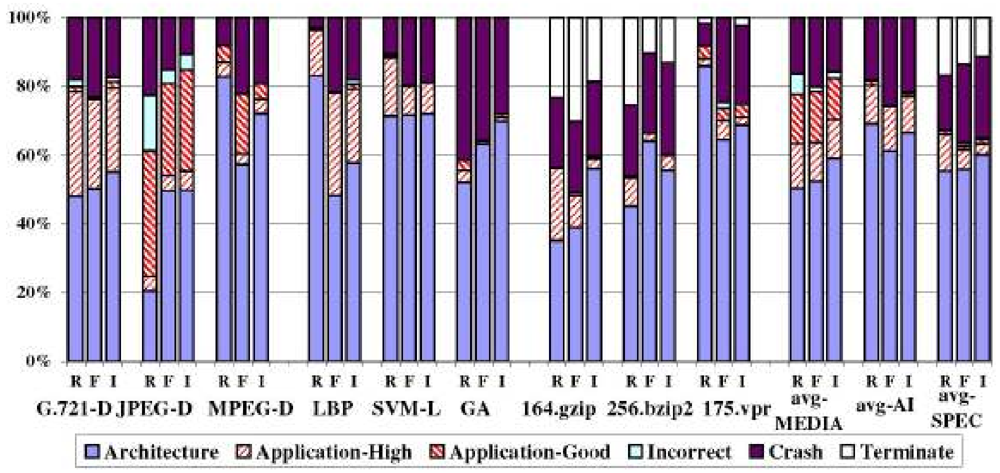


Figure 3.5 Program outcomes breakdown for architecturally visible fault injections: correct at the architecture level ("Architecture"), correct at the application level ("Application-High" and Application-Good"), unacceptable ("Incorrect"), exception or program lockup ("Crash"), and early program exit ("Terminate") [23].

In [23], it also illustrated that the outputs of equal or higher quality than the minimum fidelity satisfy the user's requirement and are considered as correct, while outputs of lower quality than the minimum fidelity are considered as incorrect. An important question, then, is how do can the minimum fidelity threshold can be determined against which application-level to be measured? Unfortunately, minimum fidelity thresholds are extremely user-dependent. In practice, different users may desire different levels of solution quality and the same user may be able to live with varying levels of solution quality under different circumstances. Hence it is impossible to define one threshold that applies universally. Instead, users should be allowed to select the threshold that best fits their correctness requirements. This provides designers with the unique opportunity to tradeoff solution quality for fault tolerance, depending on how good a solution the user needs.

3.2.2 Proposed Hardware Design

The block diagram of the proposed hardware design of the ETM is provided in Figure 3.6. The most straightforward structure consists of two parts: a multiplication part and a non-multiplication part. However, it is necessary to determine if there is logic “1” in any bit positions of the first half of the higher order bits in both the input patterns, A and B. This is to check if the operands are large enough to use the proposed ETM. Otherwise, it will proceed to generate the product using conventional multiplication method. The control block is used to generate the control signals to determine the working mode of the multiplier.

The multiplication part, which contains n to m bits, is constructed and conventional multiplication such as the Parallel Multiplier, Booth Multiplier and Wallace Tree Multiplier, etc can be used. In this thesis, the standard parallel multiplier is employed to generate the higher order bits of the product. The carry-in of the non-multiplication part is connected to ground. The multiplication part is used to compute the higher order bits of the operands. The non-multiplication part, whose size is m-bit, considers two blocks: a partial-product-free multiplication block and a check block. The partial-product-free block generates the product bits on its lower order bit positions. In the section 3.3, the design of an 8-bit multiplier, taken as an example, is described to elaborate on the design process and detailed circuit implementation of an ETM.

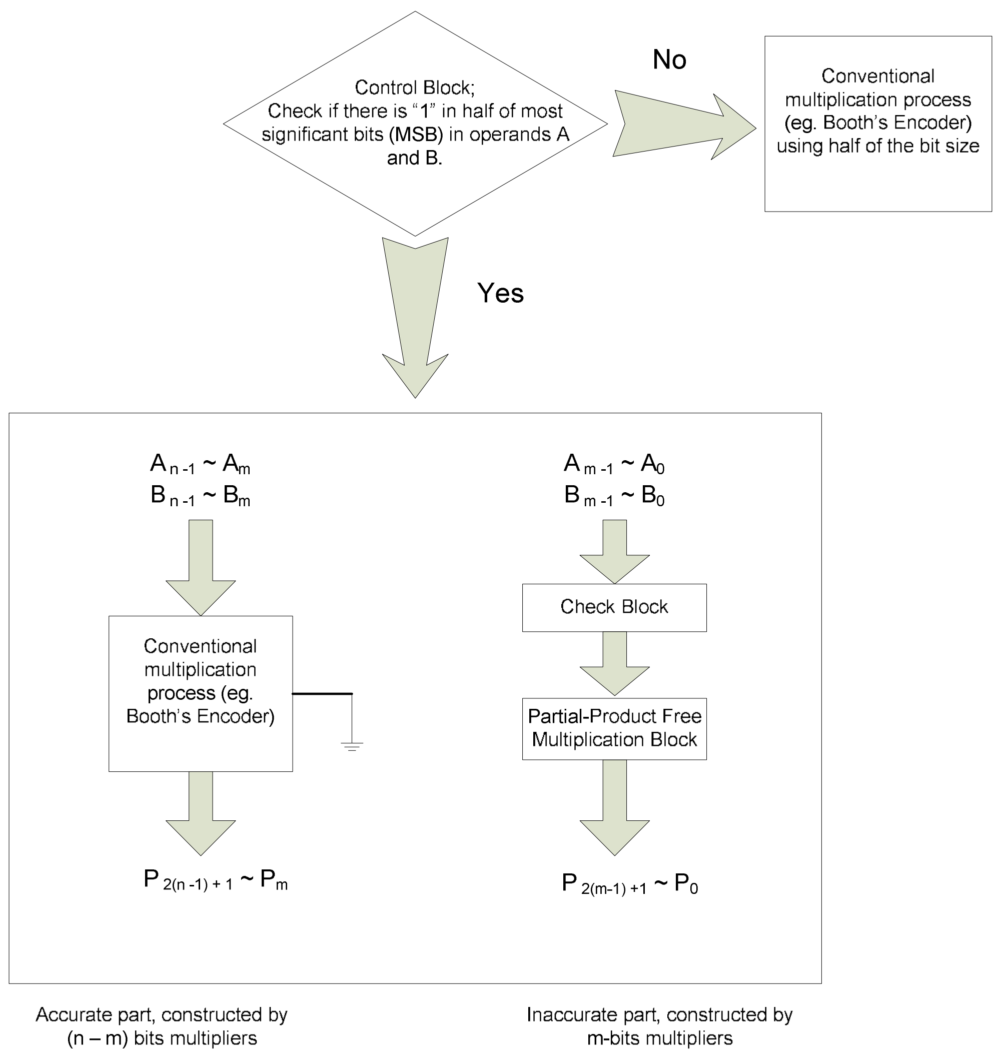

Control Block;
Check if there is "1" in half of most significant bits (MSB) in operands A and B.
No
Conventional multiplication process (eg. Booth's Encoder) using half of the bit size
Yes
$A_{n-1} \sim A_m$
$B_{n-1} \sim B_m$
Conventional multiplication process (eg. Booth's Encoder)
$P_{2(n-1)+1} \sim P_m$
$A_{m-1} \sim A_0$
$B_{m-1} \sim B_0$
Check Block
Partial-Product Free Multiplication Block
$P_{2(m-1)+1} \sim P_0$
Accurate part, constructed by (n – m) bits multipliers
Inaccurate part, constructed by m-bits multipliers

Figure 3.6 Block diagram of the hardware design of ETM where n denotes the size of the multiplier and m denotes the size of inaccurate part.

3.3 Design and implementation of an 8-bit ETM

3.3.1 Strategy of Dividing the Multiplier

The first step to design the proposed ETM is to divide the multiplier into two parts in a specific manner. The dividing strategy depends on the requirements, in terms of accuracy, speed and power.

First of all, the accuracy performance of the multiplier should meet the threshold preset by the designer/customer that best fits his correctness requirement. While minimum fidelity thresholds are user dependent, a specific set of thresholds should be chosen via simulation or conducting the experiments. Although it is necessary to study the extend to which the acceptable accuracy level in each application requirement is certainly an important direction of the research, it is beyond the scope of this work.

As an example, for a specific application, one may require the minimum acceptable accuracy to be 98%, with an acceptance probability of 0.99. With such criteria, the proposed multiplier should be divided in such a way that 98% accuracy can be attained for at least 99% of all possible inputs.

Secondly, the delay of the proposed multiplier is defined as $T_d = T_1 + \mathrm{Max}\ (T_2, T_3)$, where T_1 is the delay in the control block, T_2 is the delay in the multiplication part and T_3 is the delay in the non-multiplication part. With proper dividing strategy, a designer can make T_2 approximately equal to T_3 and hence achieving the optimal time delay.

Thirdly, due to the simplified circuit structure and the elimination of switching activities in the non-multiplication part, putting more bits in this part yields more power saving.

Having considered the above and using the proposed ETM method, an 8-bit multiplier is first examined. It is divided such way that 4 bits of the input patterns are assigned to the multiplication part and the other 4 bits in the non-multiplication part. As a result, it generates 8 bits in both the multiplication and non-multiplication parts to yield a 16 bits product.

3.3.2 Design of the Control Block

In the proposed architecture, it is required to check if the sizes of the operands are "large enough" to operate in ETM mode. This is to improve the accuracy of the final product. Although the ETM performs well in terms of speed, power and "good enough" accuracy for large number of multiplications, the accuracy becomes "degrading" and inefficient for small operands. This has been illustrated in section 3.2.1. A special mechanism is therefore employed to check small or large numbers before operating in ETM mode. The function of the control block is to determine either a conventional multiplier or ETM should be engaged to generate the final product. In this example of an 8-bit ETM, the control block can detect logic "1" in the first half of higher order bits (MSB) of the two inputs, A and B (i.e. A7 ~ A4 and B7 ~ B4).

Figure 3.7 shows the architecture and implementation of the control block. When both of the incoming signals A_i and B_i are "1", the "CTRL" signal will be high. This causes the MUX block to select and operate in the ETM multiplication mode. When all the input bits of either A or B is "0", the "CTRL" signal will be low and the conventional multiplication method will be chosen instead. This is assumed that the size of the operands is not "large enough" to employ the proposed ETM. It should be mentioned that when handing small input numbers, it is better to deploy conventional multipliers for better accuracy and speed.

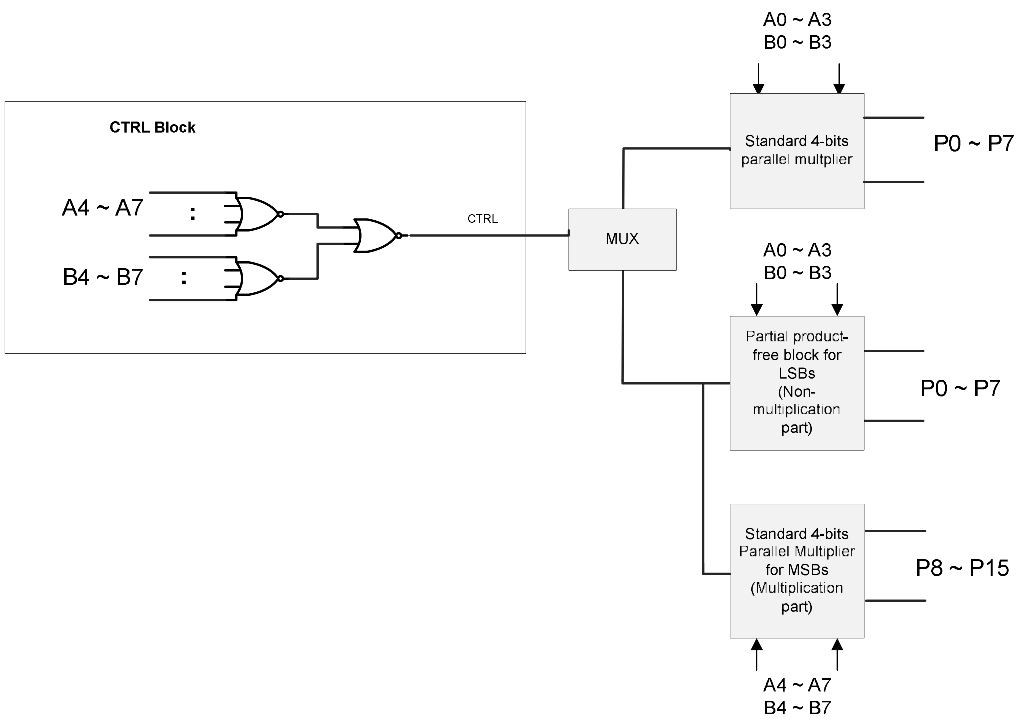

CTRL Block
A4 ~ A7
B4 ~ B7
CTRL
MUX
A0 ~ A3
B0 ~ B3
Standard 4-bits parallel multplier
P0 ~ P7
A0 ~ A3
B0 ~ B3
Partial product-free block for LSBs (Non-multiplication part)
P0 ~ P7
Standard 4-bits Parallel Multiplier for MSBs (Multiplication part)
P8 ~ P15
A4 ~ A7
B4 ~ B7

Figure 3.7 Architecture and implementation of the 8-bit ETM multiplier.

3.3.3 Design of the Conventional Multiplication Part

Any type of conventional multiplier mentioned in chapter 2 can be used and constructed for this part. In this project, the standard parallel multiplier is used due to its high performance. As mentioned in section 3.3.2, when the “CTRL” signal is low, the conventional multiplication would be employed for better accuracy. In standard N x N parallel multiplier, the 2N bit products are generated simultaneously [37]. However, in the proposed ETM, only N-bit products need to be produced by two N-bit inputs so that the hardware cost can be greatly reduced.

After the input patterns are divided into two at the initial stage, the presence of the higher orders is checked by the control block. Since there is no logic “1” located in half higher order of the operands, only 4-bit multiplier is to be used and generate the final product (P0 ~ P7), unlike the convention method, 8-bit multiplier would be necessary in such case. This provides not only the advantage of delay reduction in the carry signals traveling through the adder arrays but the power saving as compared to the traditional 8-bit multiplier.

Figure 3.8 illustrates the architecture of the conventional multiplication. All the bit products are generated by performing an AND operation on the bits of the two operands. The partial sums propagate diagonally in the south east direction along the lines of equal binary weight. The carry signals propagate downwards along increasing binary weight directions. The product can be represented by the sum of two segments, P_h and P_l representing the most and the least significant segments, respectively, i.e.,

$$P = P_h + P_l \tag{3.1}$$

$$= A_hB_h + A_hB_l + A_lB_h + A_lB_l \tag{3.2}$$

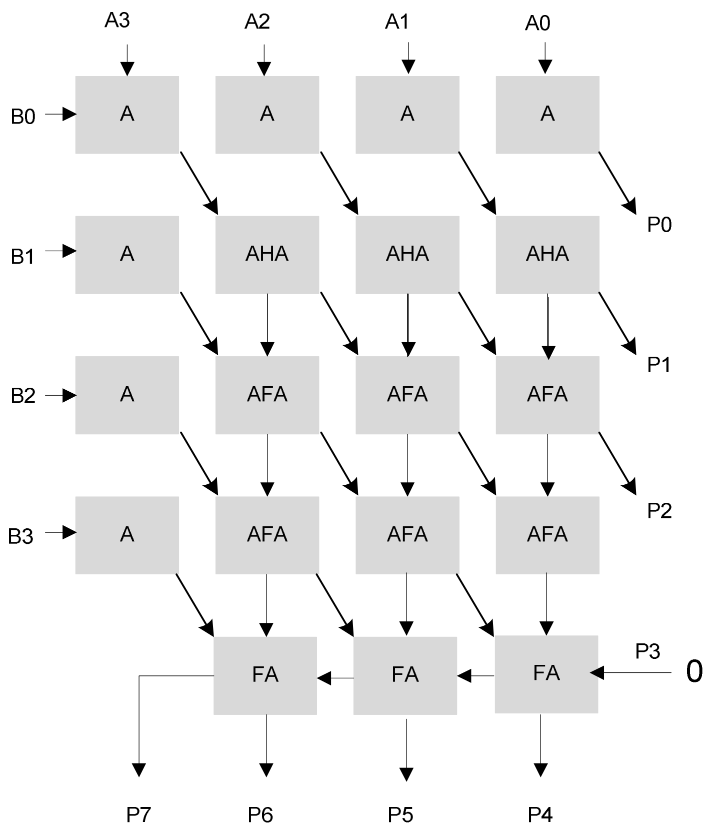
where P_h and P_l are N bits long each and N is assumed to be even. A_h and A_l are the most and least significant parts of A and B_h and B_l are those of B. The most and least significant segments of P are given by

$$P_h = \sum_{i=N}^{2N-1} p_i 2^i \tag{3.3}$$

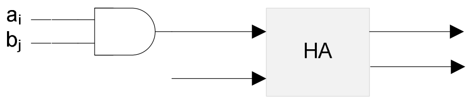
and

$$P_l = \sum_{i=0}^{N-1} p_i 2^i \tag{3.4}$$

respectively.

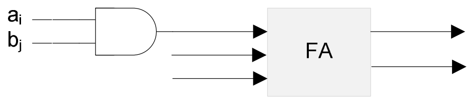


(a) Multiplier block diagram



(b)

Details of AHA cell



(c)

Details of AFA cell

Figure 3.8 A 4 x 4 multiplication using parallel multiplier where A, HA and FA are the AND, half-adder and full-adder cells, respectively.

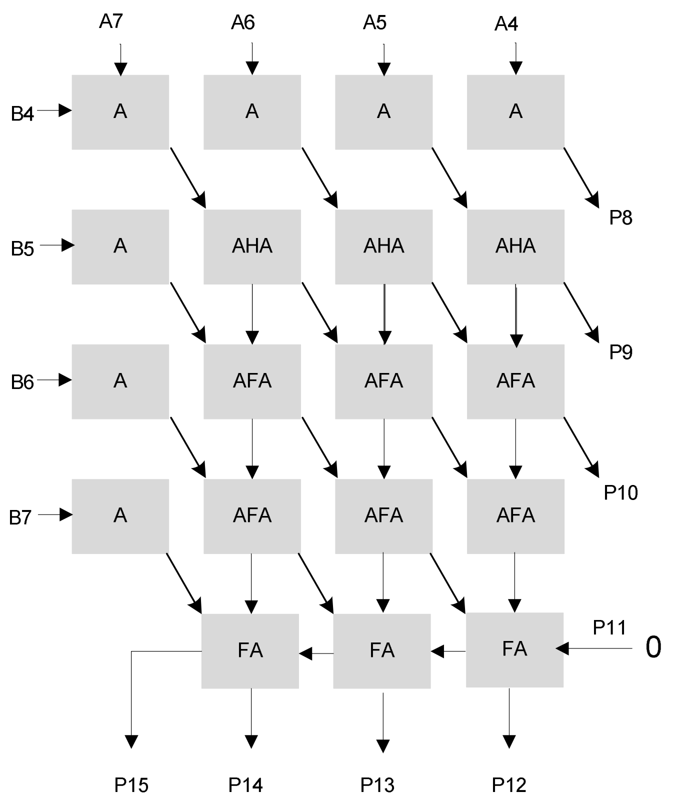
3.3.4 Design of the Non-conventional multiplication process

The non-conventional multiplication block is the most critical section in the proposed ETM as it determines the characteristic of accuracy, speed performance and power consumption of the multiplier. The proposed ETM design has a control block, which can detect logic "1" in the first half of most significant bits (MSB) of the two inputs, A and B (i.e. A7 ~ A4 and B7 ~ B4). When there is a "1", the "CTRL" signal will be activated and the input operands are high enough to operate in ETM mode. It consists of two major blocks: (i) multiplication part to generate higher order of the product and (ii) non-multiplication part to give the lower order bits of the final output.

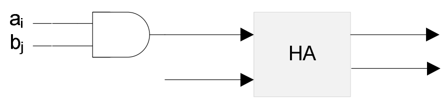
3.3.4.1 Multiplication part

Parallel multiplier is employed in the higher order bits due to the obvious reasons of having higher weight on their corresponding bit positions and also for their existence as 2's complements in signed operands. Because of the proposed design strategy, the overall delay time and power dissipation are determined mainly by the multiplication part.

In 8-bit multipliers, the multiplication operation is performed for the higher order bits, A4 ~ A7 and B4 ~ B7. The multiplication take 4 cycles to compute the final product (P8 ~ P15). In Figure 3.9, the higher order output carry bits are passed diagonally downwards and in the final stage, the carries and sums are merged in a fast carry-propagate adder stage. This structure is selected due to its merit of having a shorter worst case critical path and that it is uniquely defined. The schematic diagrams of the full-adder (FA) and half-adder (HA) had already been provided in section 3.3.3.

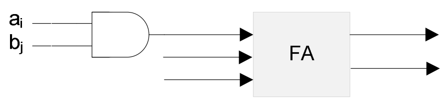


(a) Multiplier block diagram



(b)

Details of AHA cell



(c)

Details of AFA cell

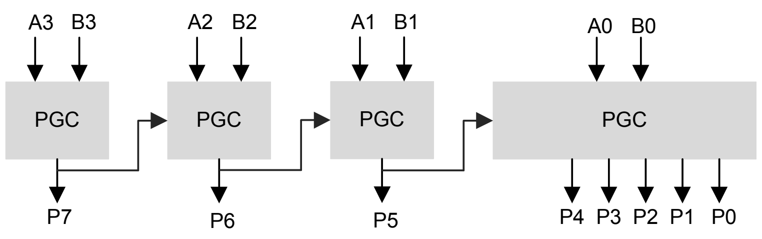
Figure 3.9 A 4 x 4 bit multiplier for generating Most Significant Bits.

3.3.4.2 Non-multiplication part

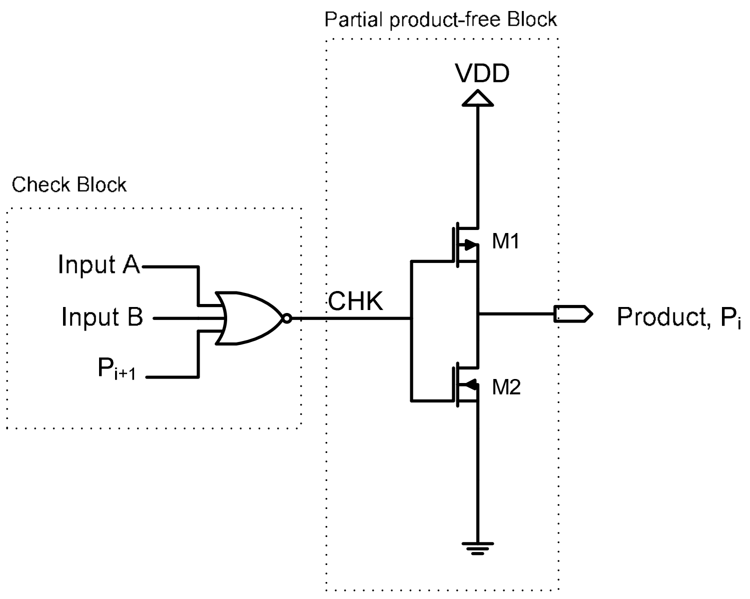
As described in section 3.2.3, the non-multiplication part consists of two: one is the check block and the other is the partial-product-free block. The block diagram of the partial-product-free multiplication block and the schematic implementation are shown in Figure 3.10. The partial-product-free block is a compensation circuit which still requires the most significant bits of the input A and B for further reduction of the truncation errors.

It is made up of four Product Generating Cells (PGC); each of them is used to generate the lower output digits. The check block generates a “CHK” signal for the corresponding bit position in the non-multiplication part. The function of the check block is to detect the first bit position where one of the input operand is “1”, to set the output signal on this position as well as those on its right to high. It can also be seen that when both of the input patterns A and B are “0”, the product bits will be set to “0”. From this observation, the PGC can be constructed as shown in Figure 3.9 (a). As can be seen in this figure, all the product generating cells (PGC) are cascaded by connecting the output of one cell to the input of the cell on its right. For the i-th PGC, when either of the input bits A_i or B_i is “1” or previous product bit P_{i+1} is “1”, the output signal P_i will be set to high. In such way, this high signal will be propagated to all the bit positions on its right.

In the PGC circuit, there are one NOR gate and two transistors M1 and M2. “CHK” is the signal coming from the check block and it is used to determine the result of the lower bits of the product. Operands A and B are checked if they both are equal to “1”. When either of these inputs is at logic “1”, M1 will turn on, allowing the output node to be directly connected to VDD, setting the product bit to “1”. This mode is also named as pull-up mode. When both of the operands A and B are at “0”, M2 will turn on, while M1 is turned off, setting the output node to “0”. The NOR gate is chosen to be used in the ETM to reduce the transistor count of the overall circuit.



(a)



(b)

Figure 3.10 Non-multiplication part: (a) Overall architecture; (b) Schematic diagram of a PGC.

Chapter 4 Circuit Simulation

The performance of multiplier is dependent on the internal and external parameters, for example, the technology used, supply voltage, output load and etc. Once these parameters are changed, the performance of the multiplier varies. In order to have a fair comparison among various multipliers, those parameters have to be fixed for all designs. The transistor level simulation of the proposed ETM circuit has been simulated using Cadence SPECTRE tool.

In this work, all the circuits are designed and simulated based on the 0.18-µm Chartered Semiconductor Manufacturing Limited CMOS process technology. Simulations are performed at the temperature, 25°C and a supply voltage of 1.8 V. Each circuit drives an output load of 10 fF. A clock frequency of 20 MHz is used for all the reported and new multipliers.

4.1 Performance Comparisons and Discussions

In this section, the propagation delay and the power consumption during difference switching activities are measured for the proposed ETM. Besides, an 8 x 8 standard parallel multiplier is also constructed, as shown in Figure 4.1, for the purpose of performance comparison. Simulated tool, circuit technology, input patterns and parameters are all the same as those introduced in ETM.

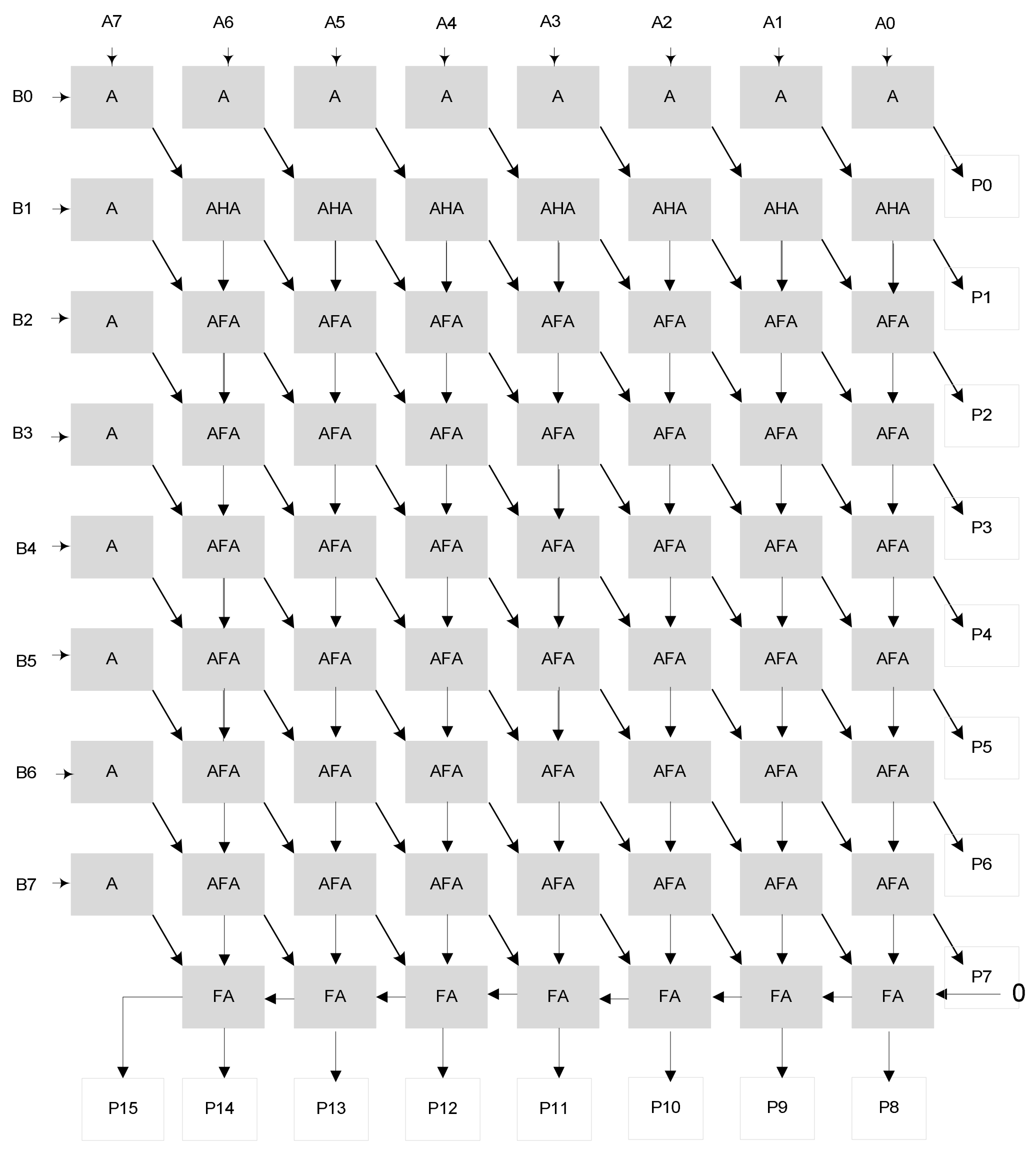


Figure 4.1 8-bit standard multiplier to be compared with proposed ETM.

To demonstrate the functionality of the proposed ETM, five different input patterns are randomly selected and simulated. These five data, with corresponding results and accuracies are listed in Table 4.1 (note that all the numbers are written in HEX format for the purpose of convenience).

Table 4.1 - Five input patterns and their corresponding results and accuracies

A	B	Correct Result	Obtained Result by ETM	Accuracy
AA	55	3872	32FF	90%
CC	33	28A4	24FF	91%
F1	FC	ED3C	E1FF	95%
F0	F0	E100	E100	100%
FC	FC	F810	E1FF	91%
09	03	01B	01B	100%

A small input pattern (when A = 08, B = 03) is included as one of the test vectors in order to illustrate the functionality of ETM. As shown in Chapter 3, ETM is able to detect the presence of "1" in higher order of input operands and when there is none, standard multiplication would be engaged. This is to provide the perfect accuracy of the final product for the small numbers.

Figure 4.2 (a) (b) (c) shows the example of new ETM output waveform (P0 ~ P15), the product of F1 (hex) and FC (hex). The expected result which contains a 5% error, E1FF is obtained via the new ETM, as opposed to the correct result of ED3C if conventional method is engaged.

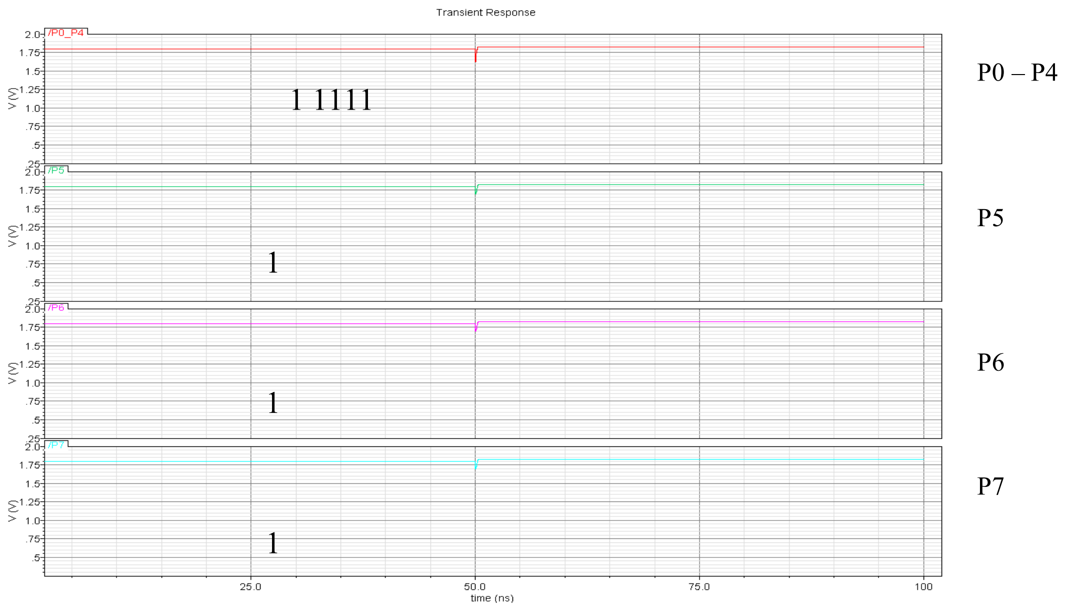


Figure 4.2 (a) Output waveform of ETM illustrating the partial product (P0 ~ P7) of F1 x FC (Transient response of Magnitude versus Time).

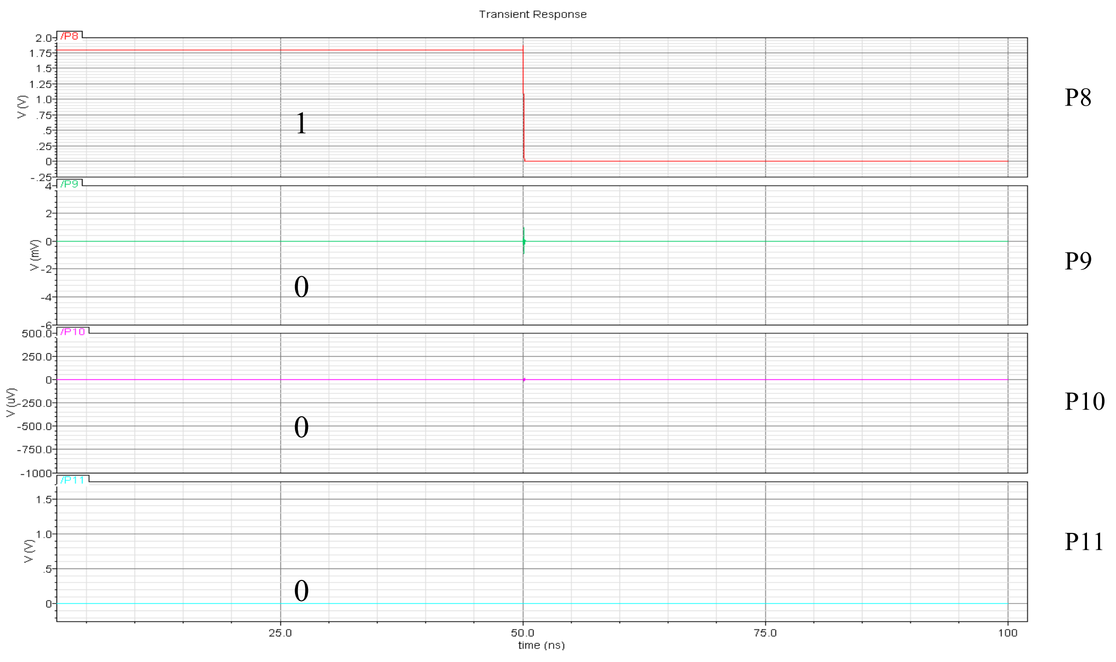


Figure 4.2 (b) Output waveform of ETM illustrating the partial product (P8 ~ P11) of F1 x FC (Transient response of Magnitude versus Time).

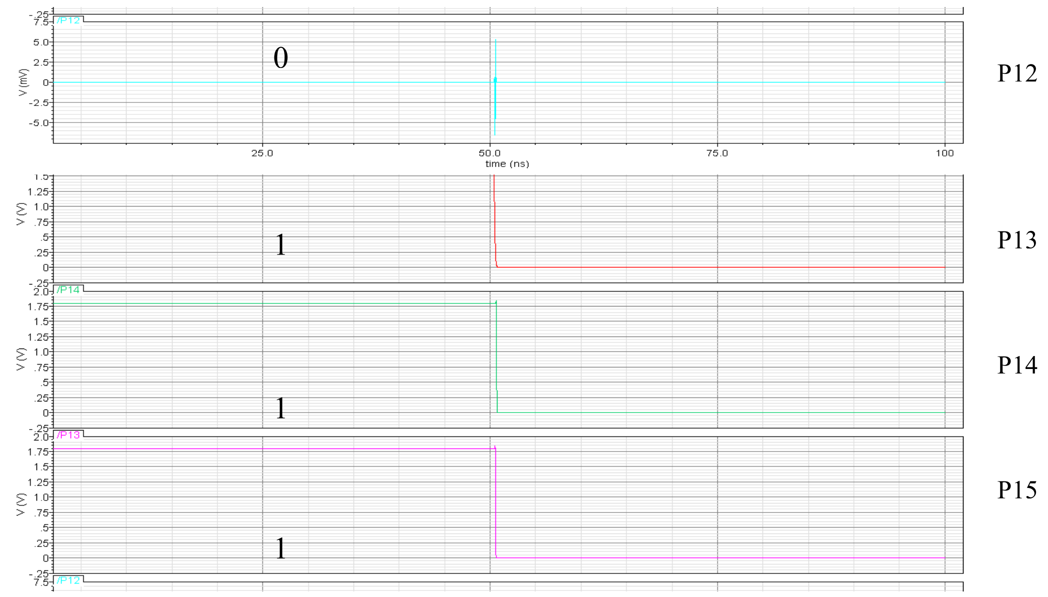


Figure 4.2 (c) Output waveform of ETM illustrating the partial product (P12 ~ P15) of F1 x FC (Transient response of Magnitude versus Time).

According to analysis, the bit sequence at the output nodes (P0 ~ P15) should be "1110 0001 1111 1111". These analytical results are exactly same as the simulation waveforms shown in Figure 4.2.

The propagation delay and power parameters for the proposed ETM are tabulated in Table 4.2. To demonstrate its advantage, the conventional 8x 8 parallel multiplier (Figure 4.1) was also simulated and the results obtained are provided in the same table.

The Power-Delay-Product (PDP) is used as a Figure-of-Merit (FOM) measurement. It is the average product of the power consumption at 20MHz and propagation delay measured from the input to output (D-Q Delay). The power dissipation of all data circuits is taken into account when calculating the total power consumption.

Table 4.2 - Simulation results of the proposed ETM and conventional multiplier

	A	B	Power (uW)	Delay (nsec)	PDP (fJ)
Proposed 8-bits ETM	AA	55	49.65	0.97	48.01
	CC	33	39.32	0.79	31.06
	E3	1C	28.90	0.76	21.96
	F0	F0	21.44	0.67	14.36
	FC	FC	15.46	0.63	9.74
	09	03	21.41	0.73	16.51
Conventional 8-bits Multiplier	AA	55	247.70	0.98	242.99
	CC	33	215.90	0.82	177.04
	E3	1C	166.10	0.85	141.19
	F0	F0	107.69	0.79	85.29
	FC	FC	96.10	0.71	68.23
	09	03	17.69	0.69	12.20

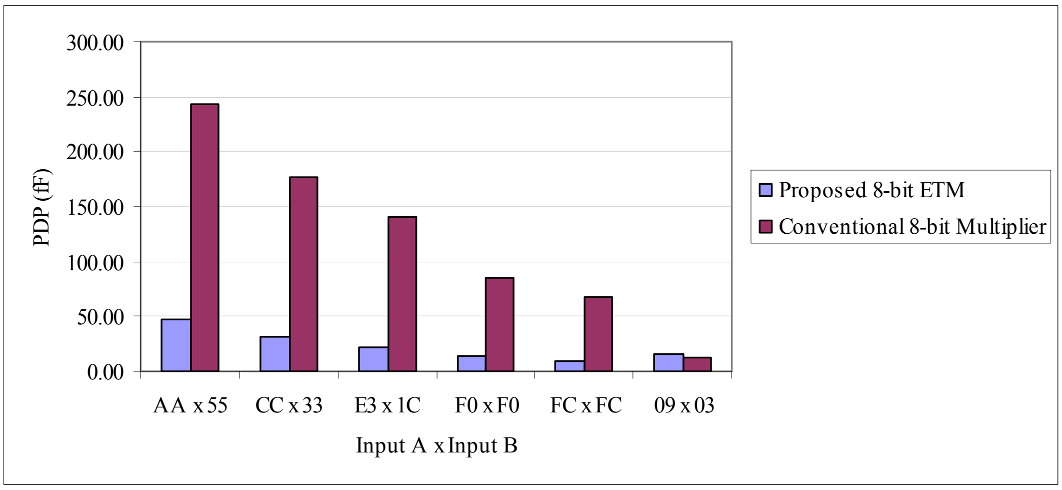


Figure 4.3 Performance comparisons of ETM and standard multiplier.

As seen in Table 4.2 and Figure 4.3, the PDP of the proposed ETM is significantly improved 5 to 7 times with different input data at the trade-off in accuracy of their corresponding outputs. This is because ETM has fewer numbers of transistors for large input patterns, which is an area-saving technique to reduce its high area cost in the summation of partial product. As a result, it consumes lower power than its full-width counterpart- the standard parallel multiplier. Besides, it eliminates the redundant internal switching activities by omitting about half of the adder cells which are needed to be used for partial products. ETM has impressive improvement in PDP especially for the cases where higher switching activities at the input data (for example, A = AA, B = 55). However, for small input data (when A = 09, B = 03), there is a compromise in PDP due the presence of overhead circuitry in control and multiplexing circuit when compared with conventional method. PDP of the ETM is slightly higher than the standard multiplier in this case.

Other than the above advantages, ETM offers the higher speed among all the compared data inputs. This explains why ETM uses truncated multiplication architecture, which generally has better driving ability to help achieve small delay. Due to the number of transistors involved, it is able to achieve the power savings in the transistor routing network. To conclude, the proposed ETM is able to yield the best performance in term of both speed and power consumption especially for large input patterns.

4.2 Power consumption analysis at different input data activities

As mentioned in section 2.4, power dissipation is proportional to the switching activities, α of the input data sequence. The higher the switching activity, the more power is being consumed. However, there is an exception, in situations where circuits consumes more power at $\alpha = 0$ than at $\alpha = 1$. Thus, power measurements should be conducted for a range of data patterns, comprising the worst and the best cases.

In this multiplier, the average power dissipation is calculated for four different data activities. Upon optimizing the reported multipliers, simulations are carried out using different input data activities: '1010' represents maximum switching activity ($\alpha = 1$), '0011' stands for 0.5 switching activity ($\alpha = 0.5$), and both '0000' and '1111' denotes zero switching activity ($\alpha = 0$).

The power consumption break-ups of multipliers for different input activities are illustrated in Table 4.3 and Figure. 4.4. It shows that the proposed ETM consumes the least power consumption at different switching activities. The power saving varies from 5 to 6 times better in ETM when compared with standard multiplier.

Table 4.3 - Average power consumption at different switching activities (μW)

	α = 0	α = 0.25	α = 0.5	α = 0.75	α = 1.0
Proposed 8-bit ETM	0.0148	7.27	25.46	32.99	49.65
Conventional 8-bit Multiplier	0.0696	34.48	146.1	214.9	247.7

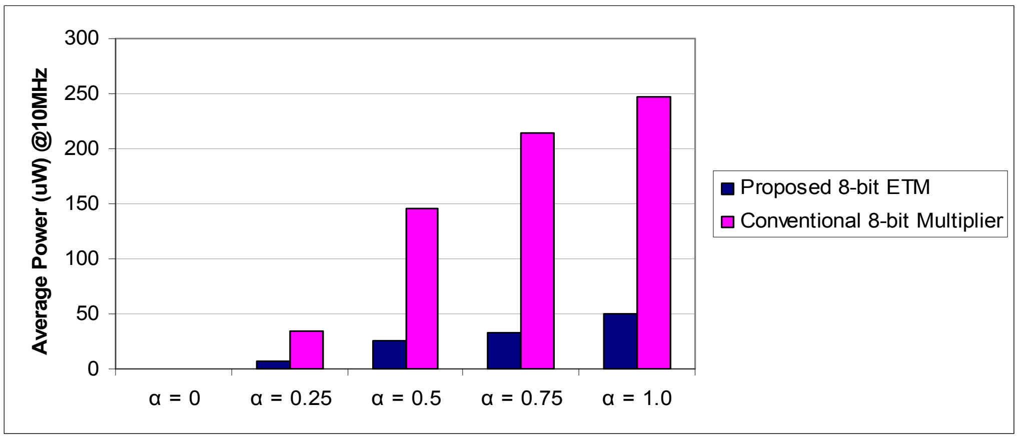


Figure 4.4 Power consumption versus switching activity.

4.3 Area comparison of ETM and standard multiplier

Since the silicon area occupied by a multiplier has a direct impact on the cost of the chip and the packing density, the area of transistors and total number of transistors is compared.

In this project, area measurement is obtained by performing a summation of the product of the individual transistors' gate lengths and widths used by both PMOS and NMOS. The proposed ETM achieves an impressive savings of more than 50% for an 8 x 8 fixed-width multiplication when operating 20 MHz. The area savings are more prominent for higher operand length and more stringent timing constraint.

The circuit size is not only determined by the number and size of transistors but also other factors such as wiring and interconnect parameters such as capacitance, resistance and inductance should be considered. These factors are becoming important when extreme performance is a goal [1]. However, from the design methodology point of view, the proposed ETM has simpler design and fewer transistor counts, which led to have better advantage in shrinking dimensions when compared with its standard counterpart.

Table 4.4 - Transistor area comparison for ETM and standard multiplier

	Area $(\mu m)^2$
Proposed 8-bit ETM	204
Conventional 8-bit Multiplier	438

4.4 Summary of 8-bit multiplier

The performance comparison between ETM and conventional multiplier at and maximum input switching activity (i.e. A = hex 55, B = hex AA) is summarized in Table 4.5. It shows that ETM is able to achieve 5 times better in PDP, with area saving of 50% at the expense of 90% accuracy when compared to its standard counterpart with the same test vector being used.

Table 4.5 - Quality comparison between ETM and standard multiplier

	PDP (fJ)	Area $(\mu m)^2$	Accuracy (%)
8-bit ETM	48	204	90%
Conventional 8-bit Multiplier	243	438	100%

4.5 Design and implementation of a 12-bit ETM

The architecture block diagram of the 12-bit ETM is depicted in Figure 4.5. In the proposed design, the entire input patterns for inputs A and B are divided into 2: 6-bit blocks each. The control block is used for checking a logic "1" in the most significant bit position of the inputs, (A11 ~ A6 and B11 ~ B6). When it detects a "1", the 24-bit product is generated by partial-product free block for first 12 least significant bits, P0 ~ P11 using a compensation circuit. The standard 6-bit parallel multiplier is employed to produce the next higher order 12-bit of the product, P12 ~ P23. If all the most significant bits are detected as "0", the 6-bit conventional multiplier is selected to directly generate the final result, P0 ~ P11.

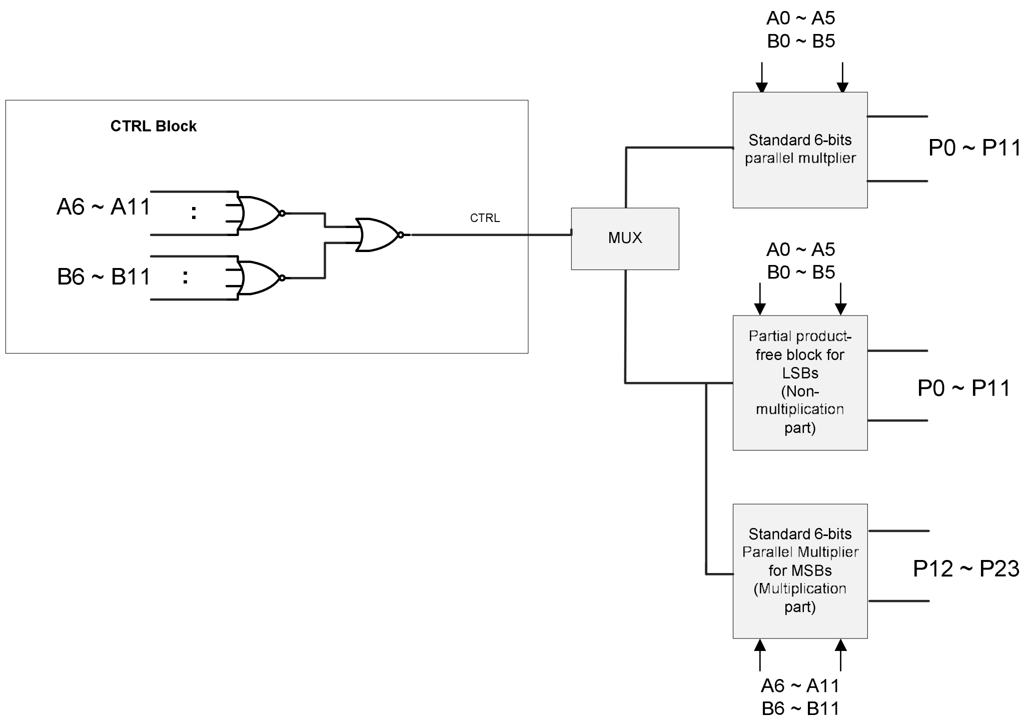

CTRL Block
A6 ~ A11
B6 ~ B11
CTRL
MUX
A0 ~ A5
B0 ~ B5
Standard 6-bits parallel multplier
P0 ~ P11
A0 ~ A5
B0 ~ B5
Partial product-free block for LSBs (Non-multiplication part)
P0 ~ P11
Standard 6-bits Parallel Multiplier for MSBs (Multiplication part)
P12 ~ P23
A6 ~ A11
B6 ~ B11

Figure 4.5 Architecture and implementation of the proposed 12-bit ETM multiplier.

Both the multiplication and non-multiplication parts adopt the circuits proposed in Figures 3.9 and 3.10. Note that additional blocks for partial product generators and product generating cells (PGC) are introduced in the proposed 12-bit ETM structure.

As concluded in section 3.2, when the inputs are of large numbers, the accuracy performance of the proposed ETM is better. To test whether the situation is true for the 12-bit ETM, the same simulations are performed and results obtained are provided in Table 4.6. It is obvious that the accuracy of the 12-bit multiplier falls in better acceptable range as compared to the 8-bit proposed multiplier (see Table 4.1). This is because the unacceptable results are distributed over the whole product range from 0 to 23 bits instead of "squeezing" in the range of small numbers.

Similar to 8-bit multiplier, a test vector (when A = A, B = 6) is selected as a small number to illustrate its 100% accuracy. This is because ETM would engage the standard multiplication when input data is very small.

Table 4.6 - Five input patterns and their corresponding results and accuracies

A	B	Correct Result	Obtained Result by ETM	Accuracy
AAA	555	38 DC72	37 2FFF	97%
CCC	333	28 F0A4	26 4FFF	94%
E38	1C7	19 4588	18 8FFF	97%
F0F	F0F	E2 C2E1	E1 03FF	98%
FC0	FF0	FB 0400	F8 1FFF	99%
0A	06	00 003C	00 003C	100%

4.5.1 Performance Comparisons and Discussions

The circuit simulation is performed using SPECTRE and the results are tabulated in Table 4.7. A conventional 12-bit multiplier is also constructed and simulated for the comparison. All the simulation parameters are the same with those described in section 4.1.

Table 4.7 - Simulation results of the 12-bit ETM and standard parallel multiplier

	A	B	Power (uW)	Delay (nsec)	PDP (fJ)
Proposed 12-bits ETM	AAA	555	122.4	0.96	117.50
	CCC	333	99.50	0.89	88.56
	E38	1C7	72.13	0.90	65.21
	F0F	0F0	59.54	0.83	49.42
	FC0	03F	18.29	0.73	13.26
	0A	06	31.20	0.84	26.21
Conventional 12-bits Multiplier	AAA	555	686.5	1.05	720.83
	CCC	333	606.40	0.95	576.08
	E38	1C7	404.70	0.89	360.18
	F0F	0F0	255.00	0.85	216.75
	FC0	03F	96.40	0.75	72.30
	0A	06	20.35	0.76	15.47

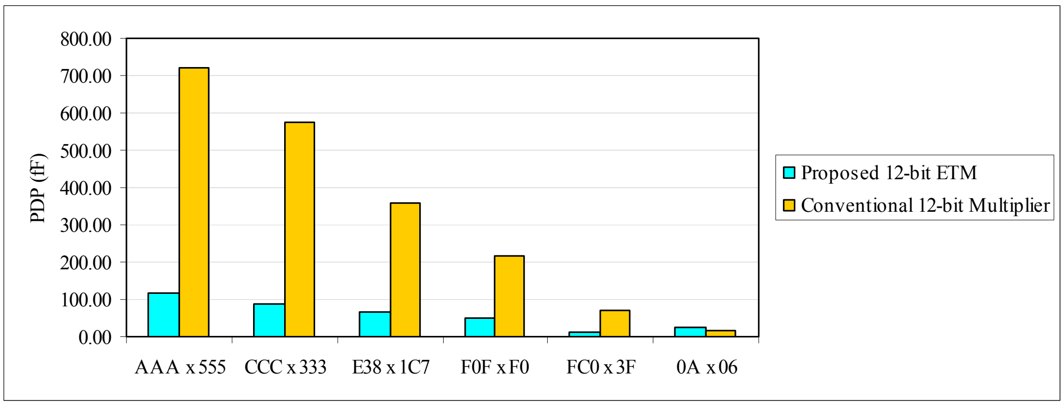


Figure 4.6 Performance comparisons of ETM and 12 x 12 parallel multiplier.

As depicted Figure. 4.6, the difference in PDP of the reported and new multiplier is compared in order to find out the high performance achieved by the ETM. Five test vectors with switching activities, α ranging from 0.2 to 1.0 were selected to calculate the average power at 20MHz. Similar to 8-bit multiplier it has better performance than most of the conventional multipliers in terms of PDP except the small test vector (A = 0A, B = 06) in this case. The performance is slightly compromised due to the overhead circuitry to check the presence of “1” in half of higher orders of the input operands.

The power consumption break-ups of the multipliers at different input switching activities are provided in Table 4.8 and Figure 4.7. When compared to standard multiplier, four to six times improvement for the proposed ETM. It offers the excellent power consumption when there is the maximum switching activities of the input operands which has very significant power dissipation reduction.

Table 4.8 - Average power consumption at different switching activities (μW)

	α = 0	α = 0.25	α = 0.5	α = 0.75	α = 1.0
Proposed 12-bit ETM	0.0374	21	65.13	90.05	122.4
Conventional 12-bit Multiplier	0.167	115	374.7	586.4	686.5

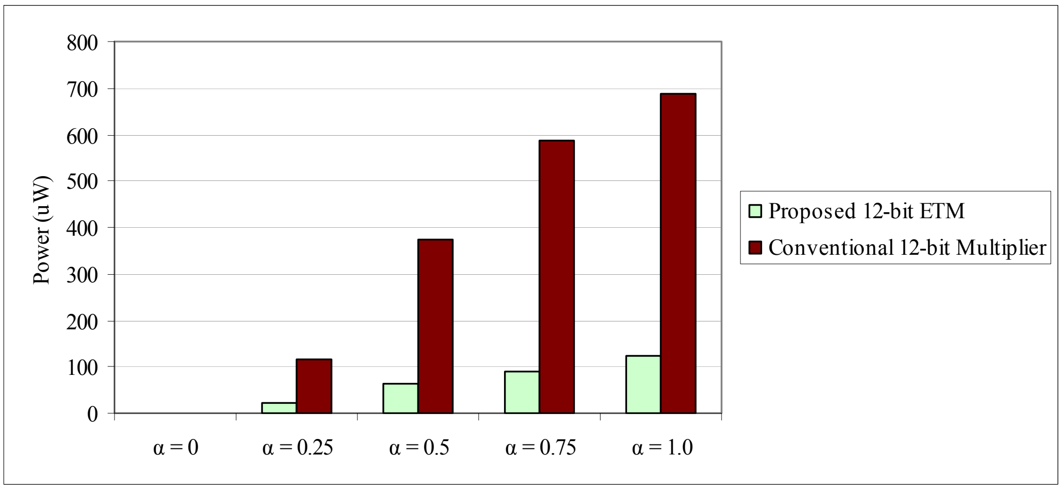


Figure 4.7 Power consumption versus switching activity.

The 12-bit ETM gives more effective area-saving technique, which reduces its high area cost in the summation of partial product. While maintaining the lower average error from the conventional method, about 50% of the precious silicon area can be saved from a 12 x 12-bit fixed-width multiplier.

Table 4.9 - Transistor area comparison for ETM and standard multiplier

	Area (μm^2)
Proposed 12-bit ETM	491
Conventional 12-bit Multiplier	1028

The performance comparison between ETM and conventional multiplier at and maximum input switching activity (i.e. A = hex 555, B = hex AAA) is summarized in Table 4.10. It shows that the ETM can achieve about 6 times better in PDP, with area saving of more than 50% as compared with its standard counterpart when the same test vector is used. As compared with 8-bit ETM in Table 4.5, it is clear that the accuracy of 12-bit ETM permits superior accuracy for the same input switching activity. Hence the chance of better accuracy is getting higher when the size of the multiplier increases. When both of the input operands are small numbers, the acceptance probability decreases and it would lead to occur unacceptable results very often. This is because small numbers will be calculated only in the inaccurate part of the multiplier. Hence the proposed ETM is especially suitable for large input patterns.

Table 4.10 - Quality comparison between ETM and standard multiplier

Performance Analysis	PDP (fJ)	Area $(\mu m)^2$	Accuracy (%)
12-bit ETM	117.5	491	97%
Conventional 12-bit Multiplier	720.8	1028	100%

Chapter 5 Conclusions and Recommendation

5.1 Conclusions

In this report, a new design approach that takes into account the accuracy in digital VLSI design is proposed. The concept is implemented on the schematic to realize the proposed Error-Tolerant Multiplier (ETM), that trades certain amount of accuracy for power saving and performance improvement. In the proposed design strategy, the multiplication part is implemented using conventional method to ensure the greater accuracy in the higher order bits while the non-multiplication part is designed in a novel method that permits certain amount of errors. Such errors will be a value that is acceptable to the circuit designer/customer, for achieving savings in power and also enhancement in speed of operation. These multipliers are widely used in application specific data paths in multimedia and wireless communication applications where some degree of saturation error within the dynamic range of interest is tolerable depending of the level of perceptual quality and signal-to-noise degradation it induced [43].

Extensive comparison with different sizes of conventional multipliers showed that our proposed ETM consumes much less power while still achieving good accuracy. These multipliers are simulated under pre-defined conditions using Cadence – Spectre tool. All simulations are done by using Chartered Semiconductor Manufacturing limited 0.18 µm CMOS process with 1.8 V supply voltage.

Pre-layout simulation is carried out on all multiplier circuits. Their gate area, power dissipation, propagation delay, power-delay-product and parameter analysis are then observed. Although the proposed ETM has smaller number of transistors, it has the small internal delay due to its configuration and structure. Simulation results showed that the ETM can achieve four to six times better reduction in power dissipation for maximum input switching activities, with area saving of 50% when compared to conventional designs at the expense of accuracy 97% for 12-bit and 90% for 8-bit multiplier respectively.

The performance comparison also shows that the new ETM gives the lowest power delay product. It has the best performance in terms of both speed and power consumption while keeping the total transistor count low. Such enhancement is possible through accommodating the multiplication and addition process, which speeds up and allow enhanced energy-efficient when the multiplier is divided into two separate parts. The proposed ETM consists of an accurate part and an inaccurate part; both performing their operations simultaneously.

Reducing the switching activity α, i.e. the charging/discharging of internal nodes, has led to reductions in the overall power consumed by the new ETM. It is also very small in area; this is an advantage as smaller size saves the manufacturing cost. The proposed design is therefore suitable for use in high speed, high performance and low voltage applications.

Table 5.1 - Summary of performance parameters of proposed ETMs for maximum switching activities

Performance Analysis	8-bit ETM	12-bit ETM
Propagation delay (D-Q) (ns)	1.21	2.64
Average Power at 20 MHz (μW)	49.65	122.4
PDP (fJ)	60	323
Number of Transistors	944	2272
Calculated Area $(\mu m)^2$	204	491
Accuracy (%)	90	97

The ETM provides an innovative way to break through the weakness of the conventional digital IC design technology that hinders circuits from simultaneously achieving both low-power and high-speed. However, this may result in certain accuracy loss. When the accuracy is brought as new parameter of the design process, it in fact, provides more flexibility to the circuit engineers. If low-power and high-speed are desired, the accuracy can be used as a trade-off and vice versa. The choice of the multiplier will depend on the application objectives but for optimal performance, it is advisable to use low power multipliers throughout the system, leaving the high accuracy multipliers for the critical paths only.

5.2 Recommendation

For future studies, the following recommendations and guidelines could be considered:

1. All analyses in this project are based on Chartered Semiconductor Manufacturing 0.18-µm CMOS technology. This could be the limitation, as some performances of circuits are technology dependent, especially in the latest 90 nm fabrication technology, or 65 nm technologies. A systematic study should be taking to study and determine those performances that are technology dependent.

2. Comparison with other different types of multiplier configurations, such as the Wallace Tree Multiplier, Booth Multiplier, and Serial-parallel Multiplier, etc. should be studied in greater detail. This is required to better appreciate all the special characteristics of different types of multiplier structures so that a more robust and better performance multiplier can be designed.

3. The functional and performance verification in the sub 1.2 V of regime can be simulated and performed for all multipliers reported in this project in order to investigate the behavior under simultaneous voltage and technology scaling.

4. The proposed multiplier in this report has a fixed dividing strategy which means the accuracy of the multiplier is fixed for every input pattern. In real applications, one may often need to choose a suitable accuracy to cater to the requirements desired. Hence the circuit design to adjust the accuracy of ETM should be explored and implemented.

5. The application of ETM in digital signal processing (DSP) shall be studied and ETM-based Fast Fourier Transform (FFT) function can be implemented to demonstrate its functionality.

6. This project focuses on the application level simulation and transistor level pre-layout simulation. As a complete design process, other works, such as drawing layout, post-layout simulation, fabrication and testing should be done.

7. The error-tolerant design can be extended from the Application Specific Integrated Circuit (ASIC) implementation to the Field Programmable Gate Array (FPGA) implementation, which is more popular in current digital system. It is hence worthwhile to employ the concept of Error-Tolerance in FPGA technology.

6. References

[1] Rabery, J., “Digital Integrated Circuits: a design perspective”, Eaglewood Cliffs, NJ, Prentice-Hall, 1996.

[2] Zhao, P., Darwish, T., and Bayoumi, M., “Low Power and High Speed Explicit-Pulsed Flip-Flops”, Midwest Symposium on Circuit and Systems, vol.2, pp.477-480, August, 2002.

[3] Gonzalez, R., Gordon, B.M., and Horowitz, M.A., “Supply and threshold voltage scaling for low power CMOS”, IEEE Journal of Solid-State Circuits, 32: 1210-1216, August, 1997.

[4] Korkmaz, P., Akgul, B.E.S., and Palem, K., “Characterizing the Behavior of a Probabilistic CMOS Switch through Analytical Models and its Verification through Simulations, CREST Technical Report no. TR-05-08-01, August, 2005.

[5] Zhu, N., Goh, W.L., Zhang, W.J., Yeo, K.S., and Kong, Z.H., “Design of Low-Power High Speed Trancation-Error-Tolerant Adder and Its Application in Digital Signal Processing”, TVLSI-00206-2008.

[6] Breuer, M.A., and Zhu, H., “Error-tolerance and multi-media,” In Proc of the 2006 International Information Hiding and Multimedia Signal Processing, 2006.

[7] Chong, I.S., and Oretega, A., “Hardware testing for error tolerant multimedia compression based on linear transforms,” Defect and Fault Tolerance in VLSI Systems Symp., pp. 523 – 531, 2005.

[8] International Technology Roadmap for Semiconductors, latest edition available online at http://public .itrs.net/.

[9] Bellaouar, A., and Elmasry, M.I., “Low-Power Digital VLSI Deign: Circuits and Systems,” Kluwer Academic Publishers, 1995.

[10] Clements, A., “The Principles of Computer Hardware,” 3rd Edition, Oxford University Press, 2000.

[11] Braun, E.L., “Digital Computer Design, Logic Circuitry, Synthesis,” Academic Press, New York, 1963.

[12] Hwang, K., "Computer Arithmetic: Principles, Architecture and Design," John Wiley and Sons, 1979.

[13] Baugh, C.R., and Wooley, B.A., "A Two's Complement Parallel Array Multiplication Algorithm," IEEE Transactions on Computers, vol. 22, no. 12, December, pp. 1045-1047, 1973.

[14] Elrabaa, M.S., Abu-Khaterand, I.S., and Elmarsy, M.I., "Advanced Low-Power Digital Circuit Techniques," Kluwer Academic Publishers, 1997.

[15] Booth, A.D., "A Signed Binary Multiplication Technique," Quarterly J. Mechanics and Applied Mathematics, vol. 4, Pt 2, June, 1951.

[16] Law, C.F., Rofail, S.S., and Yeo, K.S., "Low-Power Circuit Implementation for Partial Product Addition Using Pass-Transistor Logic," IEE Proc.Circuits Devices Systems, vol. 146, no. 3, pp. 124-129, June, 1999.

[17] Nelson, V.P., Nagle, H.T., Carrol, B.D., Irwin, J.D., "Digital Logic Circuit Analysis and Design," New Jersey, Prentice Hall, 1995.

[18] Bellaouar, A., and Elmasry, M.I., "Low-Power Digital VLSI Design: Circuit and Systems," Kluwer Academic Publishers, 2005.

[19] Yeo, K.S., Rofail, S.S., and Goh, W.L., "CMOS/BICMOS ULSI, Low-Voltage, Low-Power," Upper Saddle River, NJ, Prentice-Hall, 2002.

[20] Mukherjee, S.S., Weaver, C.T., Emer, J., Reinhardt, S.K., and Austin, T., "A Systematic Methodology to Compute the Architectural Vulnerability Factor for a High-Performance Microprocessor," In Proc. of the 36th annual IEEE/ACMInt'l Symp. On Micro architecture, December, 2003.

[21] Austin, T.M., DIVA, "A Reliable Substrate for Deep Submicron Microarchitecture Design," In Proc. of the 32nd annual IEEE/ACM Int'l Symp. on Microarchitecture, November, 1999.

[22] Gomaa, M., and Vijaykumar, T. N., "Opportunistic Transient- Fault Detection," In Proc. of the 32nd Annual Int'l Symp. On Computer Architecture, June, 2005.

[23] Li, X.H., and Yeung, D., "Application-Level Correctness and its Impact on Fault Tolerance," 1-4244-0805-9/07/IEEE, 2007.

[24] Johnson, B.W., "Fault tolerance: the electrical engineering handbook," CRC Press, 1993.

[25] Vladimir, V., Jelena, K., and Ivan, M., "Partial error tolerance for bit-plane FIR filter architecture," International Journal of Electronics and Communications, pp. 398-405, May, 2008.

[26] Tso, B. J., and Shen, F. H., "Low-Error Carry Free-Width Multipliers with Low-Cost Compensation Circuits," IEEE Transactions on Circuits and Systems, vol 52, No. 6, June, 2005.

[27] Recommendation ITU-T P.830, Subjective performance assessment of telephone band and wideband digital codecs, Int'l Telecommunications Union, 1996.

[28] Madrid, P.E., Miller B., and Swartzlander, E.E., "Modified Booth Algorithm for High Radix Fixed-Point Multiplication," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 1, no. 2, pp. 522–524, June, 1993.

[29] He, Y., and Chang, C.H., "A New Redundant Binary Booth Encoding for Fast 2^n Bit Multiplier Design," IEEE Transactions on Circuits and Systems I, vol. 56, no. 6, pp 1192 - 1201, June, 2009.

[30] Lim., Y.C., "Single precision multiplier with reduced circuit complexity for singal processing applications," IEEE Trans. Comp., vol. 41, no. 10, pp. 1333-1336, October, 1992.

[31] Schutle, M.J., and Swartzlander, Jr.E.E., "Truncated multiplication with correction constant," VLSI Signal Processing, vol. 6, pp. 388-396, 1993.

[32] Kidambi, S.S., and El-Guibaly, F., Senior Member and Antoniou, A., Fellow IEEE, "Area-Efficient Multipliers for Digital Signal Processing Applications," IEEE Trans. On Circuits and Systems-II, vol. 43, no. 2, February, pp. 90-95, 1996.

[33] Jou, J.M., Kuang, S.R., and Chen, R.D., "Design of low-error-fixed-width multipliers for DSP applications, " IEEE Trans, Circuits System II, Analog and Digital Signal Processing, vol. 46, no. 6, pp. 836-842, June, 1999.

[34] Van, L.D., Wang, S.S., and Feng, W.S., "Design of the lower error fixed-width multiplier and its application," IEEE Trans., Circuit Systems II, Analog Digital Signal Processing, vol 47, no. 10, pp. 1112-1118, October, 2000.

[35] Jou, S.J., and Wang, H.H., "Fixed-width multiplier for DSP application," in Proc. International Conference, Computer Design (ICCD), pp. 318 – 322, September, 2000.

[36] Cho, K.J., Lee, K.C., Chung J.G., and Parhi, K.K., "Design of low-error fixed-width modified booth multiplier," IEEE Trans., VLSI System, vol. 12, no. 5, pp. 522-531, May 2004.

[37] Elguibaly, F., "A Fast Parallel Multiplier-Accumulator Using the Modified Booth Algorithm," IEEE Trans. Circuits and Systems, vol. 47, no. 9, pp. 902-908, 2000.

[38] Farooqui, A., and Oklobdzija, V., "General Data-Path Organization of a MAC Unit for VLSI Implementation of DSP Processors," Proc. 1998 IEEE Int'l Symp. Circuits and Systems, vol. 2, pp. 260-263, 1998.

[39] Itoh, N., Naemura, Y., Makino, H., Nakase, Y., Yoshihara, T., and Horiba, Y., "A 600-MHz 54x54-bit Multiplier with Rectangular-Styled Wallace Tree," IEEE J. Solid-State Circuits, vol. 36, no. 2, pp. 249- 257, 2001.

[40] Oklobdzija, V.G., Villeger, D., and Liu, S.S., "A Method for Speed Optimized Partial Product Reduction and Generation of Fast Parallel Multipliers Using an Algorithmic Approach," IEEE Trans. Computers, vol. 45, no. 3, pp. 294-306, March, 1996.

[41] Stelling, P.F., Martel, C.U., Oklobdzija, V.G., and Ravi, R., "Optimal Circuits for Parallel Multipliers," IEEE Trans. Computers, vol. 47, no. 3, pp. 273-285, March, 1998.

[42] Yeh, W.C., and Jen, C.W.,"High-Speed Booth Encoded Parallel Multiplier Design," IEEE Trans. Computers, vol. 49, no. 7, pp. 692-701, July 2000.

[43] Liao, Y.C., Chang, H.C., and Liu, C.W., "Carry estimation for two's complement fixed-width multipliers," In Proc. 2006 IEEE Workshop on Signal Processing Syst. Design and Implementation, pp. 345-350, October, 2006.

Appendix A

MATLAB code for testing the accuracy of the 20-bits ETM

```
%2-input multiplier with error

%Dimensioning of variables
N1 = 20;
N2 = (N1/2);
INA = zeros(2^N2-1,N2);                    %Input A vector
INB = zeros(2^N2-1,N2);                    %Input B vector
MUL_OUT_ACT = 0;                             %Actual multiplier result
Temp1 = zeros(1,N1);
% Temp2 = zeros(1,N2);
Temp3 = double(0);
Temp4 = double(0);
ERROR_CUR = double(0);                      %Current error rate
ERROR_MAX = double(0);                       %Maximum error rate
ERROR = zeros((2^N2-1)*(2^N2-1),1);          %Error vector
MUL_OUT_ERROR = zeros((2^N2-1)*(2^N2-1),1);   %Multiplier output with
tailing ones
MUL_OUT_NOERROR = zeros((2^N2-1)*(2^N2-1),1);  %Actual multiplier result
COUNT = 0;                          %Error counter

%Initialization of inputs A & B
for i=1:(2^N2-1)
   Temp1 = de2bi(i,N2,'left-msb');
   for j=1:N2
      INA(i,j) = Temp1(j);
      INB(i,j) = Temp1(j);
   end
end
% for k=1:(2^N2-1)
%    Temp2 = de2bi(k,N2,'left-msb');
%    for l=1:N2
%       INB(k,l) = Temp2(l);
%    end
% end

%Computation of A*B
p = 0;
% u = 0;
for r=1:2^N2-1
   u = zeros(1,N1);
   u(1:N2) = INB(r,:);
   u(N2+1:N1) = randint(1,N2);
   u_int = bi2de(u,'left-msb');
   n1 = 0;
   if(bi2de(u(N2+1:N1),'left-msb')~=0)
      while(u(N2+n1+1)==0)
```

```
            n1 = n1+1;
         end
      end
   %     v = 0;
      for s=1:2^N2-1
         v = zeros (1,N1);
         v(1:N2) = INA(s,:);
         v(N2+1:N1) = randint(1,N2);
         v_int = bi2de(v,'left-msb');
         n2 = 0;
         if(bi2de(v(N2+1:N1),'left-msb')~=0)
            while(v(N2+n2+1)==0)
               n2 = n2+1;
            end
         end
         PP_TEMP = zeros(N2,N2);
         PP_TEMP_SORTED = zeros(N2,N2);
         PP_TEMP = INB(r,:)'*INA(s,:);
         j=0;
         for i=N2:-1:1
            j = j+1;
            for k=1:N2
               PP_TEMP_SORTED(j,k) = PP_TEMP(i,k);
            end
         end
         k=0;
         PP = zeros(N2,N1);
         for i=0:N2-1
            k = N1-N2-i;
            m=0;
            for l=1:N2
               k = k+1;
               PP(i+1,k) = PP_TEMP_SORTED(i+1,l);
            end
         end
         %Computation of manipulated multiplier result
         MUL_OUT_MAN = ones(1,2*N1);
   %        MUL_OUT_MAN = zeros(1,2*N1);
         PS = zeros(1,N1);          %Partial sum vector
         Co = 0;                    %Carry out flag
         for i=1:N2
            for j=N1:-1:1
               Ci = Co;
               Co = bitor((PS(j)*PP(i,j)),bitor(PS(j),PP(i,j))*Ci);
               if (Ci==1)
                  Temp3 = ~PS(j)*(~PP(i,j));
```

```
                Temp4 = PS(j) * PP(i,j);
             else
                Temp3 = ~PS(j)*PP(i,j);
                Temp4 = PS(j)*(~PP(i,j));
             end
             PS(j) = bitor(Temp3, Temp4);
          end
       end
       for i=1:N1
          MUL_OUT_MAN(i) = PS(i);
       end
       if (n1<=n2)
          n = n1;
       else
          n = n2;
       end
       if (n~=0)
          MUL_OUT_MAN(N1+1:N1+1+n-1) = 0;
       end
       %Error computation
       p = p+1;
       MUL_OUT_MAN_DEC = bi2de(MUL_OUT_MAN,'left-msb');
       MUL_OUT_ACT = u_int*v_int;
       ERROR_CUR = abs((MUL_OUT_MAN_DEC-
MUL_OUT_ACT)/MUL_OUT_ACT);
       MUL_OUT_NOERROR(p,1) = MUL_OUT_ACT;
       MUL_OUT_ERROR(p,1) = MUL_OUT_MAN_DEC;
       ERROR(p,1) = ERROR_CUR;
       if ERROR_CUR>=0.1
          COUNT = COUNT+1;
       end
       if (ERROR_CUR>ERROR_MAX)
          MUL_OUT_MAN_MAX = MUL_OUT_MAN;
          MUL_OUT_ACT_MAX = MUL_OUT_ACT;
          ERROR_MAX = ERROR_CUR;
       end
    end
end
ERROR_PROB = COUNT/length(ERROR);
DIM_MAX = 65535;
ERROR_DIM = length(ERROR);
BUFFER_CNT = 0;
if length(ERROR)>DIM_MAX
   BUFFER_CNT = ceil(length(ERROR)/DIM_MAX);
      switch BUFFER_CNT
      case 2
      BUFFER_1 = zeros(DIM_MAX,1);
      BUFFER_2 = zeros(ERROR_DIM-DIM_MAX,1);
      BUFFER_1(:) = ERROR(1:DIM_MAX);
      BUFFER_2(:) = ERROR(DIM_MAX+1:DIM_MAX+length(BUFFER_2));
      x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_1,'ERROR','A1')
      x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_2,'ERROR','B1')
   case 3
      BUFFER_1 = zeros(DIM_MAX,1);
      BUFFER_2 = zeros(DIM_MAX,1);
      BUFFER_3 = zeros(ERROR_DIM-(BUFFER_CNT-1)*DIM_MAX,1);
      BUFFER_1(:) = ERROR(1:DIM_MAX);
      BUFFER_2(:) = ERROR(DIM_MAX+1:2*DIM_MAX);
      BUFFER_3(:) =
      ERROR(2*DIM_MAX+1:2*DIM_MAX+length(BUFFER_3));
      x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_1,'ERROR','A1')
      x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_2,'ERROR','B1')
      x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_3,'ERROR','C1')
   case 4
      BUFFER_1 = zeros(DIM_MAX,1);
      BUFFER_2 = zeros(DIM_MAX,1);
      BUFFER_3 = zeros(DIM_MAX,1);
      BUFFER_4 = zeros(ERROR_DIM-(BUFFER_CNT-1)*DIM_MAX,1);
      BUFFER_1(:) = ERROR(1:DIM_MAX);
      BUFFER_2(:) = ERROR(DIM_MAX+1:2*DIM_MAX);
      BUFFER_3(:) = ERROR(2*DIM_MAX+1:3*DIM_MAX);
      BUFFER_4(:) =
      ERROR(3*DIM_MAX+1:3*DIM_MAX+length(BUFFER_4));
      x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_1,'ERROR','A1')
      x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_2,'ERROR','B1')
      x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_3,'ERROR','C1')
      x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_4,'ERROR','D1')
   case 5
      BUFFER_1 = zeros(DIM_MAX,1);
      BUFFER_2 = zeros(DIM_MAX,1);
      BUFFER_3 = zeros(DIM_MAX,1);
      BUFFER_4 = zeros(DIM_MAX,1);
      BUFFER_5 = zeros(ERROR_DIM-(BUFFER_CNT-1)*DIM_MAX,1);
      BUFFER_1(:) = ERROR(1:DIM_MAX);
      BUFFER_2(:) = ERROR(DIM_MAX+1:2*DIM_MAX);
      BUFFER_3(:) = ERROR(2*DIM_MAX+1:3*DIM_MAX);
      BUFFER_4(:) = ERROR(3*DIM_MAX+1:4*DIM_MAX);
      BUFFER_5(:) =
      ERROR(4*DIM_MAX+1:4*DIM_MAX+length(BUFFER_5));
      x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_1,'ERROR','A1')
      x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_2,'ERROR','B1')
      x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_3,'ERROR','C1')
```

```
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_4,'ERROR','D1')
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_5,'ERROR','E1')
case 6
    BUFFER_1 = zeros(DIM_MAX,1);
    BUFFER_2 = zeros(DIM_MAX,1);
    BUFFER_3 = zeros(DIM_MAX,1);
    BUFFER_4 = zeros(DIM_MAX,1);
    BUFFER_5 = zeros(DIM_MAX,1);
    BUFFER_6 = zeros(ERROR_DIM-(BUFFER_CNT-1)*DIM_MAX,1);
    BUFFER_1(:) = ERROR(1:DIM_MAX);
    BUFFER_2(:) = ERROR(DIM_MAX+1:2*DIM_MAX);
    BUFFER_3(:) = ERROR(2*DIM_MAX+1:3*DIM_MAX);
    BUFFER_4(:) = ERROR(3*DIM_MAX+1:4*DIM_MAX);
    BUFFER_5(:) = ERROR(4*DIM_MAX+1:5*DIM_MAX);
    BUFFER_6(:) =
    ERROR(5*DIM_MAX+1:5*DIM_MAX+length(BUFFER_6));
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_1,'ERROR','A1')
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_2,'ERROR','B1')
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_3,'ERROR','C1')
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_4,'ERROR','D1')
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_5,'ERROR','E1')
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_6,'ERROR','F1')
case 7
    BUFFER_1 = zeros(DIM_MAX,1);
    BUFFER_2 = zeros(DIM_MAX,1);
    BUFFER_3 = zeros(DIM_MAX,1);
    BUFFER_4 = zeros(DIM_MAX,1);
    BUFFER_5 = zeros(DIM_MAX,1);
    BUFFER_6 = zeros(DIM_MAX,1);
    BUFFER_7 = zeros(ERROR_DIM-(BUFFER_CNT-1)*DIM_MAX,1);
    BUFFER_1(:) = ERROR(1:DIM_MAX);
    BUFFER_2(:) = ERROR(DIM_MAX+1:2*DIM_MAX);
    BUFFER_3(:) = ERROR(2*DIM_MAX+1:3*DIM_MAX);
    BUFFER_4(:) = ERROR(3*DIM_MAX+1:4*DIM_MAX);
    BUFFER_5(:) = ERROR(4*DIM_MAX+1:5*DIM_MAX);
    BUFFER_6(:) = ERROR(5*DIM_MAX+1:6*DIM_MAX);
    BUFFER_7(:) =
    ERROR(6*DIM_MAX+1:6*DIM_MAX+length(BUFFER_7));
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_1,'ERROR','A1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_2,'ERROR','B1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_3,'ERROR','C1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_4,'ERROR','D1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_5,'ERROR','E1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_6,'ERROR','F1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_7,'ERROR','G1');
case 8
```

```
    BUFFER_1 = zeros(DIM_MAX,1);
    BUFFER_2 = zeros(DIM_MAX,1);
    BUFFER_3 = zeros(DIM_MAX,1);
    BUFFER_4 = zeros(DIM_MAX,1);
    BUFFER_5 = zeros(DIM_MAX,1);
    BUFFER_6 = zeros(DIM_MAX,1);
    BUFFER_7 = zeros(DIM_MAX,1);
    BUFFER_8 = zeros(ERROR_DIM-(BUFFER_CNT-1)*DIM_MAX,1);
    BUFFER_1(:) = ERROR(1:DIM_MAX);
    BUFFER_2(:) = ERROR(DIM_MAX+1:2*DIM_MAX);
    BUFFER_3(:) = ERROR(2*DIM_MAX+1:3*DIM_MAX);
    BUFFER_4(:) = ERROR(3*DIM_MAX+1:4*DIM_MAX);
    BUFFER_5(:) = ERROR(4*DIM_MAX+1:5*DIM_MAX);
    BUFFER_6(:) = ERROR(5*DIM_MAX+1:6*DIM_MAX);
    BUFFER_7(:) = ERROR(6*DIM_MAX+1:7*DIM_MAX);
    BUFFER_8(:) =
    ERROR(7*DIM_MAX+1:7*DIM_MAX+length(BUFFER_8));
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_1,'ERROR','A1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_2,'ERROR','B1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_3,'ERROR','C1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_4,'ERROR','D1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_5,'ERROR','E1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_6,'ERROR','F1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_7,'ERROR','G1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_8,'ERROR','H1');
case 9
    BUFFER_1 = zeros(DIM_MAX,1);
    BUFFER_2 = zeros(DIM_MAX,1);
    BUFFER_3 = zeros(DIM_MAX,1);
    BUFFER_4 = zeros(DIM_MAX,1);
    BUFFER_5 = zeros(DIM_MAX,1);
    BUFFER_6 = zeros(DIM_MAX,1);
    BUFFER_7 = zeros(DIM_MAX,1);
    BUFFER_8 = zeros(DIM_MAX,1);
    BUFFER_9 = zeros(ERROR_DIM-(BUFFER_CNT-1)*DIM_MAX,1);
    BUFFER_1(:) = ERROR(1:DIM_MAX);
    BUFFER_2(:) = ERROR(DIM_MAX+1:2*DIM_MAX);
    BUFFER_3(:) = ERROR(2*DIM_MAX+1:3*DIM_MAX);
    BUFFER_4(:) = ERROR(3*DIM_MAX+1:4*DIM_MAX);
    BUFFER_5(:) = ERROR(4*DIM_MAX+1:5*DIM_MAX);
    BUFFER_6(:) = ERROR(5*DIM_MAX+1:6*DIM_MAX);
    BUFFER_7(:) = ERROR(6*DIM_MAX+1:7*DIM_MAX);
    BUFFER_8(:) = ERROR(7*DIM_MAX+1:8*DIM_MAX);
    BUFFER_9(:) =
    ERROR(8*DIM_MAX+1:8*DIM_MAX+length(BUFFER_9));
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_1,'ERROR','A1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_2,'ERROR','B1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_3,'ERROR','C1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_4,'ERROR','D1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_5,'ERROR','E1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_6,'ERROR','F1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_7,'ERROR','G1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_8,'ERROR','H1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_9,'ERROR','I1');
case 10
    BUFFER_1 = zeros(DIM_MAX,1);
    BUFFER_2 = zeros(DIM_MAX,1);
    BUFFER_3 = zeros(DIM_MAX,1);
    BUFFER_4 = zeros(DIM_MAX,1);
    BUFFER_5 = zeros(DIM_MAX,1);
    BUFFER_6 = zeros(DIM_MAX,1);
    BUFFER_7 = zeros(DIM_MAX,1);
    BUFFER_8 = zeros(DIM_MAX,1);
    BUFFER_9 = zeros(DIM_MAX,1);
    BUFFER_10 = zeros(ERROR_DIM-(BUFFER_CNT-1)*DIM_MAX,1);
    BUFFER_1(:) = ERROR(1:DIM_MAX);
    BUFFER_2(:) = ERROR(DIM_MAX+1:2*DIM_MAX);
    BUFFER_3(:) = ERROR(2*DIM_MAX+1:3*DIM_MAX);
    BUFFER_4(:) = ERROR(3*DIM_MAX+1:4*DIM_MAX);
    BUFFER_5(:) = ERROR(4*DIM_MAX+1:5*DIM_MAX);
    BUFFER_6(:) = ERROR(5*DIM_MAX+1:6*DIM_MAX);
    BUFFER_7(:) = ERROR(6*DIM_MAX+1:7*DIM_MAX);
    BUFFER_8(:) = ERROR(7*DIM_MAX+1:8*DIM_MAX);
    BUFFER_9(:) = ERROR(8*DIM_MAX+1:9*DIM_MAX);
    BUFFER_10(:) =
    ERROR(9*DIM_MAX+1:9*DIM_MAX+length(BUFFER_10));
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_1,'ERROR','A1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_2,'ERROR','B1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_3,'ERROR','C1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_4,'ERROR','D1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_5,'ERROR','E1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_6,'ERROR','F1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_7,'ERROR','G1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_8,'ERROR','H1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_9,'ERROR','I1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_10,'ERROR','J1');
case 11
    BUFFER_1 = zeros(DIM_MAX,1);
    BUFFER_2 = zeros(DIM_MAX,1);
    BUFFER_3 = zeros(DIM_MAX,1);
    BUFFER_4 = zeros(DIM_MAX,1);
    BUFFER_5 = zeros(DIM_MAX,1);
```

```
            BUFFER_6 = zeros(DIM_MAX,1);
            BUFFER_7 = zeros(DIM_MAX,1);
            BUFFER_8 = zeros(DIM_MAX,1);
            BUFFER_9 = zeros(DIM_MAX,1);
            BUFFER_10 = zeros(DIM_MAX,1);
            BUFFER_11 = zeros(ERROR_DIM-(BUFFER_CNT-1)*DIM_MAX,1);
            BUFFER_1(:) = ERROR(1:DIM_MAX);
            BUFFER_2(:) = ERROR(DIM_MAX+1:2*DIM_MAX);
            BUFFER_3(:) = ERROR(2*DIM_MAX+1:3*DIM_MAX);
            BUFFER_4(:) = ERROR(3*DIM_MAX+1:4*DIM_MAX);
            BUFFER_5(:) = ERROR(4*DIM_MAX+1:5*DIM_MAX);
            BUFFER_6(:) = ERROR(5*DIM_MAX+1:6*DIM_MAX);
            BUFFER_7(:) = ERROR(6*DIM_MAX+1:7*DIM_MAX);
            BUFFER_8(:) = ERROR(7*DIM_MAX+1:8*DIM_MAX);
            BUFFER_9(:) = ERROR(8*DIM_MAX+1:9*DIM_MAX);
            BUFFER_10(:) = ERROR(9*DIM_MAX+1:10*DIM_MAX);
            BUFFER_11(:) =
ERROR(10*DIM_MAX+1:10*DIM_MAX+length(BUFFER_11));
            x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_1,'ERROR','A1');
            x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_2,'ERROR','B1');
            x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_3,'ERROR','C1');
            x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_4,'ERROR','D1');
            x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_5,'ERROR','E1');
            x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_6,'ERROR','F1');
            x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_7,'ERROR','G1');
            x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_8,'ERROR','H1');
            x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_9,'ERROR','I1');
            x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_10,'ERROR','J1');
            x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_11,'ERROR','K1');
        case 12
            BUFFER_1 = zeros(DIM_MAX,1);
            BUFFER_2 = zeros(DIM_MAX,1);
            BUFFER_3 = zeros(DIM_MAX,1);
            BUFFER_4 = zeros(DIM_MAX,1);
            BUFFER_5 = zeros(DIM_MAX,1);
            BUFFER_6 = zeros(DIM_MAX,1);
            BUFFER_7 = zeros(DIM_MAX,1);
            BUFFER_8 = zeros(DIM_MAX,1);
            BUFFER_9 = zeros(DIM_MAX,1);
            BUFFER_10 = zeros(DIM_MAX,1);
            BUFFER_11 = zeros(DIM_MAX,1);
            BUFFER_12 = zeros(ERROR_DIM-(BUFFER_CNT-1)*DIM_MAX,1);
            BUFFER_1(:) = ERROR(1:DIM_MAX);
            BUFFER_2(:) = ERROR(DIM_MAX+1:2*DIM_MAX);
            BUFFER_3(:) = ERROR(2*DIM_MAX+1:3*DIM_MAX);
            BUFFER_4(:) = ERROR(3*DIM_MAX+1:4*DIM_MAX);
```

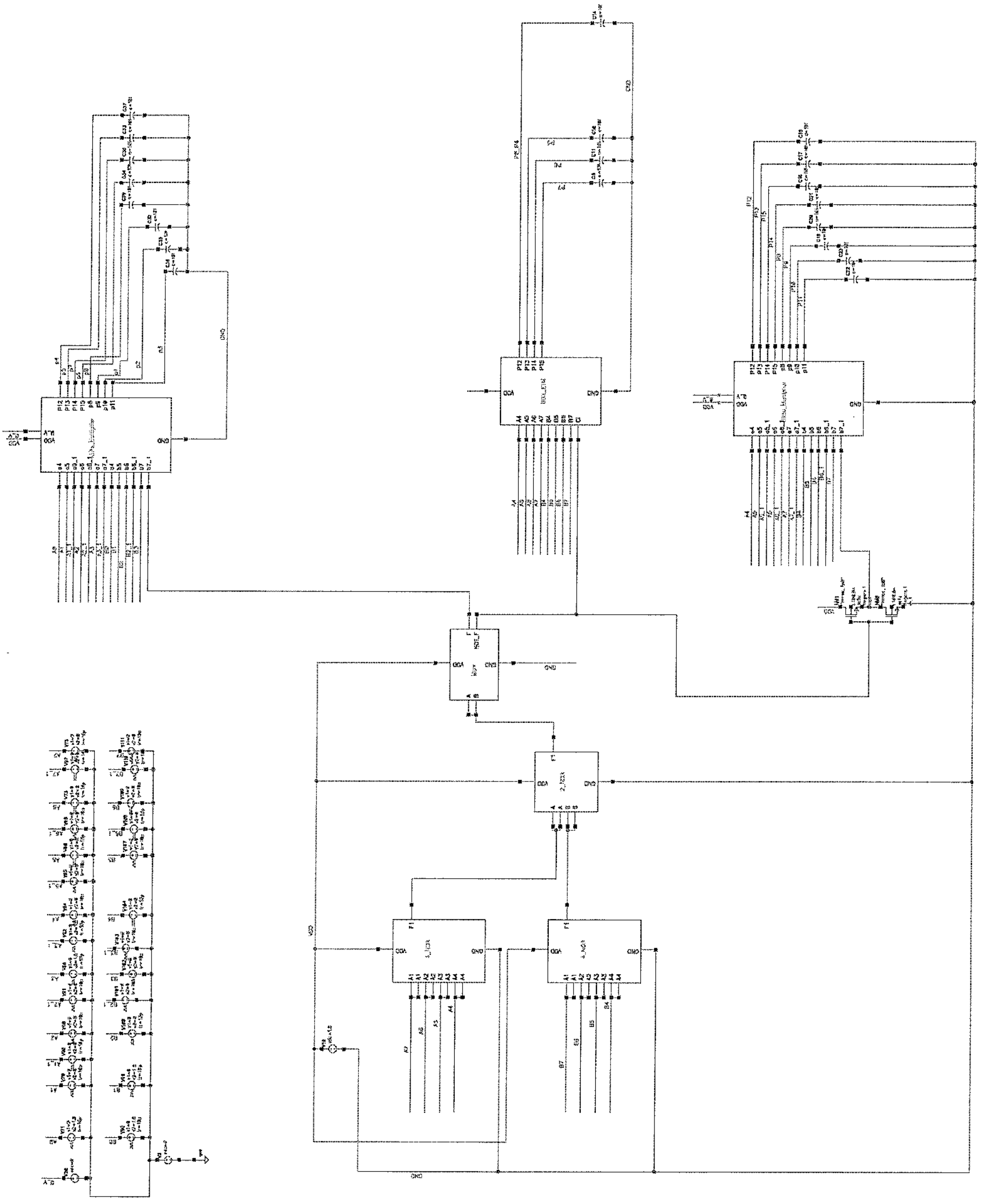
```
    BUFFER_5(:) = ERROR(4*DIM_MAX+1:5*DIM_MAX);
    BUFFER_6(:) = ERROR(5*DIM_MAX+1:6*DIM_MAX);
    BUFFER_7(:) = ERROR(6*DIM_MAX+1:7*DIM_MAX);
    BUFFER_8(:) = ERROR(7*DIM_MAX+1:8*DIM_MAX);
    BUFFER_9(:) = ERROR(8*DIM_MAX+1:9*DIM_MAX);
    BUFFER_10(:) = ERROR(9*DIM_MAX+1:10*DIM_MAX);
    BUFFER_11(:) = ERROR(10*DIM_MAX+1:11*DIM_MAX);
    BUFFER_12(:) =
    ERROR(11*DIM_MAX+1:11*DIM_MAX+length(BUFFER_12));
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_1,'ERROR','A1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_2,'ERROR','B1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_3,'ERROR','C1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_4,'ERROR','D1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_5,'ERROR','E1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_6,'ERROR','F1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_7,'ERROR','G1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_8,'ERROR','H1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_9,'ERROR','I1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_10,'ERROR','J1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_11,'ERROR','K1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_12,'ERROR','L1');
case 13
    BUFFER_1 = zeros(DIM_MAX,1);
    BUFFER_2 = zeros(DIM_MAX,1);
    BUFFER_3 = zeros(DIM_MAX,1);
    BUFFER_4 = zeros(DIM_MAX,1);
    BUFFER_5 = zeros(DIM_MAX,1);
    BUFFER_6 = zeros(DIM_MAX,1);
    BUFFER_7 = zeros(DIM_MAX,1);
    BUFFER_8 = zeros(DIM_MAX,1);
    BUFFER_9 = zeros(DIM_MAX,1);
    BUFFER_10 = zeros(DIM_MAX,1);
    BUFFER_11 = zeros(DIM_MAX,1);
    BUFFER_12 = zeros(DIM_MAX,1);
    BUFFER_13 = zeros(ERROR_DIM-(BUFFER_CNT-1)*DIM_MAX,1);
    BUFFER_1(:) = ERROR(1:DIM_MAX);
    BUFFER_2(:) = ERROR(DIM_MAX+1:2*DIM_MAX);
    BUFFER_3(:) = ERROR(2*DIM_MAX+1:3*DIM_MAX);
    BUFFER_4(:) = ERROR(3*DIM_MAX+1:4*DIM_MAX);
    BUFFER_5(:) = ERROR(4*DIM_MAX+1:5*DIM_MAX);
    BUFFER_6(:) = ERROR(5*DIM_MAX+1:6*DIM_MAX);
    BUFFER_7(:) = ERROR(6*DIM_MAX+1:7*DIM_MAX);
    BUFFER_8(:) = ERROR(7*DIM_MAX+1:8*DIM_MAX);
    BUFFER_9(:) = ERROR(8*DIM_MAX+1:9*DIM_MAX);
    BUFFER_10(:) = ERROR(9*DIM_MAX+1:10*DIM_MAX);
    BUFFER_11(:) = ERROR(10*DIM_MAX+1:11*DIM_MAX);
    BUFFER_12(:) = ERROR(11*DIM_MAX+1:12*DIM_MAX);
    BUFFER_13(:) =
    ERROR(12*DIM_MAX+1:12*DIM_MAX+length(BUFFER_13));
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_1,'ERROR','A1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_2,'ERROR','B1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_3,'ERROR','C1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_4,'ERROR','D1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_5,'ERROR','E1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_6,'ERROR','F1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_7,'ERROR','G1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_8,'ERROR','H1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_9,'ERROR','I1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_10,'ERROR','J1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_11,'ERROR','K1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_12,'ERROR','L1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_13,'ERROR','M1');
case 14
    BUFFER_1 = zeros(DIM_MAX,1);
    BUFFER_2 = zeros(DIM_MAX,1);
    BUFFER_3 = zeros(DIM_MAX,1);
    BUFFER_4 = zeros(DIM_MAX,1);
    BUFFER_5 = zeros(DIM_MAX,1);
    BUFFER_6 = zeros(DIM_MAX,1);
    BUFFER_7 = zeros(DIM_MAX,1);
    BUFFER_8 = zeros(DIM_MAX,1);
    BUFFER_9 = zeros(DIM_MAX,1);
    BUFFER_10 = zeros(DIM_MAX,1);
    BUFFER_11 = zeros(DIM_MAX,1);
    BUFFER_12 = zeros(DIM_MAX,1);
    BUFFER_13 = zeros(DIM_MAX,1);
    BUFFER_14 = zeros(ERROR_DIM-(BUFFER_CNT-1)*DIM_MAX,1);
    BUFFER_1(:) = ERROR(1:DIM_MAX);
    BUFFER_2(:) = ERROR(DIM_MAX+1:2*DIM_MAX);
    BUFFER_3(:) = ERROR(2*DIM_MAX+1:3*DIM_MAX);
    BUFFER_4(:) = ERROR(3*DIM_MAX+1:4*DIM_MAX);
    BUFFER_5(:) = ERROR(4*DIM_MAX+1:5*DIM_MAX);
    BUFFER_6(:) = ERROR(5*DIM_MAX+1:6*DIM_MAX);
    BUFFER_7(:) = ERROR(6*DIM_MAX+1:7*DIM_MAX);
    BUFFER_8(:) = ERROR(7*DIM_MAX+1:8*DIM_MAX);
    BUFFER_9(:) = ERROR(8*DIM_MAX+1:9*DIM_MAX);
    BUFFER_10(:) = ERROR(9*DIM_MAX+1:10*DIM_MAX);
    BUFFER_11(:) = ERROR(10*DIM_MAX+1:11*DIM_MAX);
    BUFFER_12(:) = ERROR(11*DIM_MAX+1:12*DIM_MAX);
    BUFFER_13(:) = ERROR(12*DIM_MAX+1:13*DIM_MAX);
    BUFFER_14(:) =
    ERROR(13*DIM_MAX+1:13*DIM_MAX+length(BUFFER_14));
```

```
        x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_1,'ERROR','A1');
        x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_2,'ERROR','B1');
        x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_3,'ERROR','C1');
        x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_4,'ERROR','D1');
        x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_5,'ERROR','E1');
        x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_6,'ERROR','F1');
        x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_7,'ERROR','G1');
        x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_8,'ERROR','H1');
        x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_9,'ERROR','I1');
        x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_10,'ERROR','J1');
        x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_11,'ERROR','K1');
        x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_12,'ERROR','L1');
        x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_13,'ERROR','M1');
        x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_14,'ERROR','N1');
    case 15
        BUFFER_1 = zeros(DIM_MAX,1);
        BUFFER_2 = zeros(DIM_MAX,1);
        BUFFER_3 = zeros(DIM_MAX,1);
        BUFFER_4 = zeros(DIM_MAX,1);
        BUFFER_5 = zeros(DIM_MAX,1);
        BUFFER_6 = zeros(DIM_MAX,1);
        BUFFER_7 = zeros(DIM_MAX,1);
        BUFFER_8 = zeros(DIM_MAX,1);
        BUFFER_9 = zeros(DIM_MAX,1);
        BUFFER_10 = zeros(DIM_MAX,1);
        BUFFER_11 = zeros(DIM_MAX,1);
        BUFFER_12 = zeros(DIM_MAX,1);
        BUFFER_13 = zeros(DIM_MAX,1);
        BUFFER_14 = zeros(DIM_MAX,1);
        BUFFER_15 = zeros(ERROR_DIM-(BUFFER_CNT-1)*DIM_MAX,1);
        BUFFER_1(:) = ERROR(1:DIM_MAX);
        BUFFER_2(:) = ERROR(DIM_MAX+1:2*DIM_MAX);
        BUFFER_3(:) = ERROR(2*DIM_MAX+1:3*DIM_MAX);
        BUFFER_4(:) = ERROR(3*DIM_MAX+1:4*DIM_MAX);
        BUFFER_5(:) = ERROR(4*DIM_MAX+1:5*DIM_MAX);
        BUFFER_6(:) = ERROR(5*DIM_MAX+1:6*DIM_MAX);
        BUFFER_7(:) = ERROR(6*DIM_MAX+1:7*DIM_MAX);
        BUFFER_8(:) = ERROR(7*DIM_MAX+1:8*DIM_MAX);
        BUFFER_9(:) = ERROR(8*DIM_MAX+1:9*DIM_MAX);
        BUFFER_10(:) = ERROR(9*DIM_MAX+1:10*DIM_MAX);
        BUFFER_11(:) = ERROR(10*DIM_MAX+1:11*DIM_MAX);
        BUFFER_12(:) = ERROR(11*DIM_MAX+1:12*DIM_MAX);
        BUFFER_13(:) = ERROR(12*DIM_MAX+1:13*DIM_MAX);
        BUFFER_14(:) = ERROR(13*DIM_MAX+1:14*DIM_MAX);
        BUFFER_15(:) =
        ERROR(14*DIM_MAX+1:14*DIM_MAX+length(BUFFER_15));
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_1,'ERROR','A1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_2,'ERROR','B1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_3,'ERROR','C1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_4,'ERROR','D1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_5,'ERROR','E1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_6,'ERROR','F1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_7,'ERROR','G1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_8,'ERROR','H1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_9,'ERROR','I1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_10,'ERROR','J1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_11,'ERROR','K1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_12,'ERROR','L1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_13,'ERROR','M1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_14,'ERROR','N1');
    x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_15,'ERROR','O1');
case 16
    BUFFER_1 = zeros(DIM_MAX,1);
    BUFFER_2 = zeros(DIM_MAX,1);
    BUFFER_3 = zeros(DIM_MAX,1);
    BUFFER_4 = zeros(DIM_MAX,1);
    BUFFER_5 = zeros(DIM_MAX,1);
    BUFFER_6 = zeros(DIM_MAX,1);
    BUFFER_7 = zeros(DIM_MAX,1);
    BUFFER_8 = zeros(DIM_MAX,1);
    BUFFER_9 = zeros(DIM_MAX,1);
    BUFFER_10 = zeros(DIM_MAX,1);
    BUFFER_11 = zeros(DIM_MAX,1);
    BUFFER_12 = zeros(DIM_MAX,1);
    BUFFER_13 = zeros(DIM_MAX,1);
    BUFFER_14 = zeros(DIM_MAX,1);
    BUFFER_15 = zeros(DIM_MAX,1);
    BUFFER_16 = zeros(ERROR_DIM-(BUFFER_CNT-1)*DIM_MAX,1);
    BUFFER_1(:) = ERROR(1:DIM_MAX);
    BUFFER_2(:) = ERROR(DIM_MAX+1:2*DIM_MAX);
    BUFFER_3(:) = ERROR(2*DIM_MAX+1:3*DIM_MAX);
    BUFFER_4(:) = ERROR(3*DIM_MAX+1:4*DIM_MAX);
    BUFFER_5(:) = ERROR(4*DIM_MAX+1:5*DIM_MAX);
    BUFFER_6(:) = ERROR(5*DIM_MAX+1:6*DIM_MAX);
    BUFFER_7(:) = ERROR(6*DIM_MAX+1:7*DIM_MAX);
    BUFFER_8(:) = ERROR(7*DIM_MAX+1:8*DIM_MAX);
    BUFFER_9(:) = ERROR(8*DIM_MAX+1:9*DIM_MAX);
    BUFFER_10(:) = ERROR(9*DIM_MAX+1:10*DIM_MAX);
    BUFFER_11(:) = ERROR(10*DIM_MAX+1:11*DIM_MAX);
    BUFFER_12(:) = ERROR(11*DIM_MAX+1:12*DIM_MAX);
    BUFFER_13(:) = ERROR(12*DIM_MAX+1:13*DIM_MAX);
    BUFFER_14(:) = ERROR(13*DIM_MAX+1:14*DIM_MAX);
```

```
            BUFFER_15(:) = ERROR(14*DIM_MAX+1:15*DIM_MAX);
            BUFFER_16(:) =
            ERROR(15*DIM_MAX+1:15*DIM_MAX+length(BUFFER_16));
            x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_1,'ERROR','A1');
            x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_2,'ERROR','B1');
            x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_3,'ERROR','C1');
            x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_4,'ERROR','D1');
            x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_5,'ERROR','E1');
            x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_6,'ERROR','F1');
            x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_7,'ERROR','G1');
            x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_8,'ERROR','H1');
            x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_9,'ERROR','I1');
            x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_10,'ERROR','J1');
            x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_11,'ERROR','K1');
            x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_12,'ERROR','L1');
            x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_13,'ERROR','M1');
            x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_14,'ERROR','N1');
            x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_15,'ERROR','O1');
            x = xlswrite('ERROR_ANALYSIS.xls',BUFFER_16,'ERROR','P1');
    end
end
```

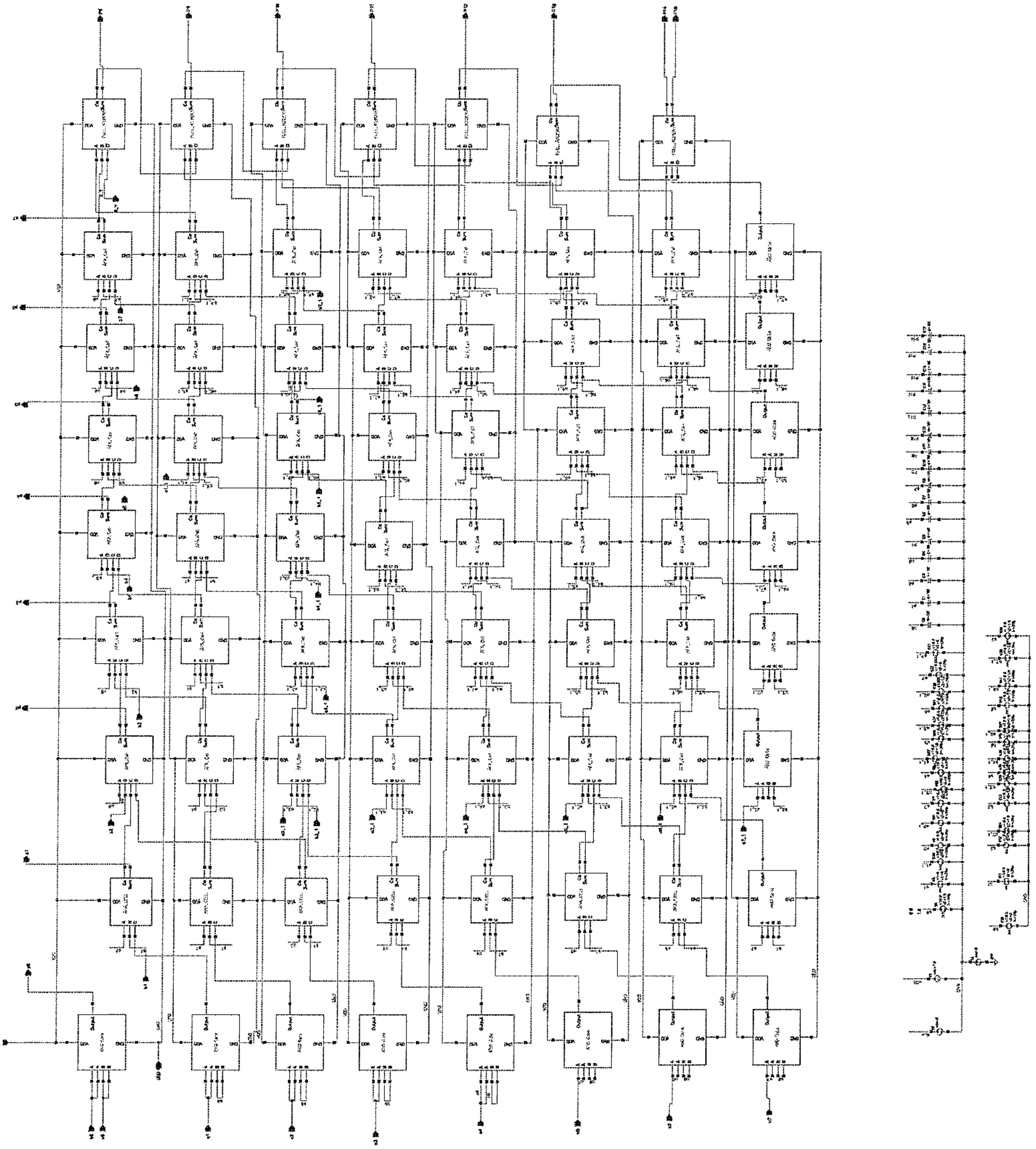
Appendix B

8-bits proposed ETM implementation using Cadence Software


VDD
GND
mux

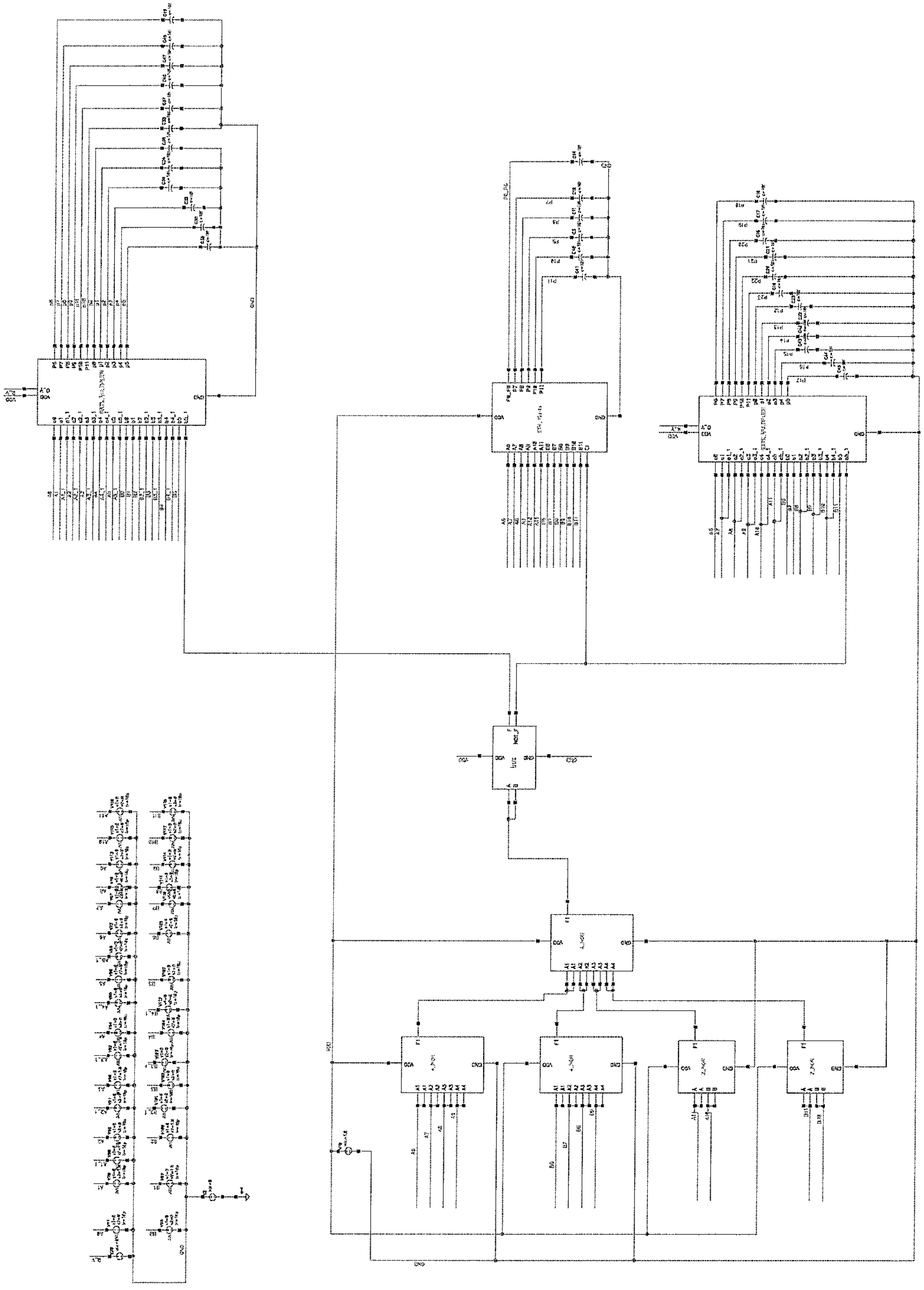
Appendix C

Conventional 8-bits Multiplier implementation using Cadence Software



Appendix D

12-bits proposed ETM implementation using Cadence Software



Appendix E

Conventional 12-bits Multiplier implementation using Cadence Software