**A LOW-POWER AND HIGH-ACCURACY APPROXIMATE MULTIPLIER WITH RECONFIGURABLE TRUNCATION**

Abstract:

In this project, we are going to design approximate multiplier by designing an approximate compressor with high accuracy compared to existing approximate compressors. Multipliers are among the most critical arithmetic functional units in many applications, and those applications commonly require many multiplications which result in significant power consumption. For applications that have error tolerance, employing an approximate multiplier is an emerging method to reduce critical path delay and power consumption. An approximate multiplier can trade off accuracy for lower energy and higher performance. In this paper, we not only propose an approximate 4-2 compressor with high accuracy, but also an adjustable approximate multiplier that can dynamically truncate partial products to achieve variable accuracy requirements. In addition, we also propose a simple error compensation circuit to reduce error distance. The proposed approximate multiplier can adjust the accuracy and power required for multiplications at run-time based on the users’ requirement. Experimental results show that the compared to existing accurate Wallace multiplier, the proposed adjustable approximate multiplier can be reduced in parameter values. The synthesis and simulation of the proposed designs can be implemented using Xilinx Vivado 2018.3.

**Key words:** Approximate computing, approximate multiplier, CNN accelerator, deep learning, high precision, reconfigurable approximate design.

**CHAPTER I**

**INTRODUCTION**

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. One common technique for energy efficiency CMOS circuits is the reduction of the supply voltage. However, there are two drawbacks: first is the increase in the gates delay. To overcome this problem, the threshold voltage to be scaled down; and the other drawback is the degradation of noise immunity of the circuits.

Reducing power dissipation has become an important objective in the design of digital circuits. One common technique for reducing power is to reduce the supply voltage. For CMOS circuits the cost of lower supply voltage is lower performance. Scaling the threshold voltage can limit this performance loss somewhat but results in increased static power dissipation. Burr et al. “Cryogenic ultra low power CMOS”, “ Ultra low power CMOS technology” have shown that if one optimizes for minimum energy, then operating in the subthreshold region is advantageous. Since minimum energy solutions are generally low performance solutions, we look instead at both energy and delay during optimization and use the energy-delay product as a measure of the efficiency of the circuit. In this paper we examine the effects of lowering the supply and threshold voltages on the energy efficiency of CMOS circuits.

Hence, the increasing noise sensitivity has become an important concern in the design of devices, circuits and systems. The data processed by many digital systems may in actual fact have already contained errors.

In conventional digital VLSI design, one usually assumes that a usable circuit/system should always provide definite and accurate results. But in fact, such perfect operations are seldom needed in our nondigital worldly experiences. The world accepts “analog computation,” which generates “good enough” results rather than totally accurate results. The data processed by many digital systems may already contain errors. In many applications, such as a communication system, the analog signal coming from the outside world must first be sampled before being converted to digital data. The digital data are then processed and transmitted in a noisy channel before converting back to an analog signal. During this process, errors may occur anywhere. Furthermore, due to the advances in transistor size scaling, factors such as noise and process variations which are previously insignificant are becoming important in today’s digital IC design.

Based on the characteristic of digital VLSI design, some novel concepts and design techniques have been proposed. The concept of error tolerance (ET) and the PCMOS technology are two of them. According to the definition, a circuit is error tolerant if: 1) it contains defects that cause internal and may cause external errors and 2) the system that incorporates this circuit produces acceptable results. The “imperfect” attribute seems to be not appealing. However, the need for the error-tolerant circuit was foretold in the 2003 International Technology Roadmap for Semiconductors (ITRS). To deal with error-tolerant problems, some truncated adders/multipliers have been reported but are not able to perform well in either its speed, power, area, or accuracy. The “flagged prefixed adder” performs better than the nonflagged version with a 1.3% speed enhancement but at the expense of 2% extra silicon area. As for the “low-error area-efficient fixed-width multipliers”, it may have an area improvement of 46.67% but has average error reaching 12.4%. Of course, not all digital systems can engage the error-tolerant concept. In digital systems such as control systems, the correctness of the output signal is extremely important, and this denies the use of the error tolerant circuit. However, for many digital signal processing (DSP) systems that process signals relating to human senses such as hearing, sight, smell, and touch, e.g., the image processing and speech processing systems, the error-tolerant circuits may be applications.

Because of the advances in VLSI scaling and the near emergence of billion transistor chips, the results of noise, process variations and spot defects will dictate that few such chips will be error-free. And unlike the conventional method, a completely new design style for an ultra-low power multiplier is proposed in this paper. In addition to the power consumption and speed, accuracy is used as a new parameter for the upcoming nano regime. With an innovative and novel multiplication method adopted, an ultra-low-power and high-speed multiplier for Error Tolerant application is introduced in this paper and we name our new design, the Error Tolerant Multiplier (ETM). The novel ETM can achieve enormous improvements on speed performance and power consumption at 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.

Approximate computing is an emerging trend in digital design, relaxing the requisite of exact computation to gain substantial performance improvement in terms of power, speed and area. This approach is becoming more and more important for embedded and mobile systems, characterized by severe energy and speed constraints.

Approximate computing can be fruitfully applied in several error-resilient applications. Examples are multimedia processing, data mining and recognition, machine learning. Multipliers are fundamental subsystems for microprocessors, digital signal processors, and embedded systems with applications ranging from filtering to convolutional neural networks. Unfortunately, multipliers are characterized by complex logic design and constitute one of the most energy-hungry digital blocks. Therefore, approximate multiplier design has become an important research subject in recent years.

A multiplier includes a few basic blocks: partial products generation, partial products reduction and carry-propagate addition. Approximations can be introduced in any of these blocks. For example, truncation of the partial products is a well-established approximation technique in which some of the partial products are not formed and the truncation error is reduced with the help of suitable correction functions.

Energy-efficiency has become the paramount concern in design of computing systems. At the same time, as the computing systems become increasingly embedded and mobile, computational tasks include a growing set of applications that involve media processing (audio, video, graphics, and image), recognition, and data mining.

A common characteristic of the above class of applications is that often a perfect result is not necessary and an approximate or less-than-optimal result is sufficient. It is a familiar feature of image processing, for example, that a range of image sharpness/resolution is acceptable. In data mining, simply a good output of, say, a search result, is hard to distinguish from the best result. Such applications are imprecision-tolerant. There may be multiple sources of imprecision-tolerance: (1) perceptual limitations: these are determined by the ability of the human brain to ‘fill in’ missing information and filter out high-frequency patterns; (2) redundant input data: this redundancy means that an algorithm can be lossy and still be adequate; and (3) noisy inputs. The primary purpose of this paper is to review the recent developments in the area of approximate computing (AC). The term spans a wide set of research activities ranging from programming languages to transistor level. The common underlying thread in these disparate efforts is the search for solutions that allow computing systems to trade energy for quality of the computed result. In this paper we focus on the solutions that involve rethinking of how hardware needs to be designed. To this end, we start with an overview of several related computing paradigms and review some recently proposed approximate arithmetic circuits.

Despite the advances in semiconductor technologies and development of energy-efficient design techniques, the overall energy consumption of computer systems is still rapidly growing at an alarming rate in order to process an ever-increasing amount of information. In particular, as computer systems become pervasive, they are increasingly used to interact with the physical world and process a large amount of data from various sources. Moreover, we expect them to be context-aware and present natural human interfaces.

Consequently, a large number of applications, commonly referred to as Recognition, Mining, and Synthesis (RMS) applications, have emerged and they account for a significant portion of computational resources across the computing spectrum, from mobile and Internet of Things (IoT) devices to large-scale data centers. It is essential to dramatically improve the energy efficiency for these emerging workloads in order to keep pace with the growth of information that needs to be processed. Fortunately, such applications usually feature an intrinsic error-resilience property. They process noisy and redundant data from non-traditional input sources such as various types of sensors (inexact inputs) and the associated algorithms are often stochastic in nature (e.g., iterative algorithms). Moreover, these applications usually do not require to compute a unique or golden numerical result (“acceptable” instead of precise outputs). For example, in multimedia processing, due to the limited perceptual capability of humans, occasional errors such as dropping a particular frame or a small image quality loss often rarely affect a user’s satisfaction. As another example in data analytics, consider two different classifiers that produce similar classification results on a set of example objects. It is very difficult, if not impossible, to tell which one is “better” for the classification of new objects. On the other hand, it is increasingly energy-inefficient to ensure fault-free computations as semiconductor technology advances to nanometer regime. This is because circuits are more sensitive to parameter variations and faults at advanced technology node with low supply voltage and ever-increasing integration density. Conventional fault free computing requires guard bands and redundancies at various levels of design hierarchy for variation tolerance and error correction, causing significant energy overhead. Motivated by the above challenges, one promising solution, known as approximate computing, has attracted significant traction from both academia and industry. By relaxing the numerical equivalence between the specification and implementation of error-tolerant applications, approximate computing deliberately introduces “acceptable errors” into the computing process and promises significant energy-efficiency gains. Considering the fact that traditional Dennard’s scaling provides diminishing returns with technology advancement, leveraging the new source of energy-efficiency provided by approximate computing is increasingly important.

With increasing importance of big data processing and artificial intelligence, an unprecedented challenge has arisen due to the massive amounts of data and complex computations required in these applications. Energy efficient and high-performance general-purpose compute engines, as well as application-specific integrated circuits, are highly demanded to facilitate the development of these new technologies. Meanwhile, exact or high-precision computation is not always necessary. Instead, some small errors can compensate each other or will not have a significant effect in the computed results. Hence, approximate computing (AC) has emerged as a new approach to energy efficient design, as well as to increasing the performance of a computing system, at a limited loss in accuracy.

Approximate computing is an emerging trend in digital design that trades off the requirement of exact computation for improved speed and power performance. This paper proposes novel approximate compressors and 8 bit and 16 bit multiplier designs to evaluate the performance of the proposed compressors Comparison with previously presented approximated multipliers shows that the proposed circuits provide better power or speed for a target precision.

Approximate computing has emerged as a new paradigm for high-performance and energy-efficient design of circuits and systems. For the many approximate arithmetic circuits proposed, it has become critical to understand a design or approximation technique for a specific application to improve performance and energy efficiency with a minimal loss in accuracy. This article aims to provide a comprehensive survey and a comparative evaluation of recently developed approximate arithmetic circuits under different design constraints. Specifically, approximate adders, multipliers, and dividers are synthesized and characterized under optimizations for performance and area. The error and circuit characteristics are then generalized for different classes of designs. The applications of these circuits in image processing and deep neural networks indicate that the circuits with lower error rates or error biases perform better in simple computations, such as the sum of products, whereas more complex accumulative computations that involve multiple matrix multiplications and convolutions are vulnerable to single-sided errors that lead to a large error bias in the computed result. Such complex computations are more sensitive to errors in addition than those in multiplication, so a larger approximation can be tolerated in multipliers than in adders. The use of approximate arithmetic circuits can improve the quality of image processing and deep learning in addition to the benefits in performance and power consumption for these applications.

The overhead on computation units in a processor to deliver high performance and execution efficiency can be leveraged by introducing approximation. Speed of operation which is inversely proportional to the delay of the system requires immense parallel operations that incur huge hardware and power dissipation. Energy and area efficient systems can be realised by relaxing the precision and reliability of the system. In order to maintain the balance between delay, area and power, approximate computing has emerged as a promising solution.

Approximation in arithmetic operations result in faster systems with lesser design complexity and power consumption. The trade-off would be reduction in accuracy, which does not necessarily affect the normal operation for machine learning and multimedia applications. Such applications effectively take advantage of the The associate editor coordinating the review of this manuscript and approving it for publication was Yizhang Jiang . inability of human eye to detect variation in finer details within images and videos. This level of error tolerance is used to design approximate arithmetic circuits for Artificial Intelligence (AI) and Digital Signal Processing (DSP) applications.

Extensive research has been carried out to enhance the efficiency of approximate arithmetic units. In multiplication operation, partial product summation has in-arguably been the major contributor to the power dissipation and delay in the system. Research shows that compressors can reduce the delay associated with the partial product summation. Compressors estimate the count of logic 1 in the input using half adders and/or full adders.

**CHAPTER 2**

**LITERATURE SURVEY**

**Narayanamoorthy, S., Moghaddam, H. A., Liu, Z., Park, T., & Kim, N. S. (2015). Energy-Efficient Approximate Multiplication for Digital Signal Processing and Classification Applications. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 23(6), 1180–1184.**

The need to support various digital signal processing (DSP) and classiﬁcation applications on energy-constrained devices has steadily grown. Such applications often extensively perform matrix multiplications using ﬁxed-point arithmetic while exhibiting tolerance for some computational errors. Hence, improving the energy efﬁciency of multiplications is critical. In this brief, we propose multiplier architectures that can tradeoff computational accuracy with energy consumption at design time. Compared with a precise multiplier, the proposed multiplier can consume less energy/op with average computational error of ∼1%. Finally, it is demonstrated that such a small computational error does not notably impact the quality of DSP and the accuracy of classiﬁcation applications.

**Summary:** In this paper, Power consumption and area can further be reduced.

**Zervakis, G., Xydis, S., Tsoumanis, K., Soudris, D., &Pekmestzi, K. (2015). Hybrid approximate multiplier architectures for improved power-accuracy trade-offs. 2015 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED).**

Approximate computing forms a promising design alternative for inherently error resilient applications, trading accuracy for power savings. In this paper, we exploit multi-level approximation, i.e. at the algorithmic, the logic and the circuit level, to design low power approximate arithmetic architectures for hardware multipliers. Motivated from the limited power savings that approximation techniques can achieve in isolation, we explore hybrid methods that apply simultaneously more than one techniques from different layers. We introduce the concept of perforation for approximate arithmetic circuit design and we explore the newly defined design space of hybrid designs showing that it leads to lower power consumption at every examined error range. To address the increased complexity of the target design space, we introduce an heuristic optimization technique and the corresponding design framework that automatically generates hybrid low-power approximate multipliers requiring a small number of design evaluations, i.e. synthesis, simulation, power and timing analysis. Through extensive experimentation, we show that the proposed techniques converge towards optimal solutions and deliver approximate designs that are always more efficient with respect to state-of-art approaches. Power savings of 11% are reported for small error bounds and more than 30% in case of more relaxed error constraints.

**Summary:** In this paper, power consumption can further be reduced.

**A. Momeni, J. Han,“Design and Analysis of Approximate Compressors for Multiplication” IEEETransactions on Computers.**

Inexact (or approximate) computing is an attractive paradigm for digital processing at nanometric scales. Inexact computing is particularly interesting for computer arithmetic designs. This paper deals with the analysis and design of two new approximate 4-2 compressors for utilization in a multiplier. These designs rely on different features of compression, such that imprecision in computation (as measured by the error rate and the so-called normalized error distance) can meet with respect to circuit-based figures of merit of a design (number of transistors, delay and power consumption). Four different schemes for utilizing the proposed approximate compressors are proposed and analyzed for a Dadda multiplier. Extensive simulation results are provided and an application of the approximate multipliers to image processing is presented. The results show that the proposed designs accomplish significant reductions in power dissipation, delay and transistor count compared to an exact design; moreover, two of the proposed multiplier designs provide excellent capabilities for image multiplication with respect to average normalized error distance and peak signal-to-noise ratio

**Summary:** In this paper, high speed is achieved, but transistor count is more.

**Chang, Y.-J., Cheng, Y.-C., Lin, Y.-F., Liao, S.-C., Lai, C.-H., & Wu, T.-C.(2019). An Imprecise 4-2 Compressor Design Used in Image Processing Applications.IET Circuits, Devices & Systems.**

Approximate computing can be used in the error-tolerant applications since it can provide meaningful results with lower power consumption. In this study, the authors proposed a novel imprecise 4-2 compressor which is used in the multipliers of image processing applications. Besides the output values, they also consider the pattern distribution to resynthesise the 4-2 compressor in imprecise style. Compared to the precise compressor, the proposed imprecise 4-2 compressor can reduce power consumption and delay by 56 and 39%. Compared to the precise multiplier, the simulation results show that the multiplier which uses the proposed imprecise 4-2 compressor can achieve 33 and 30% improvement in power consumption and delay, respectively. In addition, the image quality of their design is good for human perception because peak signal-to-noise ratio PSNR values are more than 33 and mean structural similarity values are more than 0.99. Even compared to the related imprecise works, their design has a better error rate to improve the quality of images while maintains both the high power efficiency and low circuit complexity.

**Summary:** In this paper, high speed is achieved.

**Esposito, D., Strollo, A. G. M., Napoli, E., De Caro, D., & Petra, N. (2018). Approximate Multipliers Based on New Approximate Compressors. IEEE Transactions on Circuits and Systems I: Regular Papers.**

Approximate computing is an emerging trend in digital design that trades off the requirement of exact computation for improved speed and power performance. This paper proposes novel approximate compressors and an algorithm to exploit them for the design of efficient approximate multipliers. By using the proposed approach, we have synthesized approximate multipliers for several operand lengths using a 40-nm library. Comparison with previously presented approximated multipliers shows that the proposed circuits provide better power or speed for a target precision.

**Summary:** Applications to image filtering and to adaptive least mean squares filtering are presented along with reduced power and efficiency in the paper.

**Xu, Q., Mytkowicz, T., & Kim, N. S. (2016). Approximate Computing: A Survey. IEEE Design & Test, 33(1), 8–22.**

Many emerging applications do not require exact answers but rather acceptable ones. By relaxing the numerical equivalence between the specification and implementation of such applications, approximate computing promises significant energy-efficiency improvements and has gained significant traction over the past few years.

**Summary:** In this article, we survey state-of-the-art knowledge in this area, ranging from approximate circuit design, approximate architecture exploration to approximate software implementation.

**Karri Manikantta Reddy, M.H. Vasantha, Y.B. Nithin Kumar, Devesh Dwivedi,Design and analysis of multiplier using approximate 4-2 compressor, AEU - International Journal of Electronics and Communications.**

Approximate computing has received significant attention as an attractive paradigm for error-tolerant applications to reduce the power consumption, delay and area with some trade-off in accuracy. This paper proposes the design of a novel approximate 4–2 compressor. A modified architecture of Dadda Multiplier is presented for the effective utilization of the proposed compressor and to reduce the error at the output. Through extensive experimental evaluation, the efficiency of the proposed compressor and multiplier are evaluated in a 45 nm standard CMOS technology and their parameters are compared with those of the state-of-the-art approximate multipliers. The results show that the proposed compressor accomplish a significant reduction in error rate compared to other approximate compressors available in the literature. In addition, the proposed multiplier shows 35%, 36% and 17% reduction in power consumption, delay and area respectively compared to those of exact multiplier.

**Summary:** The effectiveness of multiplier is assessed by some of the image processing applications. On an average, the proposed multiplier processes images with 85% structural similarity compared to the exact output image.

**R. Kiruthika , S. Suguna, 2019, Low Latency and Power Efficient Aproximate Multipliers using Compressors, International Journal Of Engineering Research & Technology (IJERT) Volume 08, Issue 04 (April – 2019),**

Approximate computing has been considered to improve the accuracy-performance trade-off in error- tolerant applications. For many of these applications, multiplication is a key arithmetic operation. Given that approximate compressors are a key element in the design of power-efficient approximate multipliers, we first propose an initial approximate 4:2 compressor that introduces a rather large error to the output. According to the mean relative error distance (MRED), the most accurate of the proposed 16Ã—16 unsigned designs has a 44% smaller power-delay product (PDP) compared to other designs with comparable accuracy. The radix-4 signed Booth multiplier constructed using the proposed compressor achieves a 52% reduction in the PDP-MRED product compared to other approximate Booth multipliers with comparable accuracy. The proposed multipliers outperform other approximate designs in image sharpening and joint photographic experts group (JPEG) applications by achieving higher quality outputs with lower power consumptions.

**Summary:** In this approximate designs are implemented which can enhance the performance and can be implemented in image sharpening and JPEG applications.

**D. Esposito, A. G. M. Strollo, E. Napoli, D. De. Caro, and N. Petra. “Approximate Multipliers Based on New Approximate Compressors,” IEEE Trans. Circuits and Syst. Ⅰ: Reg. Papers, vol. 65, no. 12, pp. 4169- 4182, Dec. 2018.**

Approximate computing is an emerging trend in digital design that trades off the requirement of exact computation for improved speed and power performance. This paper proposes novel approximate compressors and an algorithm to exploit them for the design of efficient approximate multipliers. By using the proposed approach, we have synthesized approximate multipliers for several operand lengths using a 40-nm library. Comparison with previously presented approximated multipliers shows that the proposed circuits provide better power or speed for a target precision. Applications to image filtering and to adaptive least mean squares filtering are also presented in the paper.

The paper presents novel approximate compressors, that overcome previously proposed circuits in terms of error performance and circuit complexity. The approximate compressors are used to build approximate multipliers, using an algorithm that allocates the approximate compressors with the aim of optimize electrical performance while providing small error. For each operand size, we developed four approximate multiplier versions, with different precision vs electrical performance trade-off. The circuits developed in this paper and previously proposed approximated multipliers, have been synthesized by using a 40nm CMOS technology. Syntheses show that our circuits provide very good error-electrical performance trade-off, compared with previously proposed approximate multipliers.

We have also investigated the use of approximate multipliers in two applications: image filtering and LMS system identification. In both cases, proposed approximate multipliers show remarkable good results, compared with the state of the art. The proposed approach could, in principle, be applied also to Booth-encoded multipliers. In this case, however, each partial product is obtained by a Booth selector, rather than a simple AND gate. Thus, for each partial product, the probability of being high is different compared to a non-Booth multiplier. Additional investigation is required to design ad-hoc approximate compressors optimized for Booth- multipliers.

**Summary:** From the paper it shows that the tolerable approximation level strongly depends on the application, motivating the need of energy-efficient design-time (or run-time) quality configurable systems.

**CHAPTER 3**

**EXISTING METHOD**

A multiplier is one of the most important arithmetic units used in computational systems. A variety of arithmetic techniques have been presented to design an effective multiplier. One of the best methods is Wallace tree technique. This technique is capable to produce an effective hardware consisting of full-adder and half-adder circuits in three stages, which performs the multiplication operation in parallel. The Wallace-based multiplying operation includes three stages:

**Stage1.** Multiply (or better expressing, AND) each bit of multiplicand by each bit of multiplier, yielding n2 partial products.

**Stage2.** Reduce the number of partial products using the layers of a FA and a HA blocks.

**Stage3.** Adding two n-sets resulted from the previous stage to an n-bit adder. It should be noted that the second stage is carried out as follows. As long as there are three or more bits with the same value, add a following layer:

* Three bits of the same value enter into FA and as a result, two bits with different values are produced (one bit with the same value and one bit with a higher value).
* If two bits with the same value remain, put them into an HA.
* If there is just one bit, transfer it to the next layer.

An effective design of reversible 4\*4 parallel multiplier based on Wallace tree is proposed. The product of two 4-bit digits X= (x3… x1, x0) and Y= (y3… y1, y0) results in 16 partial products. After generating the partial products, all three partial products with the same value are classified and fed into FAs. If two partial products with the same value remain, one HA is used. Moreover, if only one partial product remains, it will be moved directly to the next layer. Finally, in the last stage, resulted two bits are added to a 4-bit carry ripple adder.The speed of the multiplier depends on how fast the generation of partial product is done and they are added. To speed up the generation of partial products, the number of partial product has to be minimized and also the addition can be speed up by using efficient adder.There are no. of techniques that to perform binary multiplication, low power multiplier using MACunit ,modified booth multiplier, and low power multiplier are some of approaches to have hardware implementation of binary multiplier which are suitable for VLSI implementation at CMOS level.Array multiplier is a traditional method for multiplication.A multiplier is one of the key hardware blocks in most digital signal processing (DSP) systems. Typical DSP applications where a multiplier plays an important role include digital filtering, digital communications and spectral analysis .Many current DSP applications are targeted at portable, battery-operated systems, so that power dissipation becomes one of the primary design constraints. Since multipliers are rather complex circuits and must typically operate at a high system clock rate, reducing the delay of a multiplier is an essential part of satisfying the overall design.

The half adder is used to add only two numbers. To overcome this problem, the full adder was developed. The full adder is used to add three 1-bit binary numbers A, B, and carry C.The hardware requirement in terms of full adder (FA) and the length of final adder (FAL) for different size of array multipliers is obtained in the manner given in below Figure.

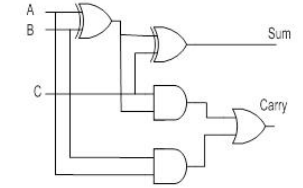
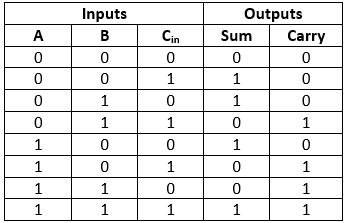


Fig : Full adder

Full Adder is the adder which adds three inputs and produces two outputs. The first two inputs are A and B and the third input is an input carry as C. The output carry is designated as CARRY and the normal output is designated as SUM.



In the above table,

* 'A' and' B' are the input variables. These variables represent the two significant bits which are going to be added.
* 'Cin' is the third input which represents the carry. From the previous lower significant position, the carry bit is fetched.
* The 'Sum' and 'Carry' are the output variables that define the output values.
* The eight rows under the input variable designate all possible combinations of 0 and 1 that can occur in these variables.

Array multiplier is an efficient layout of a combinational multiplier. Multiplication of two binary number can be obtained with one micro-operation by using a combinational circuit that forms the product bit all at once thus making it a fast way of multiplying two numbers since only delay is the time for the signals to propagate through the gates that forms the multiplication array. In array multiplier, consider two binary numbers A and B, of m and n bits. There are an summands that are produced in parallel by a set of an AND gates. n x n multiplier requires n (n-2) full adders, n half-adders and n2 AND gates. Also, in array multiplier worst case delay would be (2n+1) td.

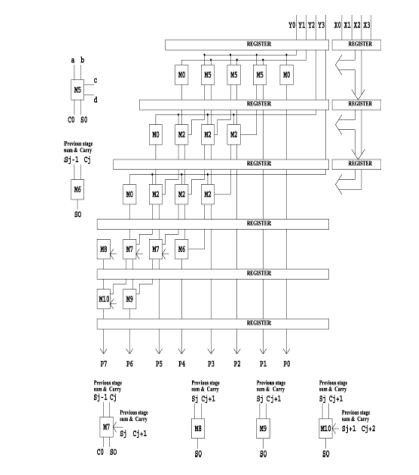


Fig : Array Multiplier

Binary multiplication of positive operands can be implemented in a combinational two dimensional logic array. A 4X4 array multiplier is shown in figure above. The functions of M0, M1, M2, and M4 are also shown in figure 1. X3X2X1X0 is the 4 bit multiplicand and Y3Y2Y1Y0 is the 4 bit multiplier. The main component in each cell is a full adder. The AND gate in each cell determines whether a multiplicand bit, Xj is added to the incoming partial product bit based on the value of the multiplier bit Yi. Each row adds the multiplicand (appropriately shifted) to the incoming partial product, PPi to generate the outgoing partial product PP(i+1), if Yi =1. If Yi=0, PPi is passed vertically downward unchanged. The worst case signal propagation delay path is from the upper right corner of the array to the high order product bit output at the bottom left corner of the array. Assuming that there are two gate delays from the inputs to the outputs of a full adder block. Since the array multiplier is having low speed and consumes more power many low power multiplier design techniques are available now.

The design techniques such as Glitching power minimization by selective gate freezing, Fast power efficient circuit lock switch-off schemes, power-aware scalable pipelined Booth multiplier, partially guarded computation (PGC) , High performance low power left to right array multiplier design are existing works that reduce the dynamic power consumption by minimizing the switched capacitance. It can only achieve savings of low amount in total power dissipation since it operates in the layout environment which is tightly restricted. The Fast power efficient circuit block switch-off scheme proposes a double switch circuit block switch scheme capable of reducing power dissipation during down time by shortening the settling time after reactivation. The drawbacks of this scheme are the necessity for two independent virtual rails and the necessity for two additional transistors for switching each cell. A power aware scalable pipelined Booth multiplier design presents a multiplier using Dynamic Range Detection unit. It is used to select the input operand with a smaller effective dynamic range to yield the Booth codes. There are three separate Wallace trees for the 4X4, 8X8, and 16X16 multiplications.

Wallace tree is an efficient hardware implementation of a digital circuit that multiplies two integers, devised by Australian Computer Scientist Chris Wallace in 1964. The Wallace tree has three steps:

* In one of the arguments each bit is multiplied by each bit of the other, yielding results. The wires carry different weights depending on position of the multiplied bits,
* The number of partial products reduced to two by layers of full and half adders.
* The wires in two numbers gets grouped, and added with a conventional adder.

In Wallace method the multiplication of two numbers is done by reducing the partial product matrix into a two row matrix by a half adder, full adder, carry save adder & these two rows are added utilizing a fast carry propagate adder to produce the output product.in this Wallace tree method we used half adder for summation of 2 bit and used full adder for summation of 3 bit. For multiplicands of higher than 8-bits this advantage is more beneficial, because the addition of partial products is low in Wallace tree and hence increases speed. Here each bit of each partial product in every column is added together by a set of counters used in parallel so that no carry is propagated further. Then this matrix is reduced by another set of counters until a two row matrix generates to get the final product result. Basic Building Blocks of Multiplier

* Formation of partial product using AND gate logic.
* Reducing the n number of partial products to a two-row partial products by compressing the columns with [3,2] & [2,2] adders.
* Merging two-rowed partial products with carry propagation Adders.
* 2bit result.

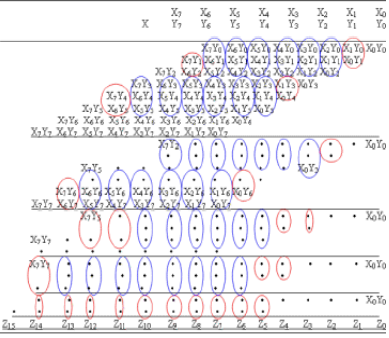


Fig : Wallace multiplier

The Wallace tree module adds with the 4 partial products and generates two intermediate operands for final addition.This will certainly induce area and capacitance penalties. Partially guarded computation (PGC) divides the arithmetic units, e.g., adders, and multipliers, into two parts, and turns off the unused part to minimize the power consumption. A circuit is reversible if the number of inputs and outputs are equal and there is a one-to-one correspondence between its inputs and outputs, i.e. not only the outputs can be uniquely obtained from the inputs, but also the inputs can be recovered from the outputs.

The reversible circuits are evaluated with different criteria such as gates count, the number of constant inputs, the number of garbage outputs, QC, delay and hardware complexity. The total number of reversible gates required to realize a reversible circuit is its gates count. Constant inputs are the input lines that are either set to 0 or 1 in the circuit’s input side. The outputs that would not be utilized for the further computations are called garbage outputs. The linear quantum cost of a quantum circuit is defined as the sum of primitives quantum gates needed to realize a reversible circuit. The delay is considered as the maximum number of gates in critical path from the inputs to the outputs.

**Disadvantages:**

* Power consumption is more.
* Area efficiency is less.
* Delay is more

**CHAPTER 4**

**PROPOSED METHOD**

In this project we are proposing an approximate with a dynamic truncation of partial products and then an error compensation circuit are proposed. This reconfigurable truncation helps us for dynamic input truncation that is used to construct the adjustable multiplier. Here is the difference between the existing accurate Wallace tree implementation and the proposed approximate dynamically adjustable multiplier process.

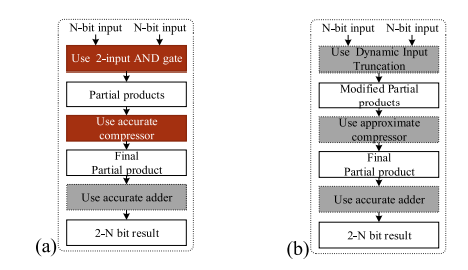


Fig.4.1 illustration of approximate multiplier and exact Wallace multiplier flow

Figure 4.1(a) shows the overall flow of the traditional flow for multiplication that generates accurate results. First, accurate partial products are produced using 2-input AND gates, and later compressed by the accurate compressors. Finally, accurate adders sum the compressed partial products to generate the result. Figure 4.1(b) shows our proposed flow for the proposed approximate multipliers. The differences between traditional multiplication and the proposed multiplication are the steps of generating partial products and compressing the partial products. In the step of generating partial products, we use the dynamic input truncation to generate the modified partial products.

Here, a high-accuracy and low-power approximate 4-2 compressor is proposed. The proposed 4-2 approximate compressor is shown in figure 4.2. The design of the proposed 4-2 approximate compressor is described as follows. Four inputs X1 ∼ X4 are used to generate W1 ∼ W4 using Eqs. (10)-(13). Because an incorrectly computed carry bit has a higher error distance than the sum bit, i.e., an incorrect carry bit produces two times ED of that produced by an incorrect sum bit, the carry bit in the proposed compressor is designed always to be correctly generated. The equations of generating carry bit are shown in (14)-(16). The carry bit will become 1 under three circumstances. One is X1 and X2 are both 1.

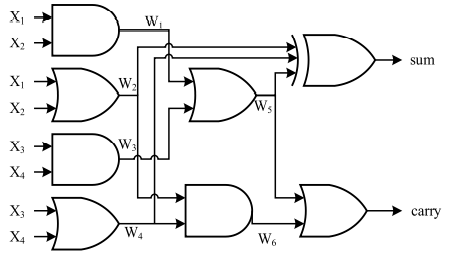
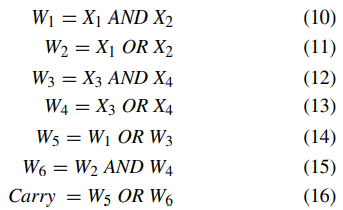


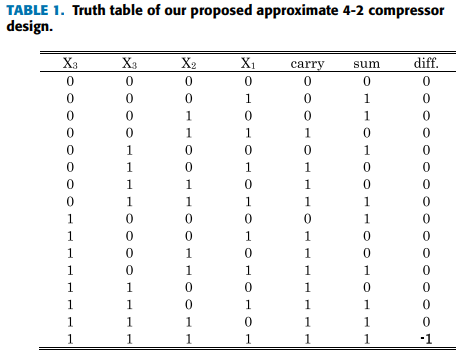
Fig.4.2 Gate-level implementation of proposed 4-2 compressor

Another is X3 and X4 are both 1. The third is either of X1 or X2 is 1 and either of X3 or X4 is 1. (14) Checks the first two situations, and (15) checks third situation. (16) Produces the final carry bit.



The proposed equation to generate the sum bit is shown in (17). In an accurate 4-2 compressor, the sum bit is generated with four XOR gates built within the two full adders. Whereas in our proposed compressor, we generate the sum bit by inputting W2 and W4 into a 2-input XOR gate to utilize the signals that are used to generate the carry bit. By sharing the common signals, we can reduce the circuit area and static power consumption. However, we found that the error distance is large if only W2 and W4 are fed into a 2-input XOR gate. Because W2 and W4 are generated with OR gates, the error occurs either when both X1 and X2 are 1 or both X3 and X4 are 1, which lead the sum bit to the result of 1 when it is supposedly 0. To achieve high accuracy, we add W5, the signal used to detect these two cases, into the XOR gate. For example, if both X1 and X2 are 1, both W2 and W5 will be 1, and the sum bit will turn out to be ‘0 XOR W4’, resulting in W4 as the sum bit. In this case, the number of bits that need to be considered are only X3 and X4. However, when all four inputs are 1, the sum bit turns out to be 1, resulting in the error distance of 1.





For the error detection purpose, we only need an extra AND gate to detect whether both W1 and W3 are 1, because W1 uses an AND gate to detect whether both X1 and X2 are 1, and W3 uses an AND gate to detect whether both X3 and X4 are 1. The equation of the error detection circuit (EDC) is shown in (18). Therefore, the error compensation circuit of the proposed 4-2 compressor can be easily constructed by adding an extra AND gate.



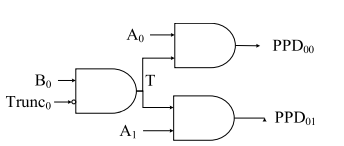
To achieve an adjustable approximate multiplier at runtime, we propose a dynamic input truncation technique, which uses two 2-input AND gates, as shown in figure 4.3, to produce a partial product, whose equation is shown in (19), where A is the multiplicand and B is the multiplier. The Trunc signal is used to determine whether the partial product PPD should be truncated. If the Trunc is 1, the partial product is truncated to 0. To be more precise, the Trunc signals save the power by truncating the PPDs in the multiplications to zeros. In other words, we can think of the Trunc signals serving the role as disabling the hardware units in the corresponding columns.





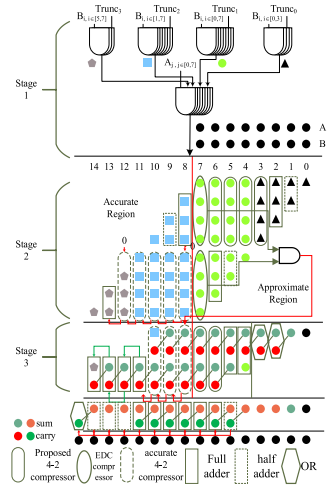
**Fig.4.3. Modified partial product**

For an 8 × 8 multiplier, each bit of the multiplier is corresponded to 8 bits of the multiplicand; therefore, we propose to reduce hardware costs by sharing gates with an extra AND gate. For example, PPD00 equals ∼Trunc0·B0·A0 and PPD01 equals ∼Trunc0·B0·A1. In this case, ∼Trunc0·B0 can be computed in advance to generate a mask, and three 2-input AND gates are required, as shown in figure 4.4. The control of the Trunc signals in the proposed approximate multiplier makes the approximate multiplier dynamically adjustable.



**Fig.4.4 A gate sharing example to reduce the number of gates.**

Figure 4.5 shows an approximate multiplier with the proposed technique. Even though the input width of the multiplier is designed to be 8-bit, the proposed technique can still be extended to larger multipliers. The proposed approximate multiplier contains three stages. In the first stage, each partial product is generated by two 2-input AND gates as shown previously in figure 4.3 with the gate sharing technique applied in figure 4.4 to further reduce hardware costs. Depending on the requirements, the accuracy of the generated partial product can be determined based on the Trunc signal. In our proposed approximate multiplier, to make the control more efficient as well as to reduce the hardware costs, we design a 4-bit Trunc signal with each bit to control more than one partial product column, which we call ‘‘3-4-4-4 partition’’, specifically, each bit from MSB to LSB to control column 14th∼12th, 11th ∼8th, 7th∼4th and 3rd ∼0th respectively, corresponding to the color of khaki, sky blue, green and black in Stage 2 in FIGURE 7. For example, if the Trunc(3−0) is 01012, column 14th ∼12th and 7th ∼4th are accurate, and column 11th ∼8th and 3rd ∼0th are truncated.



**Fig. 4.5 proposed approximate multiplier**

The second stage shows the steps of compressing the partial products. After the partial products are generated, they are divided into two regions: column 14th∼8th being the accurate region, and 7th∼0th being the approximate region. The split of accurate and approximate region is decided from the most intuitive half-half separation. If we do 30-70 split, for example, with too many computations done by the approximate multiplier, the accuracy loss will be significant. On the other hand, if we do 70-30 split, the effect of the approximate computing for power reduction will be little. Because the weight of the partial products in the accurate region is higher and more important, we compress the partial products in that region with accurate 4-2 compressors. On the other hand, we use our proposed approximate 4-2 compressors and error compensation circuit to compress the partial products in the approximate region. In the third stage, we use OR gates in columns 3rd ∼0th to generate results and ignore carry propagation considering them close to LSB, whose errors have less effect on the final results. We detect errors in the second stage with the EDC, i.e., a single AND gate, to determine whether the compensation bit should be produced. We use the proposed approximate 4-2 compressors, accurate 4-2 compressors, full adders, and half adders to compress the partial products in the remaining columns. After finishing the third stage, we get the final two partial product rows, which is summed up by using accurate adders to produce the final results.

**CHAPTER 5**

**ADVANTAGES & APPLICATIONS**

**ADVANTAGES**

* Power consumption is less.
* Area efficiency is high.
* Delay is less

**APPILICATIONS**

* FFT
* DIP
* DSP applications
* In MAC Unit
* Image smoothening

**CHAPTER 6**

**XILINX VIVADO AND VERILOG HDL**

**HISTORY OF VERILOG**

Verilog was started initially as a proprietary hardware modeling language by Gateway Design Automation Inc. around 1984. It is rumored that the original language was designed by taking features from the most popular HDL language of the time, called HiLo, as well as from traditional computer languages such as C. At that time, Verilog was not standardized and the language modified itself in almost all the revisions that came out within 1984 to 1990.

Verilog simulator was first used beginning in 1985 and was extended substantially through 1987. The implementation was the Verilog simulator sold by Gateway. The first major extension was Verilog-XL, which added a few features and implemented the infamous "XL algorithm" which was a very efficient method for doing gate-level simulation.

The time was late 1990. Cadence Design System, whose primary product at that time included thin film process simulator, decided to acquire Gateway Automation System. Along with other Gateway products, Cadence now became the owner of the Verilog language, and continued to market Verilog as both a language and a simulator.

At the same time, Synopsys was marketing the top-down design methodology, using Verilog. This was a powerful combination. In 1990, Cadence recognized that if Verilog remained a closed language, the pressures of standardization would eventually cause the industry to shift to VHDL. Consequently, Cadence organized the Open Verilog International (OVI), and in 1991 gave it the documentation for the Verilog Hardware Description Language. This was the event which "opened" the language.

**INTRODUCTION**

* HDL is an abbreviation of Hardware Description Language. Any digital system can be represented in a REGISTER TRANSFER LEVEL (RTL) and HDLs are used to describe this RTL.
* Verilog is one such HDL and it is a general-purpose language –easy to learn and use. Its syntax is similar to C.
* The idea is to specify how the data flows between registers and how the design processes the data.
* To define RTL, hierarchical design concepts play a very significant role. Hierarchical design methodology facilitates the digital design flow with several levels of abstraction.
* Verilog HDL can utilize these levels of abstraction to produce a simplified and efficient representation of the RTL description of any digital design.
* For example, an HDL might describe the layout of the wires, resistors and transistors on an Integrated Circuit (IC) chip, i.e., the switch level or, it may describe the design at a more micro level in terms of logical gates and flip flops in a digital system, i.e., the gate level. Verilog supports all of these levels.

**DESIGN STYLES:**

Any hardware description language like Verilog can be design in two ways one is bottom-up design and other one is top-down design.

**Bottom-Up Design:**

The traditional method of electronic design is bottom-up (designing from transistors and moving to a higher level of gates and, finally, the system). But with the increase in design complexity traditional bottom-up designs have to give way to new structural, hierarchical design methods.

**Top-Down Design:**

For HDL representation it is convenient and efficient to adapt this design-style. A real top-down design allows early testing, fabrication technology independence, a structured system design and offers many other advantages. But it is very difficult to follow a pure top-down design. Due to this fact most designs are mix of both the methods, implementing some key elements of both design styles.

**Features of Verilog HDL**

* Verilog is case sensitive.
* Ability to mix different levels of abstract freely.
* One language for all aspects of design, testing, and verification.
* In Verilog, Keywords are defined in lower case.
* In Verilog, Most of the syntax is adopted from "C" language.
* Verilog can be used to model a digital circuit at Algorithm, RTL, Gate and Switch level.
* There is no concept of package in Verilog.
* It also supports advanced simulation features like TEXTIO, PLI, and UDPs.

**VLSI DESIGN FLOW**

The VLSI design cycle starts with a formal specification of a VLSI chip, follows a series of steps, and eventually produces a packaged chip.



**System Specification:**

The first step of any design process is to lay down the specifications of the system. System specification is a high level representation of the system. The factors to be considered in this process include: performance, functionality, and physical dimensions like size of the chip.

The specification of a system is a compromise between market requirements, technology and economical viability. The end results are specifications for the size, speed, power, and functionality of the VLSI system.

**Architectural Design**

The basic architecture of the system is designed in this step. This includes, such decisions as RISC (Reduced Instruction Set Computer) versus CISC (Complex Instruction Set Computer), number of ALUs, Floating Point units, number and structure of pipelines, and size of caches among others. The outcome of architectural design is a Micro-Architectural Specification (MAS).

**Behavioral or Functional Design:**

In this step, main functional units of the system are identified. This also identifies the interconnect requirements between the units. The area, power, and other parameters of each unit are estimated.

Modules. The key idea is to specify behavior, in terms of input, output and timing of each unit, without specifying its internal structure.

The outcome of functional design is usually a timing diagram or other relationships between units.

**Logic Design:**

In this step the control flow, word widths, register allocation, arithmetic operations, and logic operations of the design that represent the functional design are derived and tested.

This description is called Register Transfer Level (RTL) description. RTL is expressed in a Hardware Description Language (HDL), such as VHDL or Verilog. This description can be used in simulation and verification

**Circuit Design:**

The purpose of circuit design is to develop a circuit representation based on the logic design. The Boolean expressions are converted into a circuit representation by taking into consideration the speed and power requirements of the original design. Circuit Simulation is used to verify the correctness and timing of each component

The circuit design is usually expressed in a detailed circuit diagram. This diagram shows the circuit elements (cells, macros, gates, transistors) and interconnection between these elements. This representation is also called a netlist. And each stage verification of logic is done.

**Physical design:**

In this step the circuit representation (or netlist) is converted into a geometric representation. As stated earlier, this geometric representation of a circuit is called a layout.

Layout is created by converting each logic component (cells, macros, gates, transistors) into a geometric representation (specific shapes in multiple layers), which perform the intended logic function of the corresponding component. Connections between different components are also expressed as geometric patterns typically lines in multiple layers.

**Layout verification:**

Physical design can be completely or partially automated and layout can be generated directly from netlist by Layout Synthesis tools. Layout synthesis tools, while fast, do have an area and performance penalty, which limit their use to some designs. These are verified.

**Fabrication and Testing:**

Silicon crystals are grown and sliced to produce wafers. The wafer is fabricated and diced into individual chips in a fabrication facility. Each chip is then packaged and tested to ensure that it meets all the design specifications and that it functions properly.

**MODULE:**

A module is the basic building block in Verilog. It can be an element or a collection of low level design blocks. Typically, elements are grouped into modules to provide common functionality used in places of the design through its port interfaces, but hides the internal implementation.

**Syntax:**

module<module name> (<module\_port\_list>);

…..

<module internals> //contents of the module

….

Endmodule

**Instances**

A module provides a template from where one can create objects. When a module is invoked Verilog creates a unique object from the template, each having its own name, variables, parameters and I/O interfaces. These are known as *instances*.

**Ports:**

* Ports allow communication between a module and its environment.
* All but the top-level modules in a hierarchy have ports.
* Ports can be associated by order or by name.

You declare ports to be input, output or inout. The port declaration syntax is:

*Input* [range\_val:range\_var] list\_of\_identifiers;

*output*[range\_val:range\_var] list\_of\_identifiers;

*inout*[range\_val:range\_var] list\_of\_identifiers;

**Identifiers**

* Identifiers are user-defined words for variables, function names, module names, and instance names. Identifiers can be composed of letters, digits, and the underscore character.
* The first character of an identifier cannot be a number. Identifiers can be any length.
* Identifiers are case-sensitive, and all characters are significant.

An identifier that contains special characters, begins with numbers, or has the same name as a keyword can be specified as an escaped identifier*.* An escaped identifier starts with the backslash character(\) followed by a sequence of characters, followed by white space.

**Keywords:**

* Verilog uses keywords to interpret an input file.
* You cannot use these words as user variable names unless you use an escaped identifier.
* Keywords are reserved identifiers, which are used to define language constructs.
* Some of the keywords are always, case, assign, begin, case, end and end case etc.

**Data Types:**

Verilog Language has two primary data types:

* ***Nets*** - represents structural connections between components.
* ***Registers*** - represent variables used to store data.

Every signal has a data type associated with it. Data types are:

* ***Explicitly declared*** with a declaration in the Verilog code.
* ***Implicitly declared*** with no declaration but used to connect structural building blocks in the code. Implicit declarations are always net type "wire" and only one bit wide.

**Register Data Types**

* Registers store the last value assigned to them until another assignment statement changes their value.
* Registers represent data storage constructs.
* Register arrays are called memories.
* Register data types are used as variables in procedural blocks.
* A register data type is required if a signal is assigned a value within a procedural block
* Procedural blocks begin with keyword initial and always.

The data types that are used in register are register, integer, time and real.

**MODELING CONCEPTS:**

**Abstraction Levels:**

* Behavioral level
* Register-Transfer Level
* Gate Level
* Switch level

**Behavioral or algorithmic Level**

* This level describes a system by concurrent algorithms (Behavioral).
* Each algorithm itself is sequential meaning that it consists of a set of instructions that are executed one after the other.
* The blocks used in this level are ‘initial’, ‘always’ ,‘functions’ and ‘tasks’ blocks
* The intricacies of the system are not elaborated at this stage and only the functional description of the individual blocks is prescribed.
* In this way the whole logic synthesis gets highly simplified and at the same time more efficient.

**Register-Transfer Level:**

* Designs using the Register-Transfer Level specify the characteristics of a circuit by operations and the transfer of data between the registers.
* An explicit clock is used. RTL design contains exact timing possibility, operations are scheduled to occur at certain times.
* Modern definition of a RTL code is "Any code that is synthesizable is called RTL code".

**Gate Level:**

* Within the logic level the characteristics of a system are described by logical links and their timing properties.
* All signals are discrete signals. They can only have definite logical values (`0', `1', `X', `Z`). The usable operations are predefined logic primitives (AND, OR, NOT etc gates).
* It must be indicated here that using the gate level modeling may not be a good idea in logic design.
* Gate level code is generated by tools like synthesis tools in the form of netlists which are used for gate level simulation and for backend.

**OPERATORS**

Verilog provided many different operators types. Operators can be,

* Arithmetic Operators
* Relational Operators
* Bit-wise Operators
* Logical Operators
* Reduction Operators
* Shift Operators
* Concatenation Operator
* Conditional Operator

**Arithmetic Operators**

* These perform arithmetic operations. The + and - can be used as either unary (-z) or binary (x-y) operators.
* Binary: +, -, \*, /, % (the modulus operator)
* Unary: +, - (This is used to specify the sign)
* Integer division truncates any fractional part
* The result of a modulus operation takes the sign of the first operand
* If any operand bit value is the unknown value x, then the entire result value is x
* Register data types are used as unsigned values (Negative numbers are stored in two's complement form).

**Relational Operators**

Relational operators compare two operands and return a single bit 1or 0. These operators synthesize into comparators. Wire and reg variables are positive Thus (-3’b001) = = 3’b111 and (-3d001)>3d1 10, however for integers -1<>



* The result is a scalar value
* 0 if the relation is false (a is bigger than b)
* 1 if the relation is true ( a is smaller than b)
* x if any of the operands has unknown x bits (if a or b contains X)

**Note:**If any operand is x or z, then the result of that test is treated as false (0)

**Bit-wise Operators**

Bitwise operators perform a bit wise operation on two operands. This take each bit in one operand and perform the operation with the corresponding bit in the other operand. If one operand is shorter than the other, it will be extended on the left side with zeroes to match the length of the longer operand



Computations include unknown bits, in the following way:

-> ~x = x

-> 0&x = 0

-> 1&x = x&x = x

-> 1|x = 1

-> 0|x = x|x = x

-> 0^x = 1^x = x^x = x

-> 0^~x = 1^~x = x^~x = x

When operands are of unequal bit length, the shorter operand is zero-filled in the most significant bit positions.

**Logical Operators**

Logical operators return a single bit 1 or 0. They are the same as bit-wise operators only for single bit operands. They can work on expressions, integers or groups of bits, and treat all values that are nonzero as “1”. Logical operators are typically used in conditional (if ... else) statements since they work with expressions.



Expressions connected by && and || are evaluated from left to right

Evaluation stops as soon as the result is known

The result is a scalar value:

* 0 if the relation is false
* 1 if the relation is true
* x if any of the operands has x (unknown) bits

**Reduction Operators**

Reduction operators operate on all the bits of an operand vector and return a single-bit value. These are the unary (one argument) form of the bit-wise operators.



* Reduction operators are unary.
* They perform a bit-wise operation on a single operand to produce a single bit result.
* Reduction unary NAND and NOR operators operate as AND and OR respectively, but with their outputs negated.

**Shift Operators**

Shift operators shift the first operand by the number of bits specified by the second operand. Vacated positions are filled with zeros for both left and right shifts (There is no sign extension).



* The left operand is shifted by the number of bit positions given by the right operand.
* The vacated bit positions are filled with zeroes

**Concatenation Operator**

* The concatenation operator combines two or more operands to form a larger vector.
* Concatenations are expressed using the brace characters { and }, with commas separating the expressions within.
* ->Example: + {a, b[3:0], c, 4'b1001} // if a and c are 8-bit numbers, the results has 24 bits
* Unsized constant numbers are not allowed in concatenations

**Operator Precedence**



**Switch Level:**

This is the lowest level of abstraction. A module can be implemented in terms of switches, storage nodes and interconnection between them. However, as has been mentioned earlier, one can mix and match all the levels of abstraction in a design. RTL is frequently used for Verilog description that is a combination of behavioral and dataflow while being acceptable for synthesis.

**Xilinx Verilog HDL Tutorial**

**Xilinx Verilog HDL Tutorial**

**Getting started**

Frist we need to download and install Xilinx and ModelSim. These tools both have free student versions. Please accomplish Appendix B, C, and D in that order before continuing with this tutorial. Additionally if you wish to purchase your own Spartan3 board, you can do so at Digilent’s Website. Digilent offers academic pricing. Please note that you must download and install Digilent Adept software. The software contains the drivers for the board that you need and also provides the interface to program the board.

**Introduction**

Xilinx Tools is a suite of software tools used for the design of digital circuits implemented using Xilinx Field Programmable Gate Array (FPGA) or Complex Programmable Logic Device (CPLD). The design procedure consists of (a) design entry, (b) synthesis and implementation of the design, (c) functional simulation and (d) testing and verification. Digital designs can be entered in various ways using the above CAD tools: using a schematic entry tool, using a hardware description language (HDL) – Verilog or VHDL or a combination of both. In this lab we will only use the design flow that involves the use of Verilog HDL.

The CAD tools enable you to design combinational and sequential circuits starting with Verilog HDL design specifications. The steps of this design procedure are listed below:

1. Create Verilog design input file(s) using template driven editor.

2. Compile and implement the Verilog design file(s).

3. Create the test-vectors and simulate the design (functional simulation) without using a PLD (FPGA or CPLD).

4. Assign input/output pins to implement the design on a target device.

5. Download bitstream to an FPGA or CPLD device.

6. Test design on FPGA/CPLD device

A Verilog input file in the Xilinx software environment consists of the following segments:

***Header:*** module name, list of input and output ports.

***Declarations:*** input and output ports, registers and wires.

***Logic Descriptions:*** equations, state machines and logic functions.

***End:*** endmodule

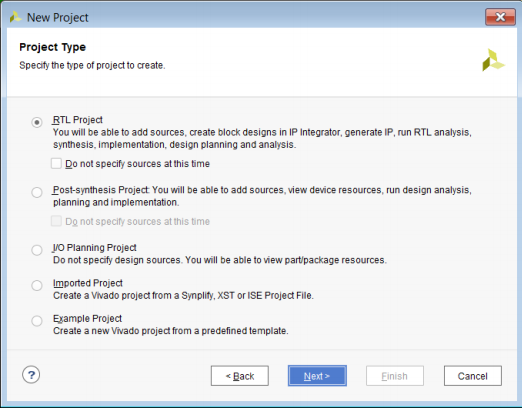
All your designs for this lab must be specified in the above Verilog input format. Note that the *state diagram* segment does not exist for combinational logic designs.

**Programmable Logic Device: FPGA**

In this lab digital designs will be implemented in the Basys2 board which has a Xilinx Spartan3E –XC3S250E FPGA with CP132 package. This FPGA part belongs to the Spartan family of FPGAs. These devices come in a variety of packages. We will be using devices that are packaged in 132 pin package with the following part number: XC3S250E-CP132.

**Creating a New Project**

Creating Projects You can use the New Project wizard to easily create different types of projects in the Vivado IDE. To open the New Project wizard, select File > New Project. This wizard enables you to specify a project location and name and create the types of projects shown in below figure



**New Project Wizard—Project Type Page**

**Project Name**: Write the name of your new project which is user defined.

**Project Location**: The directory where you want to store the new project in the specified location in one of your drive. In above window they are stored in location c drive which is not correct, the location of software and code should not be same location and Clicking on NEXT.

For each of the properties given below, click on the ‘**value**’ area and select from the list of values that appear.

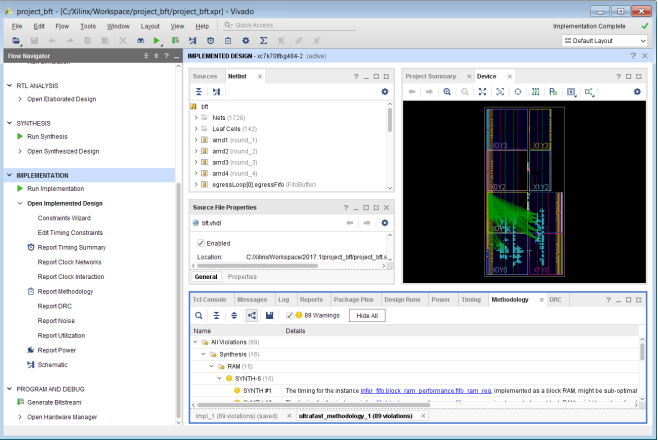
* **Device Family**: Family of the FPGA/CPLD used. In this laboratory we will be using the Spartan3E FPGA’s.
* **Device**: The number of the actual device. For this lab you may enter **XC3S250E** (this can be found on the attached prototyping board)
* **Package**: The type of package with the number of pins. The Spartan FPGA used in this lab is packaged in CP132 package.
* **Speed Grade**: The Speed grade is “-4”.
* **Synthesis Tool**: **XST** [VHDL/Verilog]
* **Simulator:** The tool used to simulate and verify the functionality of the design. Then click on **NEXT** to save the entries.

**Opening Designs:**

Use the Flow Navigator or Flow menu to select the following commands:

* Open Elaborated Design
* Open Synthesized Design
* Open Implemented Design

The Flow > Open Implemented Design command populates the Vivado IDE as shown in below figure.



**Implemented design**

All project files such as schematics, netlists, Verilog files, VHDL files, etc., will be stored in a subdirectory with the project name.

In order to open an existing project in Xilinx Tools, select **File->Open Project** to show the list of projects on the machine. Choose the project you want and click **OK**.

If creating a new source file, click on the NEW SOURCE.

Creating a Verilog HDL input file for a combinational logic design:

In this lab we will enter a design using a structural or RTL description using the Verilog HDL. You can create a Verilog HDL input file (**.v** file) using the HDL Editor available in the Xilinx Vivado Tools (or any text editor).

In the previous window, click on the NEW SOURCE

(Note: “**Add to project**” option is selected by default. If you do not select it then you will have to add the new source file to the project manually.)

Select **Verilog Module** and in the “File Name:” area, enter the name of the Verilog source file you are going to create. Also make sure that the option **Add to project** is selected so that the source need not be added to the project again. Then click on **Next** to accept the entries.

In the **Port Name** column, enter the names of all input and output pins and specify the **Direction** accordingly. A Vector/Bus can be defined by entering appropriate bit numbers in the **MSB/LSB** columns. Then click on **Next>** to get a window showing all the new source information above window. If any changes are to be made, just click on **<Back** to go back and make changes. If everything is acceptable, click on **Finish > Next > Next > Finish** to continue.

Once you click on **Finish**, the source file will be displayed in the sources window in the **Project Navigator.** If a source has to be removed, just right click on the source file in the **Sources in Project** window in the **Project Navigator** and select **remove** in that. Then select **Project -> Delete Implementation Data** from the Project Navigator menu bar to remove any related files.

**Editing the Verilog source file**

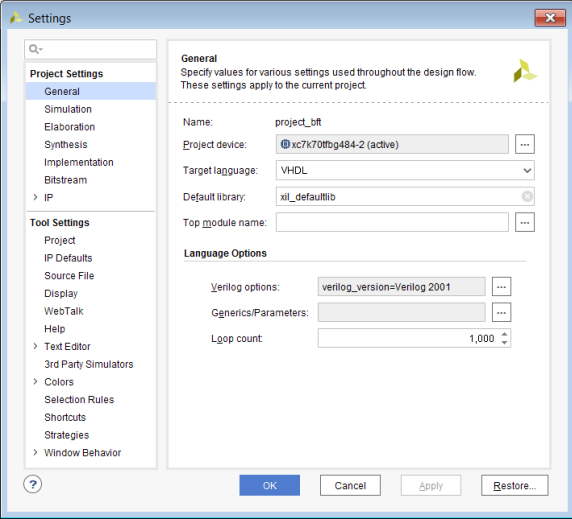
The source file will now be displayed in the **Project Navigator** window (Figure 8). The source file window can be used as a text editor to make any necessary changes to the source file. All the input/output pins will be displayed. Save your Verilog program periodically by selecting the **File->Save** from the menu. You can also edit Verilog programs in any text editor and add them to the project directory using “Add Copy Source”.

Here in the above window we will write the Verilog programming code for specified design and algorithm in the window.

After writing the programming code we will go for the synthesis report.

**Configuring Project Settings**

You can configure the Project Settings in the Settings dialog box to meet your design needs. These settings include general settings, related to the top module definition and language options, as well as simulation, elaboration, synthesis, implementation, bitstream, and IP settings.



**Settings Dialog Box—Project Settings General Category**

To open the Settings dialog box, use any of the following methods:

• In the Flow Navigator Project Manager section, click Settings.

• Select Tools > Settings.

• In the main toolbar, click the Settings toolbar button .

• In the Project Summary, click the Edit link next to Settings.

***Synthesis and Implementation of the Design:***

The design has to be synthesized and implemented before it can be checked for correctness, by running functional simulation or downloaded onto the prototyping board. With the top-level Verilog file opened (can be done by double-clicking that file) in the HDL editor window in the right half of the Project Navigator, and the view of the project being in the **Module view** , the **implement design** option can be seen in the **process view**. **Design entry utilities** and **Generate Programming File** options can also be seen in the process view.

To synthesize the design, double click on the **Synthesize Design** option in the **Processes window**.

To implement the design, double click the **Implement design** option in the **Processes window**. It will go through steps like **Translate, Map and Place & Route**. If any of these steps could not be done or done with errors, it will place a **X** mark in front of that, otherwise a tick mark will be placed after each of them to indicate the successful completion

After synthesis right click on synthesis and click view text report in order to generate the report of our design.

**XILINX VIVADO SIMULATION PROCEDURE:**

After completion of synthesis we will go simulation in order to verify the functionality of the implemented design.

Click on **Run Simulation** and set the module that is need to Run

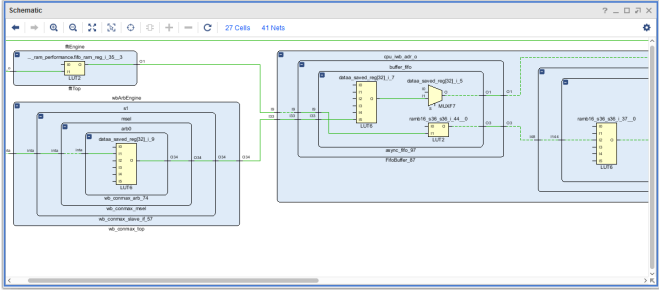
Next **double click on Run Behavioral Simulation** to check the errors. If no errors are found then double click on simulate behavioral model to get the output waveforms.

After clicking on **simulate behavioral model**, the simulation widow will appear pass the input values by making force constant and if it is clock by making force clock. Mention the simulation period and run for certain time and results will appear as shown in following window. Verify the results to the given input values.

**Using the Schematic Window:**

You can generate a Schematic window for any level of the logical or physical hierarchy. You can select a logic element in an open window, such as a primitive or net in the Netlist window, and use the Schematic command in the popup menu to create a Schematic window for the selected object.

An elaborated design always opens with a Schematic window of the top-level of the design, as shown in below figure.



**Schematic window**

**Using the Project Summary**

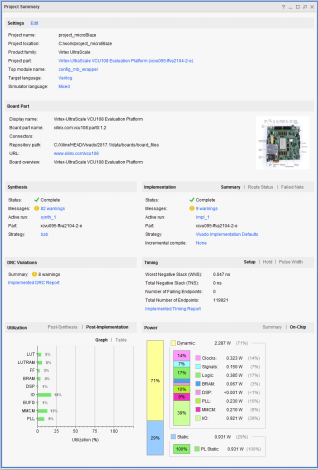
The Vivado IDE includes an interactive Project Summary (Figure 3-11) that updates dynamically as design commands are run and the design progresses through the design flow. It provides project and design information, such as the project part, board, and state of synthesis and implementation.

It also provides links to detailed information, such as links to the Messages and Reports windows as well as the Settings dialog box.

As synthesis and implementation complete, DRC violations, timing values, utilization percentages, and power estimates are also populated. To open the Project Summary, do either of the following:

• Select Window > Project Summary.

• Click the Project Summary toolbar button.

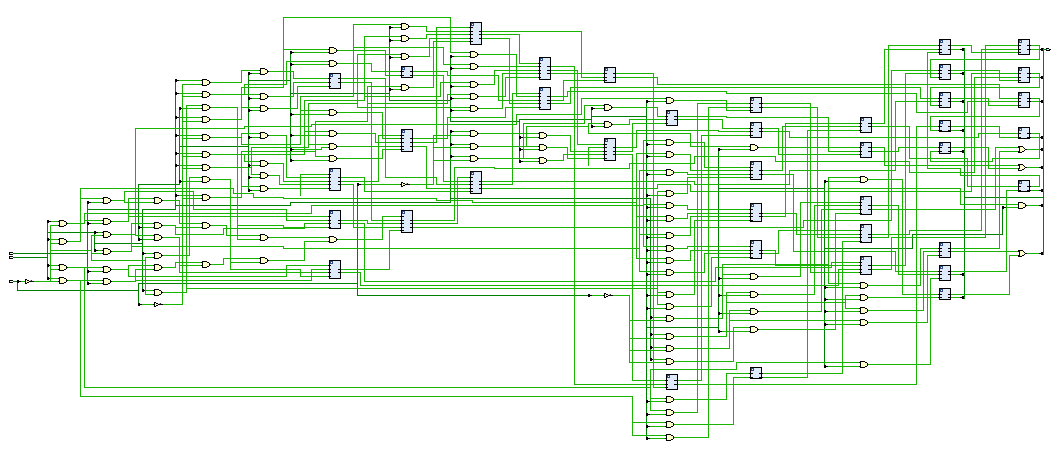


**Project Summary**

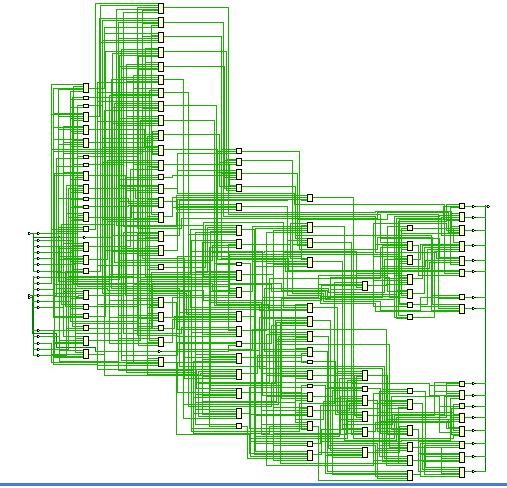
**CHAPTER 7**

**RESULTS**

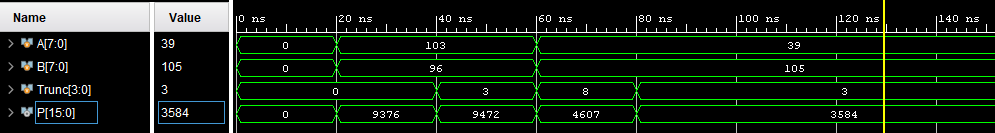
RTL schematic:



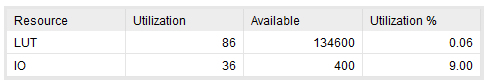
Technology schematic:



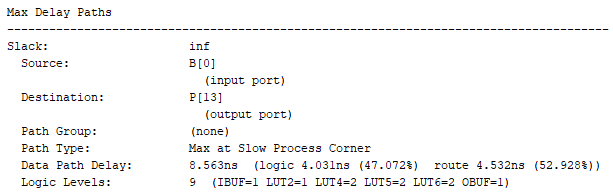
Simulation:



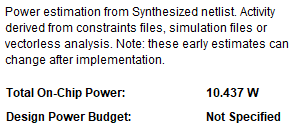
Area:



Delay:



Power:



**Evaluation table for Area, Delay:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Area (LUT’s)** | **Power (W)** | **Delay (ns)** |
| Proposed method | 86 | 8.563 | 10.437 |

**CHAPTER 8**

**CONCLUSION**

In this project, a high accuracy approximate 4-2 compressor that can be used to construct an approximate multiplier is proposed. The proposed approximate multiplier dynamically truncates partial products to adjust the accuracy and a simple error compensation circuit is used to reduce the error distance. The delay and the average power consumption of the proposed adjustable approximate multiplier is reduced, compared to the Wallace tree multiplier. Compared to other approximate multipliers, our proposed multiplier has the lowest mean error distance and lowest average power consumption

**FUTURE SCOPE**

In the multiplier structure the delay and power can be further optimized by simply applying the pipelining concept to the existing multiplier architecture. We can further enhance the design by using some other efficient multiplier architecture can also be a further modification that is possible.

**REFERENCES:**

[1] A. Wang, B. H. Calhoun, and A. P. Chandrakasan, Sub-Threshold Design for Ultra Low-Power Systems, vol. 95. New York, NY, USA: Springer, 2006.

[2] Q. Xu, T. Mytkowicz, and N. S. Kim, “Approximate Computing: A Survey,” IEEE Design & Test, vol. 33, no. 1, pp. 8-22, Feb. 2016.

[3] V. Leon, G. Zervakis, D. Soudris, and K. Pekmestzi, “Approximate Hybrid High Radix Encoding for Energy-Efficient Inexact Multipliers,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 26, no. 3, pp. 421– 430, Mar. 2018.

[4] C.–H. Chang, J. Gu, and M. Zhang, “Ultra low-voltage low-power CMOS 4-2 and 5-2 compressors for fast arithmetic circuits,” IEEE Trans. Circuits and Syst. Ⅰ: Reg. Papers, vol. 51, no. 10, pp. 1985-1997, Oct. 2004.

[5] A. Momeni, J. Han, P. Montuschi, and F. Lombardi. “Design and Analysis of Approximate Compressors for Multiplication,” IEEE Trans. Comput., vol. 64, no. 4, pp. 984-994, Apr. 2015.

[6] O. Akbari, M. Kamal, A. Afzali-Kusha, and M. Pedram. “Dual-Quality 4:2 Compressors for Utilizing in Dynamic Accuracy Configurable Multipliers,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol.25, no. 4, pp. 1352-1361, Apr. 2017.

[7] Z. Yang, J. Han, and F. Lombardi. “Approximate compressor for error resilient multiplier design,” IEEE Int. Symp. on Defect and Fault Tolerance in VLSI and Nano. Syst. (DFTS), Amherst, MA, 2015, pp. 183- 186.

[8] M. Ha and S. Lee. “Multipliers with Approximate 4-2 Compressors and Error Recovery Modules,” IEEE Embedded Systems Letters, vol. 10, no. 1, pp. 6-9, Mar. 2018.

[9] L. Qian, C. Wang, W. Liu, F. Lombardi, and J. Han, “Design and evaluation of an approximate Wallace-Booth multiplier,” 2016 IEEE Int. Symp. Circuits and Syst. (ISCAS), Montreal, QC, 2016, pp. 1974-1977.

[10] X. Yi, H. Pei, Z. Zhang, H.Zhou, and Y. He. “Design of an EnergyEfficient Approximate Compressor for Error-Resilient Multiplications,” 2019 IEEE Int. Symp. Circuits and Syst. (ISCAS). Sapporo, Japan, 2019, pp. 1-5.

[11] S. Venkatachalam and S.-B. Ko. “Design of Power and Area Efficient Approximate Multipliers,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 25, no. 5, pp. 1782-1786, May 2017.

[12] D. Esposito, A. G. M. Strollo, E. Napoli, D. De. Caro, and N. Petra. “Approximate Multipliers Based on New Approximate Compressors,” IEEE Trans. Circuits and Syst. Ⅰ: Reg. Papers, vol. 65, no. 12, pp. 4169- 4182, De