LOW ENERGY COMPUTER ARCHITECTURE DESIGNS

By

Mervat Mohamed Adel Mahmoud

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
DOCTOR OF PHILOSOPHY
in
Electronics and Communication Engineering

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT
2019

LOW ENERGY COMPUTER ARCHITECTURE DESIGNS

By

Mervat Mohamed Adel Mahmoud

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
DOCTOR OF PHILOSOPHY
in
Electronics and Communication Engineering

Under the Supervision of

Prof. Dr. Hossam A. H. Fahmy

Electronics and Communications Department
Faculty of Engineering, Cairo University, Cairo, Egypt

Dr. Dalia A. El-Dib

Electrical and Computer Engineering Department
Faculty of Engineering, Dalhousie University, Halifax, NS, Canada

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT
2019

Disclaimer

I hereby declare that this thesis is my own original work and that no part of it has been submitted for a degree qualification at any other university or institute.

I further declare that I have appropriately acknowledged all sources used and have cited them in the references section.

Dedication

I would like to dedicate my thesis to my beloved mother.

Acknowledgments

بسم الله الرحمن الرحيم

"سُبْحَانَكَ لَا عِلْمَ لَنَا إِلَّا مَا عَلَّمْتَنَا إِنَّكَ أَنتَ الْعَلِيمُ الْحَكِيمُ"

صدق الله العظيم

Foremost, all thanks are due to Almighty, the merciful God. God blessed me and gave me the strength to finish this study.

Dr. Hossam Fahmy and Dr. Dalia El-Dib, my thesis supervisors, have guided my work insightfully, and supported me during the thesis journey. Dr. Hossam (مُعَلِمـي) and I would like to write it in Arabic as he represents all the real meanings of this word and the verse of Ahmed Shawky’s poem:

قُمْ للمُعلّمِ وَفِّهِ التبجيلا ... كادَ المُعلّمُ أن يكونَ رسولا

أعلِمتَ أشرفَ أو أجلَّ من الذي ... يَبني ويُنشئُ أنفُساً وعقولا

Dr. Dalia has enriched my research with her experience and has always supported, guided and encouraged me to achieve higher in my entire career path - not only in my research.

My parents have supported me with their love and prayers. I would never succeed in life without their early support. Owing them my success, I will never be able to thank them enough. To them, I would like to express my sincere thanks.

My colleagues, Safaa Ahmed Elawamy, Dina Ellithy, and Rasha Mahmoud have provided me with cooperation and help with the design simulations and CAD tools. Also, I would like to thank my sister Marwa. And finally, I would like to thank my dear friends Asmaa Adawy, Rasha Shoitan, Abeer Farouk, Nahla El-azab, Ebtsam Arafa, Heba Draz, Ratshih, Nahla El-Sayed, and Hoda Hosny, who endured this long journey with me, always offering unconditional support and love.

Table of Contents

List of Tables

List of Figures

Abstract

The continuous increase in chip integration and the associated power consumption concerns with moving towards portability made low energy design one of the main challenges facing VLSI systems. As low energy design has low power with high operating frequency. Power/Energy management has various strategies at all design process levels. That includes energy optimization at technology, circuit, logic, architecture and system levels of abstraction. In this thesis, we present two low energy designs. A low energy design at circuit level of abstraction is proposed for combined binary/decimal multipliers. And a low energy design at architecture level of abstraction is proposed for main memory data compression.

As combined binary/decimal arithmetic is optimal in supporting binary and decimal high speed and low power applications. A low energy clock-gated pipelined dual base binary/decimal fixed-point multiplier is suggested extending a previously proposed non-pipelined design. A thorough study conducted on both the pipelined and non-pipelined designs versus other architectures in literature proves tremendous reductions in energy consumption. The pipeline stages are chosen to achieve energy reductions with acceptable latency. In addition, clock gating the pipelined multiplier design is introduced to provide a total of 43% energy reduction for the pipelined design if compared to the lowest energy design in the literature.

In addition, a new low energy lossless compression/decompression approach is suggested for the data of main memory. The proposed approach depends on the delta coding and the observation that, for many applications, the lines of the main memory pages are mostly similar. The target is to achieve a simple low energy compression design for exact storage of memory data. The proposed design lowers energy consumption by up to 66% when compared to previous designs. This is due to its simplicity and low latency. Furthermore, the frequency of operation is increased from 300 MHz to 800 MHz. The new design also allows the main memory to store up to 30% more data according to PARSEC and PERFECT benchmarks applications data.

Chapter 1 Introduction

In predicting the future of integrated electronics, Gordon Moore predicted in 1965 that the number of components per chip will double every year in the period till 1975 [1] reaching 65,000 components on a single quarter-inch semiconductor. In 1975, Moore reduced the rate to a doubling every two years due to integrating more microprocessors which are in general less dense in electronic circuits [2]. In 1995, Moore's stated that his projection is not going to stop soon [3]. In fact, Moore's rule was considered one of the driving forces of electronics industry. It challenged technologists to deliver annual breakthrough in manufacturing Integrated Circuits (ICs) to comply with Moore's law. In 2014, a die was able to hold over seven billion transistors. Moore's law worked perfectly and was continuously fulfilled and has caused many of the most important changes in the electronics manufacturing technology.

Since 1970s, The most dominant electronics manufacturing technologies used were bipolar and nMOS transistors [4]. Nevertheless, these consume non-negligible power even in static (non-switching) state. Consequently, by 1980s, the power consumption of bipolar designs and its cooling solution costs were considered too high to be sustainable. This caused an expected switch to a slower, but lower-power Complementary Metal Oxide Semiconductor (CMOS) technology. At that time, CMOS transistors consumed lower power largely because static (leakage) power was negligible if compared to dynamic (switching) power. Along with fulfilling Moore's law, the aim is always to increase processing power of electronic circuits. This is achieved by scaling down the technology, increasing the number of components per chip, and increasing the frequency of operation. In the late 2004 with scaling down the CMOS fabrication technology to 45-nm and downwards, we encountered a high increase in leakage power to the extent that it is comparable to dynamic power and can even dominate the overall power dissipation. Also, with integrating more and more components, the power increases dramatically, and causes a challenge regarding excessive thermal dissipation. Thus, another paradigm shift in computing electronics was inevitable. The shift to multi-core computing was in the aim to increase performance while keeping the hardware simple, retain acceptable power consumption and transfer complexity to higher levels of the system design abstraction, including software level. [4]. Approximate computing (AC) is one of the energy efficient computing paradigms that use the inaccuracy tolerance inherited in applications for significant performance improvements [5][6]. It leads to another tradeoff, energy and performance versus computing quality. Where, slightly losing computing quality can improve energy and/or density.

1.1. Power vs Energy

Both terms energy and power are ex-changeably used although energy is different from power. For example, a specific task needs a specified amount of energy E to complete over time T. Its power consumption P is the rate at which energy is consumed (E/T). The time needed to complete the task can be increased by reducing the frequency of operation for example. Whereas, the same amount of energy is still needed to complete the task. Thus, the power consumption is reduced; however the energy consumption (area under the graph) is still the same as shown in Figure 1.1.

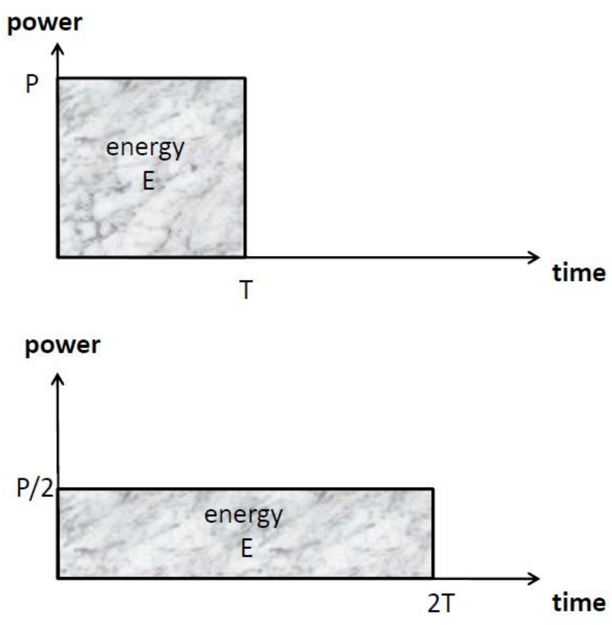


Figure 1.1 Energy versus power [7]

While energy measures the total quantity of work done, it doesn't say how fast the work done. As a loaded semi-trailer can be moved across the country with a lawnmower engine if time is not essential. If other things being equal, the tiny engine would do the same amount of energy and burn the same amount of fuel as the truck's big one. But the bigger engine has more power, so it can get the job done faster.

The question now, in computer architecture design, which is more important power or energy?. Desktop computers or wired digital devices are permanently connected to a power supply. Power supply feeds the design components with power ($P = V.I$), whereas these design's components consume this power. That makes energy efficiency here a bonus compared to a functional necessity. However, laptop computers or portable devices have limited battery life time ($E = P.T$), before get rid of the battery or recharge it.

1.2. Power/Energy Measurement in Digital Design

An IC's energy consumption is defined as its power consumption by the operating frequency ($E = P/f$) while power consumption is mainly composed of static power and dynamic power.

$$P_{total} = P_{dynamic} + P_{static} \tag{1.1}$$

Dynamic power consumption is frequency-dependent and results from one of the following three sources: Switching power, short circuit power and glitching power [11]. The dominant part of the dynamic power is the switching power which is consumed during the charging and discharging of capacitive nodes, Figure 1.2. It can be represented with the following equation;

$$P_{dynamic} = \alpha . C_l . V_{dd}^2 . f \tag{1.2}$$

Where α is the switching activity of the circuit; C_l is the effective capacitance of the circuit; V_{dd} is the supply voltage; and f is the operating frequency.

Short circuit power occurs during the momentary current flow that occurs when two complementary transistors conduct during a logic transition, which arises from long rise or fall times of input signals, Figure 1.3. Moreover, glitching power occasionally arises due to the finite delay of the logic gates that cause spurious transitions at different nodes in the circuit, Figure 1.4.

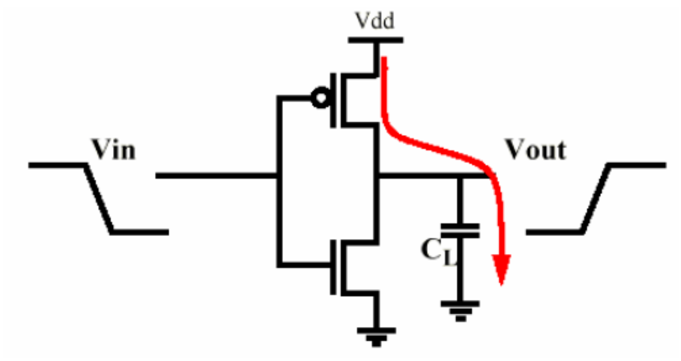


Figure 1.2 Switching power

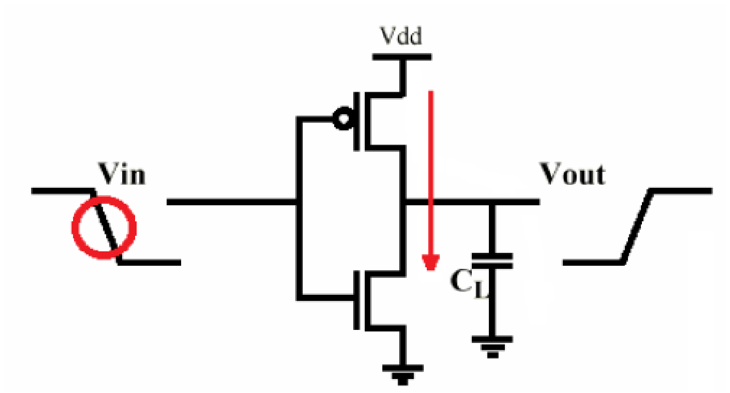


Figure 1.3 Short circuit power

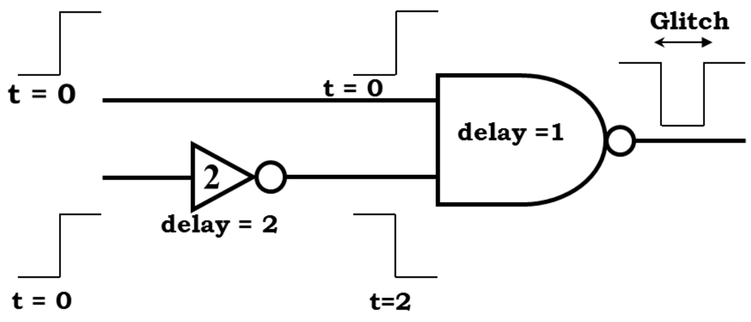


Figure 1.4 Example of how glitches occur

Static power typically comes from leakage current and dc current sources. Static power consumption has many components and has many paths, Figure 1.5. The most important contributor to static power in CMOS is the subthreshold leakage which is exponentially dependent on $(V_{gs} - V_T)$, where V_{gs} is the gate to source voltage and V_T is the threshold voltage. Another part of leakage is caused by reduced gate oxide thickness t_{ox} which increases gate oxide tunneling current. All parts of leakage current are increased excessively due to scaling down of technology which requires reducing V_T and t_{ox} to keep up with higher processing requirements.

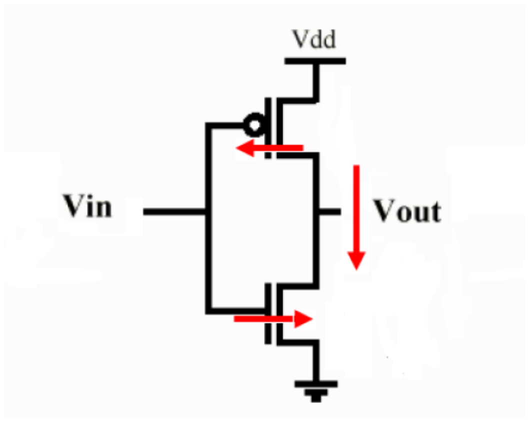


Figure 1.5 Leakage Power

1.3. Low Power/Energy Digital Design

Low power/energy design methodologies can be applied at all design levels of abstraction including system, algorithm, architecture, logic, circuit, device and technology levels. Low energy design can be defined as a design that has low power at high operating frequency. However, some of the low energy techniques reduce energy in exchange of reduced performance. Eventually, one has to reach a compromise between energy, power, performance, and cost to satisfy overall design requirements. Nevertheless, improvement at a higher level design abstraction will definitely affect all subsequent design abstraction levels to comply with the changes at that higher level. At system level of abstraction, multi threshold voltage, V_{th}, recommends to use high V_{th} devices on the non-critical paths which reduces the leakage power without increasing the total delay of the design [8]. Similarly, multi V_{dd} systems and variable frequencies are enabled according to current needs as higher V_{dd} and higher operating frequencies generally translate to higher power dissipation. Moreover, Frequencies can even be cut off parts of the design in idle mode using clock-gating to eliminate switching power [9]; or turning off V_{dd}/V_{ss} to the unused blocks temporarily to mitigate leakage power using power-gating [10]. Moreover, the increasing performance requirements of digital signal processing systems causes the capacity of memory integrated in systems to become larger; this increases the power consumption of such systems. Many domain-specific approaches are performed to improve the system energy efficiency [11]. Pipelining and multi-core processing (parallelism) are used for low power system designs at architecture level of abstraction. Also, software and HDL coding style can minimize power consumption for designs targeting reconfigurable devices [12][13].

On the other hand, approximate computing appears recently to decrease the power/energy consumption. In the next section, the approximate computing idea, and techniques at different levels of abstractions is explained.

1.4. Approximate Computing for Saving Energy

Data-intensive applications like machine learning, data analytics, web search, and digital signal processing (image, audio, video, and graphics) are main processing bottlenecks in data center and server applications. A perfect result is not necessary and an approximate result is sufficient in these applications because of human perceptual limitations in recognizing high-frequency patterns, redundant input data, or noisy inputs [14]. As example, for image processing, users of smartphones and mobile devices want to take photos and to have all the pictures available on the devices. However, the amount of solid-state storage available in these devices is limited and these devices are usually backed to the cloud. So, it is acceptable to allow image quality degradation, as there can always be a full-quality version of the image stored in the cloud [15]. Also, distributed applications and web services, such as online stores and social networks, are expected to be available, responsive, and fault-tolerant [16].

In the past few years, approximate computing has gained a lot of research attention. Nevertheless, it is related to the concept of stochastic/probabilistic computing [14]. Stochastic computing (SC) started by von Neumann in his work on probabilistic logic in 1956 to find low cost alternative to conventional binary computing [17]. Though, stochastic computing leads to very low complexity arithmetic; it has long computation time and low accuracy [18]. Approximate computing allows a significant improvement in energy efficiency for systems and applications that can tolerate some loss of quality of the computed result such as global positioning system (GPS) sensors, and speech recognition. These complex systems need complex algorithms to give good results quickly. And to be energy efficient, approximation is a way to meet these goals.

Approximate computing can be applied at software, architecture, circuit, and logic levels of abstraction. For instance, certain calculations or memory accesses, which are not critical to the application's final output quality, are selectively ignored in the programming language [19] [16]. At the transistor level, logic complexity can be reduced by reducing the number of transistors, and therefore the load capacitances. Gupta et al. propose various imprecise full adder cells, and use them to design approximate n-bit adders and approximate arithmetic unit. They depend on reducing switched capacitance, and significantly shorting the critical paths which enable voltage scaling [20]. For circuit level of abstraction, a less accurate computation circuit design or a reduced supply voltage for certain hardware components are suggested to trade off energy and accuracy [21][22]. In several suggested approximate adders, design is divided into two sections; upper accurate part of most significant bits and lower approximate part of less significant bits. Imprecise full adder cells are used for the addition of the lower approximate part [23][24][25]. On the other hand, Yazdanbakhsh et al. in [26] suggest a Verilog syntax annotation for the designer to identify the parts of the design that need to be approximate in the synthesis process. There are some recent efforts for high level approximate synthesis. Nepal et al. in [27] generate approximate designs depending on the algorithmic structure of the input high-level behavioral description. Their proposed synthesis methodology uses an iterative stochastic greedy approach that generates variant approximate designs. Their main techniques for approximation are truncating the size of the intermediate signals, simplify arithmetic computations, use approximate arithmetic circuits, and/or approximately unroll loops. The variant approximate designs are compared with the original exact design in terms of functional accuracy, power, area, and time to identify the optimal inexact designs. Moreover, Li et al. in [28] developed an approach for scheduling and binding considering approximate circuits.

For architecture level of abstraction, the processor designs that support approximate computing depend on identifying the instructions or code segments that can be run in approximate mode to enhance the traditional code running on general purpose processors to energy efficient code. For fine-grained approximate computing, an instruction set architecture (ISA) may define a set of special instructions that allow the compiler to convey what can be approximated without specifying how. Then in processor architecture, approximation technique can be freely chosen. It avoids the overheads of dynamic correctness checks and error recovery [29]. So, runtime and design time approximation techniques are separated and processor can contain both precise and approximate data paths, as shown in Figure 1.6. For coarse-grained approximate computing, segments that can be run in approximation mode are unloaded where runtime and design time are correlated. So, processor can contain some of its cores operate at an aggressive voltage to run these segments to be energy efficient with the sacrifice of the precision [30].

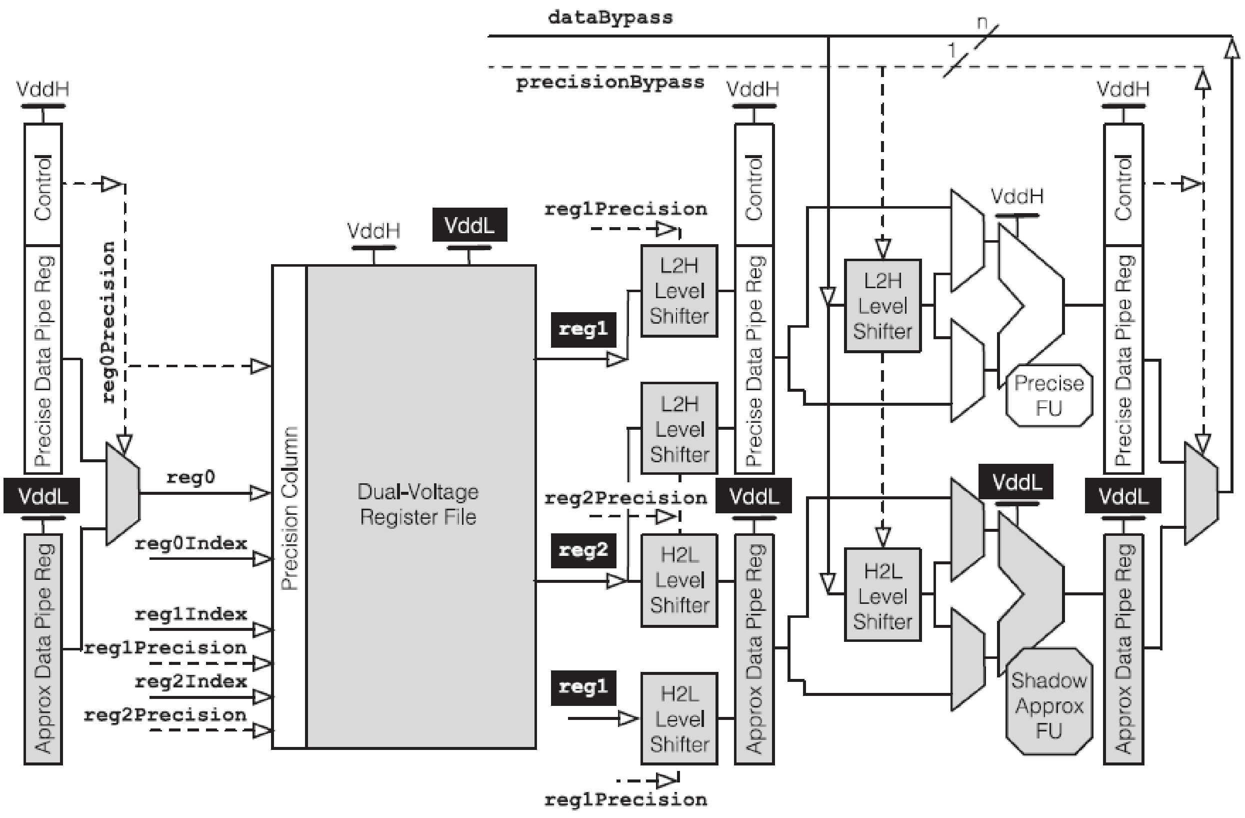


Figure 1.6 Approximate computing in architecture level of abstraction [29]

1.5. Thesis Overview

The thesis target low power with low latency designs for low energy portable applications. The first design targets applications that have decimal computations such as contactless smart card, and wireless sensor networks, as well as, mobile application processors for the second design.

The thesis is organized as follows: Chapter 2 introduces low power/energy design at the circuit level of abstraction for one of the important operator of the computer arithmetic unit, multiplication. A low energy clock-gated pipelined dual base binary/decimal fixed-point multiplier, DBM, is suggested extending a previously proposed non-pipelined design. A comparison between the suggested DBM multiplier design and the previously proposed binary/decimal multipliers are discussed in chapter 3 as well as the simulation results. Chapter 4 presents an introduction in memory hierarchy and lossless compression as an introduction to a low energy design for memory system at architecture level of abstraction. A literature review in basic lossless data coding, statistical, and dictionary based techniques is also discussed. Then, a new low energy lossless compression ASIC design is suggested in chapter 5 for memory/cache data compression/decompression. Finally, conclusions and future work are presented. Appendix A is added at the end of the thesis for the RTL synthesis flow commands explanation using synopsis design compiler.

Chapter 2 : Low Energy Pipelined Dual Base (Decimal/Binary), DBM, Multiplier

2.1. Introduction

There are various approaches for low power/energy multipliers at most design abstraction levels. At system level, optimum partitioning and dynamic power management of the design result in orders of magnitude power reduction [31]. At algorithmic level, the topology of the partial products accumulation tree affects power, speed, and area of the multiplier significantly. Different topologies are used for multipliers, such as Dadda, Wallace, Three Dimensional Method (TDM), Baugh-Wooley, and Overturned-Stairs adder tree design [32]. At architecture level, parallelism, pipelining, power gating [10] , and data encoding are major strategies that decrease energy dissipation. At logic level, power can be reduced by controlling switching activity. Moni and Sophia [12] monitor the input of the multiplier and detect the non-effective ranges, then, deactivate unusable circuitry to reduce switching activity. In addition, the output product is truncated (output precision is compromised) to decrease the power consumption and increase the speed. Sharma [13] uses bus inverter encoding to use either the original information or its inverted form depending on minimum switching activity. Row and column bypassing is used by Yan and Chen [33] , and Wen et al. [34] to save switching power. Their designs achieve high power reduction with a small area overhead. Abid et al. [35] use NAND gates instead of AND gates to generate most of the multiplier partial product bits. Thereby, the power consumption and the total number of needed transistors is decreased with little increase in delay. Furthermore, many techniques are used to decrease power at transistor/layout level [36][37][38][39]. However, power optimization at architecture, algorithmic and system levels achieve considerable power reduction compared to other levels' techniques [40].

On the other hand, commercial databases' numeric data are decimal. This is evidenced by a survey on a wide range of applications including airline systems, financial applications, inventory control, management reporting, marketing services, etc. [34]. Decimal arithmetic is supported by many programming languages, such as C, Visual Basic, COBOL, Java, and Python [35]. However, on the hardware side, microprocessors typically use binary arithmetic. Therefore, initial benchmarks indicate that applications spend 50% to 90% of their time in decimal processing overhead including binary/decimal conversions [34]. Moreover, software solutions are not accurate because they cannot exactly represent decimal fractions [35]. Dramatic advances in VLSI technology allow the hardware realization to replace the software realization of many complex functions such as decimal arithmetic. The hardware implementation of a complex function consumes less energy than its software equivalent and is even faster [36]. Thus, decimal arithmetic specific hardware is more efficient when dealing with commercial and financial data. Yet, the binary arithmetic specific hardware is still optimal in many applications including simulation programs where conversion time is much smaller than the run time. Also, binary addresses handle a wider range of addresses than decimal ones [41]. Thus, the combined binary/decimal arithmetic hardware arose, where each can be alternatively used according to the application at hand.

2.2. Available Combined Binary/Decimal Multipliers

Decimal hardware significantly reduces the arithmetic delay in commercial and financial applications eliminating programming and conversion overheads [42]. Decimal hardware also gives accurate results with fraction numbers, as the most of binary representation of decimal fractions are not accurate. However, binary hardware is still optimal in many major applications and binary addresses handle a wider range of addresses than decimal ones [42]. Literally, in financial data center processors, two hardware arithmetic units are required; a binary one and a decimal one. Therefore, combined binary/decimal arithmetic is suggested in literature to support both decimal and binary applications simultaneously offering a high speed and low area solution.

Parallel multiplication, in general, consists of three basic stages: multiples generation, partial products selection, and partial products accumulation. Early decimal multipliers generate all decimal multiplicand multiples, from $2x$ to $9x$. To reduce the area and delay, a reduced set of decimal multiplicand multiples was generated [43]. The remaining multiples are obtained dynamically during the algorithm. A signed digit, SD, recoding technique for the pre-generated multiples was presented in [44][45][46]. SD recoding allows the use of negative multiples from the set and reduces area and delay. Multiplexers controlled by the multiplier digits are used for partial products selection. If all decimal multiplicand multiples are generated in the previous stage, only one multiplexer is needed. But, if a reduced set of the multiplicand multiples are generated, a multiplier recoding is needed to represent each multiplier digit. Thereby, two/three multiplexers, for secondary/tertiary sets, are needed to choose the suitable multiplicand multiples for each multiplier digit. When all multiplicand multiples are generated, one partial product for each multiplier digit, 4-bit, is selected. So, n/4 partial products are generated, where n is the number of multiplier bits. The presence of all possible multiples leads to a smaller number of partial products. But, the associated carry propagate addition has a large multiples generation delay, $O(log_2 n)$. On the other hand, when a reduced set of multiplicand multiples are generated, two/three partial products for each multiplier digit are selected. So, $\frac{n}{2}$ / $\frac{3n}{4}$ partial products are generated for secondary/tertiary sets. Consequently, the number of partial products is increased leading to extra two/three levels in the carry save adder (CSA) accumulation tree. But the associated multiples generation delay is approximately of $O(1)$. Originally, decimal addition is implemented using binary adders with decimal correction block for every digit. Adding 6 to the digit if the digit's value exceeds 9 to correct the binary sum digits [47]. A new decimal adder presented by Vázquez et al. [46] uses the binary carry save hardware with all-valid BCD formats, such as BCD-4221 and BCD-5211. These BCD formats have a valid decimal value for all 16 combinations. Thus, they attain a valid sum and carry outputs which deduct the area and delay needed for correction blocks. Only a decimal multiplication by two for the carry outputs is used after each CSA level.

In this work the partial products accumulation stage is mostly investigated and worked on because this stage has the most significant area and delay. Integrating binary and decimal operations using conventional binary and decimal addition leads to extra delay in the multiplication path. Multiplexers used in binary/decimal selection overload binary and decimal paths. Also, decimal correction blocks add up to binary multiplication path delay. Dadda [48] suggests binary column addition for each decimal digit then converts the binary output sum of each column into its decimal format. Using this idea in integrating binary/decimal additions separates decimal correction blocks from binary path, and eliminates the usage of multiplexers. A shared partial products accumulation tree was proposed by Vázquez et al. in their 64-bits combined binary/decimal multiplier design [46]. A shared CSA tree is used for binary and decimal accumulation. Carry multiplication by two in the tree uses multiplexers to select between binary and decimal multiplication by two. The use of multiplexers leads to an increase in area and delay especially those of the binary path. Hickmann et al. proposed three combined binary/decimal designs [49]. The first is a 64-bits multiplier which uses a shared binary/decimal tree with improved CSA tree design. This design decreases the area but has the same binary/decimal delay of Vázquez et al. Its binary delay is larger than that of a standalone binary design. The second 64-bits design uses split binary and decimal CSA trees which has a small area increase if compared to the first design. However, this design decreases the delay of the binary multiplication by 41% with almost the same decimal delay. The third design is the same as the split design but uses 53bits/16digits. The 53-bits and 16-digits are the lengths of the significands of double precision binary and decimal floating-point numbers respectively. The second design of Hickmann et al. has a significant contribution. So, the second design of Hickmann et al. along with the design of Vázquez et al. are chosen as reference designs to compare and evaluate the architecture/system modifications proposed in this work.

The pipelined Dual Base (decimal/binary) Multiplier, DBM, extends a previously proposed non-pipelined DBM design [50]. A block diagram of the non-pipelined DBM design is shown in Figure 2.1. The objective of the non-pipelined DBM design is separating the decimal and binary multiplication paths without increasing the area. Consequently, the non-pipelined DBM design uses a shared binary column tree for both binary and decimal partial products accumulation. Then the binary and decimal paths are split for the last addition steps. This results in decreasing the area of the accumulation stage achieving a speed equal to that of the fastest known design, Hickmann et al.’s second design. Vázquez et al., as detailed in [51], and Hickmann et al. use radix-5 for decimal multiples generation stage. They compute 10's complement for negative multiples which takes some area and delay. Jaberipur and Kaivani [52] propose a direct combinational generation of 8X and 9X for decimal multiples to avoid the decimal negative multiples. The non-pipelined DBM design [50] uses radix-5 for decimal multiples generation stage. However, binary and decimal negative multiples are generated using one's complement, and 9's complement, implemented in two level gates, respectively. Then the (+1) is included in the binary column tree levels without extra delay in the accumulation stage.

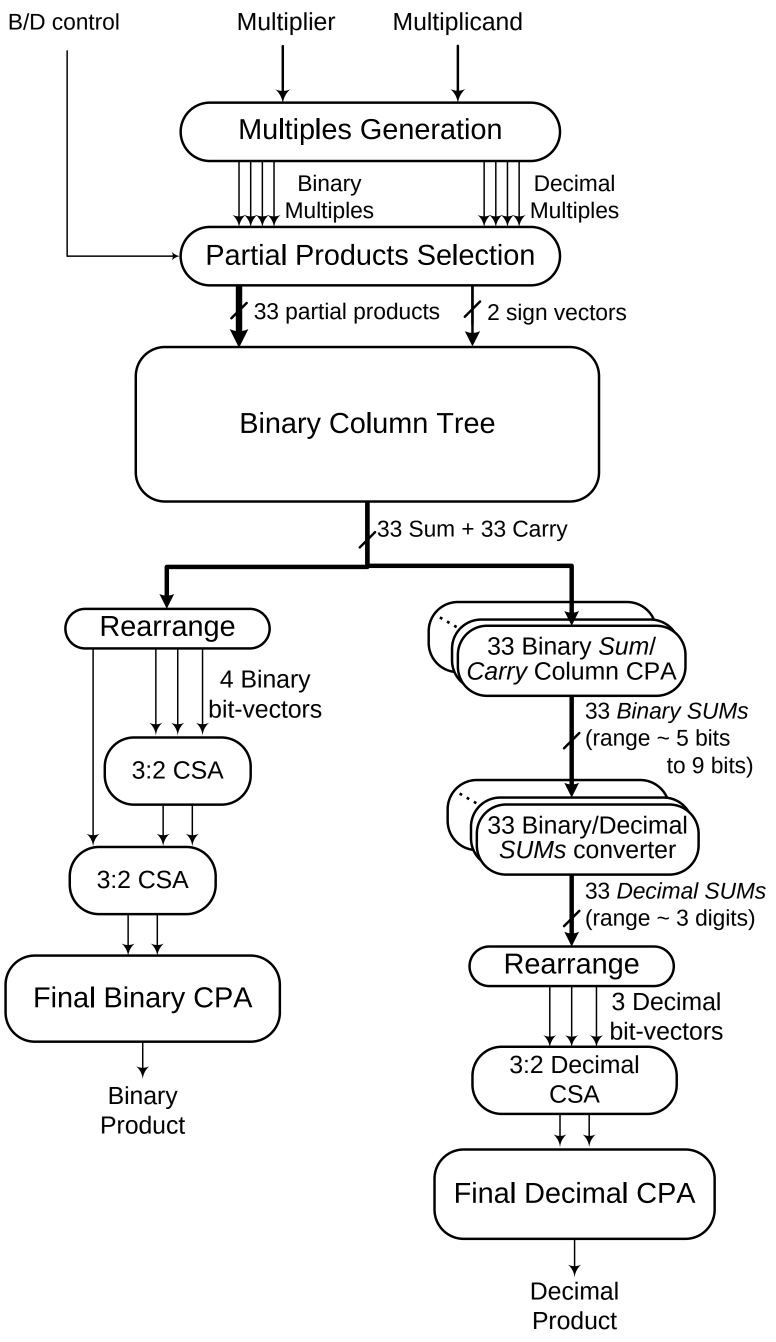


Figure 2.1 non-pipelined Dual Base (decimal/binary) Multiplier, non-pipelined DBM. [50]

2.3. Low Energy Pipelined Dual Base (decimal/binary) Multiplier, DBM, Design

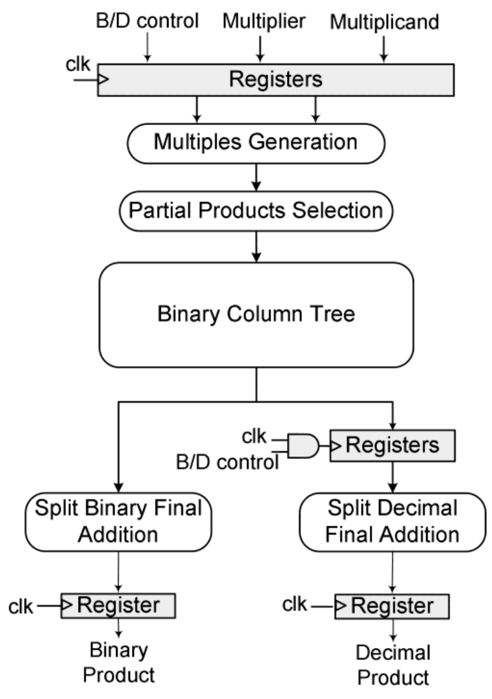
Energy reduction is based on power (static and dynamic) and the operating frequency. Pipeline stages are chosen to achieve low energy. The shared binary column tree in the DBM design achieves lowest area if compared to previously proposed binary/decimal accumulation trees and thus consumes lowest static power as will be detailed in the simulations. Clock gating is used in the split binary/decimal part of the accumulation stage. Clock gating eliminates the switching activity of the binary multiplication part during the decimal multiplication. Thus, when a binary multiplication is running, the decimal multiplication hardware remains idle, thereby, consuming no dynamic power dissipation and vice versa. Moreover, the multiplier enable is clock gated to stop the switching activity when the multiplier is not in use.

For pipelining purposes, dividing the design into too many pipeline stages would result in too many registers, thereby, increasing area and power. To decide the adequate number of pipeline stages for binary and decimal multiplication, the delay of each multiplication stage of the non-pipelined DBM is studied first. Then, various pipelining structures (as shown in Figure 2.2) were implemented and simulated to decide the minimum energy design. The comparison among the attempted pipelined designs is presented in the simulation section. Accordingly, pipelining both binary and decimal multiplication into two stages is chosen to be the pipelined DBM.

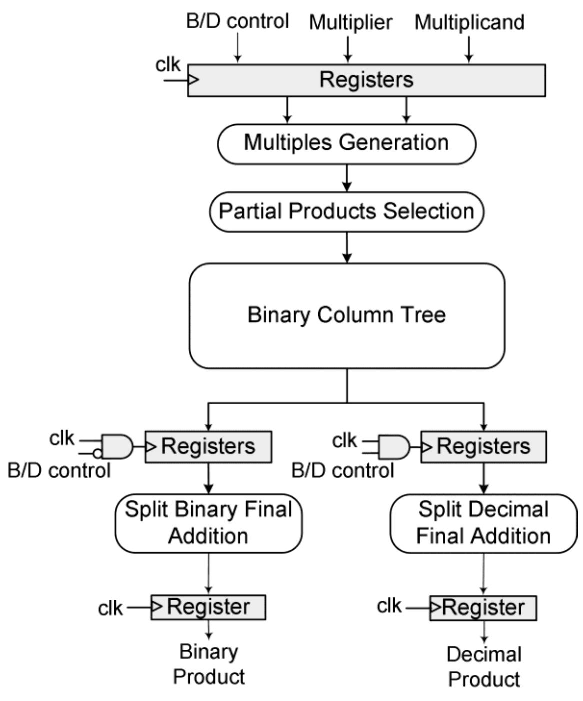
The pipelined DBM design that we will use throughout the rest of this thesis is detailed in Figure 2.3. The design has two 64-bits operands, $Multiplicand\ A$ and B, Binary/Decimal control signal, $B/D\ control$, and a multiplier enable signal, $Mult\_en$. $B/D\ control$ signal is used to determine whether the operands are binary or decimal. The operands are binary when the $B/D\ control$ is '0'. Otherwise, operands are Binary Coded Decimal with weight 8421 (BCD-8421), when the $B/D\ control$ is '1'. $Mult\_en$ signal is used for multiplier global clock gating. To ensure that the $B/D\ control$ signal is correct and stable before the second pipeline stage, it is assigned to a negative edge register. The $B/D\ control$ signal is input to the clock gating AND gate. The stages of the DBM design are detailed in the following sections.

2.3.1. Multiplicand Multiples Generation Stage

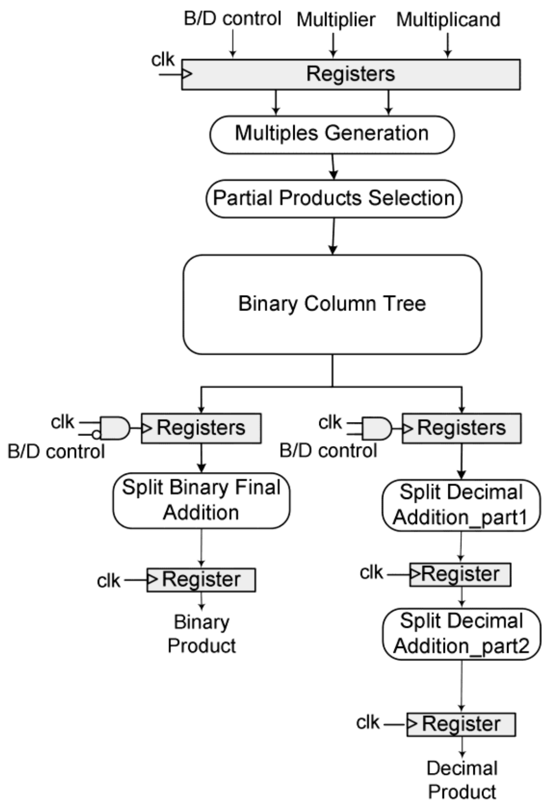
The $Multiplier$ is represented by 16 digits, each group of 4-bits is considered as one digit for both binary and decimal multiplications. So, multiplicand multiples $\{A$ to $15A\}$, and $\{A$ to $9A\}$ are required for binary and decimal multiplications respectively. The DBM design uses radix-16 booth multiplication for binary recoding. It has a $Multiplicand$ multiples set of $(\pm A, \pm 2A, \pm 4A, \pm 8A)$ which is generated using $(0, 1, 2, 3)$ shift registers respectively. This binary recoding significantly reduces the area, delay, and power needed to have $Multiplicand$ multiples from A to $15A$. Negative multiples are generated using 2's complement operation. One's complement is first generated in this stage by inverting the positive multiple bits using NOT gates. Then in the partial products selection stage a sign bit is generated to be added to partial products in the accumulation stage. Decimal multiples use BCD-8421 signed-digit radix-5 recoding with $Multiplicand$ multiples set $(\pm A, \pm 2A, 5A, 10A)$. $2A$ and $5A$ multiples are generated using shifting and conversion between different BCD formats [21] as shown in Figure 2.4. The decimal $10A$ multiple is generated using 4-bits shifting.



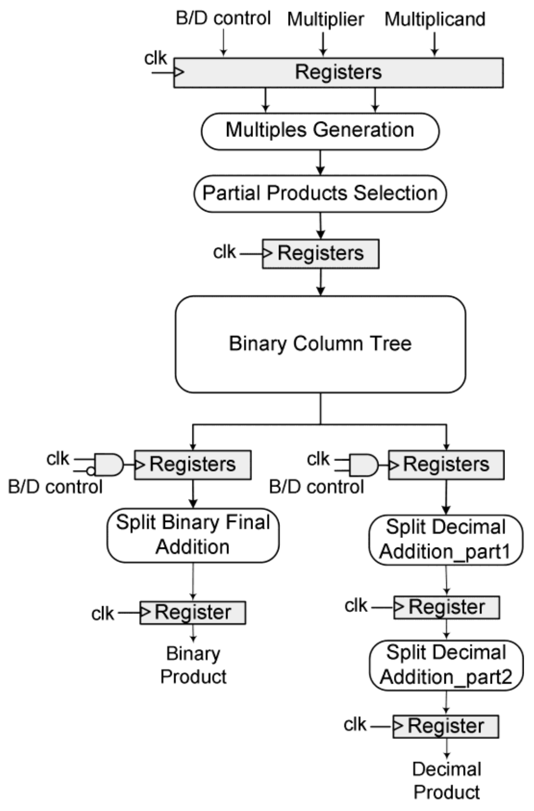
Scheme A



Scheme B



Scheme C



Scheme D

Figure 2.2 Pipelining Schemes. (Scheme A) for one binary stage and two decimal stages. (Scheme B) for two binary stages and two decimal stages. (Scheme C) for two binary stages and three decimal stages. (Scheme D) for three binary stages and four decimal stages

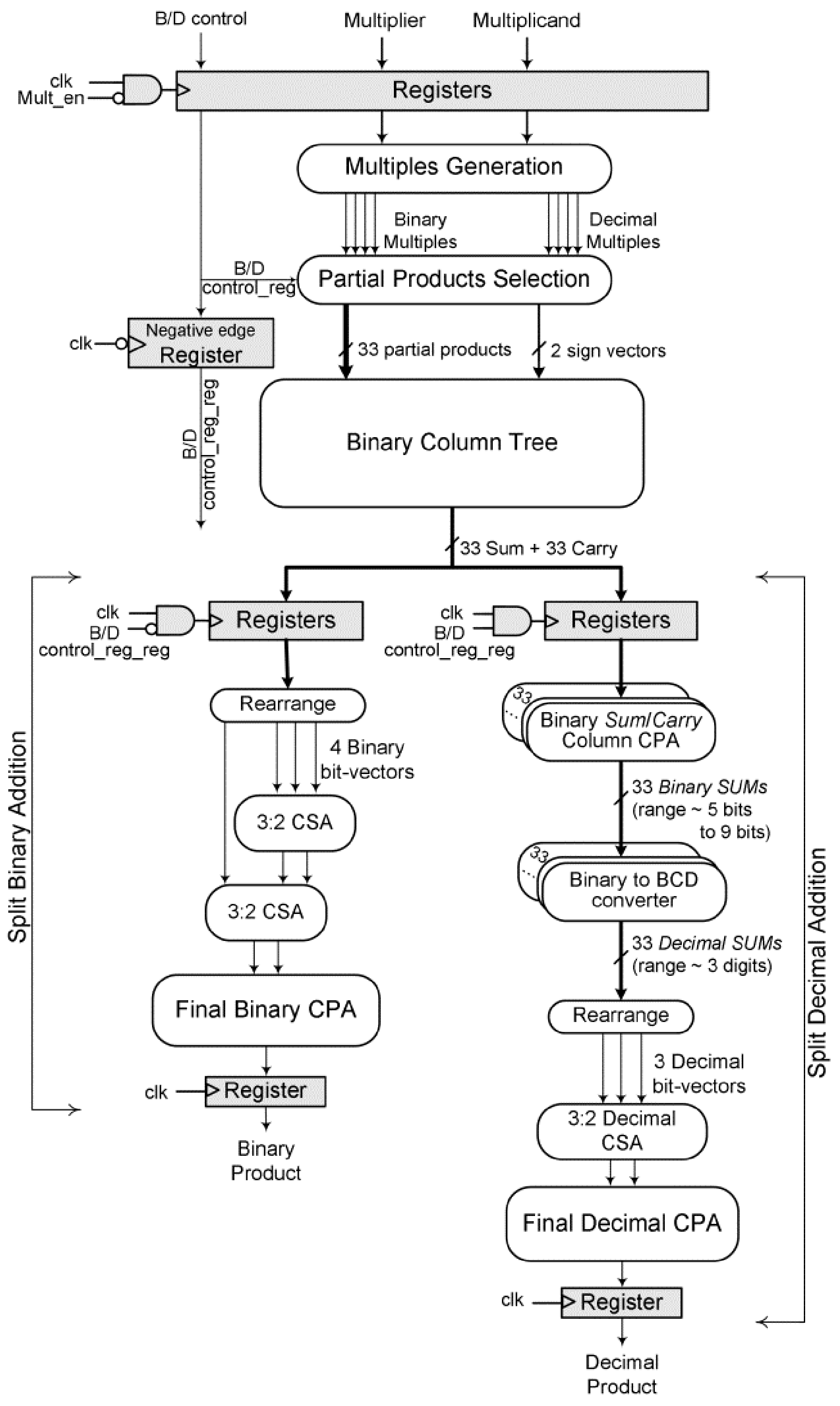


Figure 2.3 Proposed pipelined Combined Binary/Decimal Multiplier.

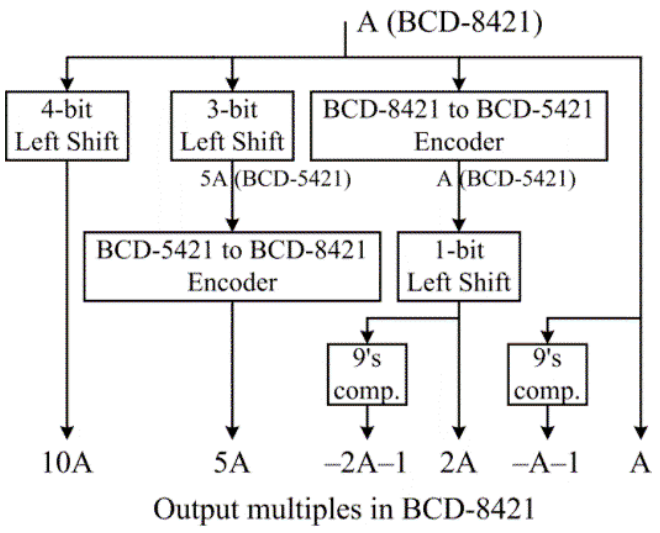


Figure 2.4 Decimal multiples generation

A nine's complement is obtained for each digit for negative multiples. Then, a sign bit is generated at the partial products selection stage to get the 10's complement of the $Multiplicand$ multiple.

2.3.2. Partial Products Selection Stage

The partial products selection block diagram for binary and decimal multiplications is shown in Figure 2.5. Each of the binary and decimal Multiplicand multiples sets is divided into two groups. The binary set is divided into (–2A, –A, A, 2A) and (–8A, –4A, 4A, 8A) groups. The decimal set is divided into (–2A, –A, A, 2A) and (5A, 10A) groups. For binary radix-16 booth multiplication, the Multiplier is padded with one '0' bit to the right and four '0' bits to the left. So, the Multiplier is divided into 17 digits. Two Multiplicand multiples are selected for the first 16 Multiplier digits. The last Multiplier digit, composed of the last five bits, is "00000" or "00001". So it needs only one partial product for binary multiplication, where it selects between 0 or A Multiplicand multiple. Negative multiples are needed in the two groups, so two sign bits are generated for each Multiplier digit. The decimal partial products selection according to BCD-8421 signed-digit radix-5 recoding is shown in Table 2.1.

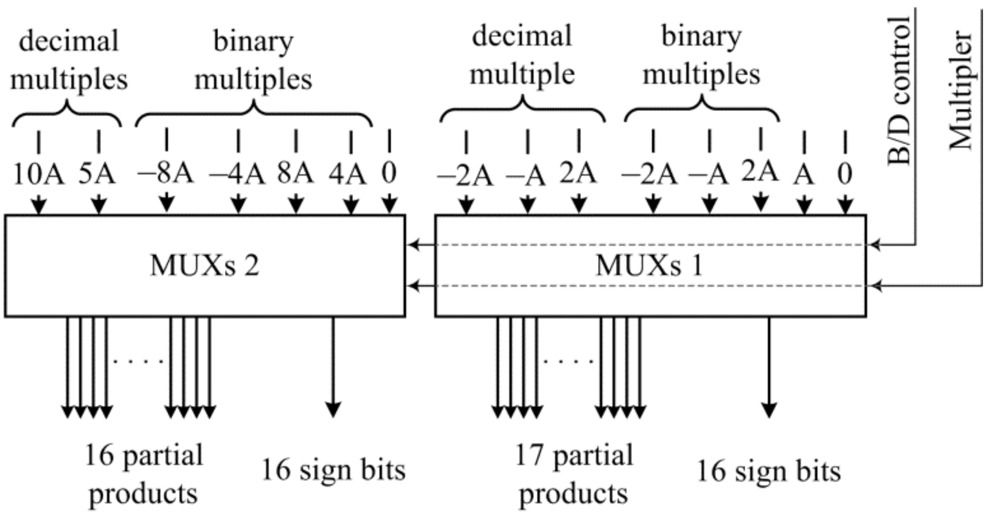


Figure 2.5 Partial products selection

Table 2.1 Decimal Multiplicand multiples selection

Multiple	**MUX1 multiple selection**	**MUX2 multiple selection**
0	0	0
A	A	0
2A	2A	0
3A	-2A	5A
4A	-A	5A
5A	0	5A
6A	A	5A
7A	2A	5A
8A	-2A	10A
9A	-A	10A

The conditions that control the multiplexers depend on $Multiplier$'s current digit, $(b_{i+3}b_{i+2}b_{i+1}b_i)$, most significant bit of previous $Multiplier$ digit, b_{i-1}, and $B/D\ control, c$. The equations that control the two multiplexers are as follows;

For MUXs1

$$cond.1_B = b_i\bar{b}_{i-1} + \bar{b}_i b_{i-1} \tag{2.1}$$

$$cond.1_D = \bar{b}_i b_{i+2} + b_i\bar{b}_{i+1}\bar{b}_{i+2} \tag{2.2}$$

$$cond.1 = cond.1_B \cdot \bar{c} + cond.1_D \cdot c \tag{2.3}$$

$$cond.2_B = \left(\overline{b_0}b_1\overline{b_{-1}} + b_0\bar{b}_1 b_{-1}\right) \cdot \bar{c} \tag{2.4}$$

$$cond.2_D = \left(\overline{b_0}b_3 + b_0b_1 + b_1\overline{b_2}\right) \cdot c \tag{2.5}$$

$$Inv.1_B = \left(\overline{b_0}b_1 + b_1\overline{b_{-1}}\right) \tag{2.6}$$

$$Inv.1_D = (b_3 + b_0b_1\overline{b_2} + \overline{b_0}\bar{b}_1 b_2) \tag{2.7}$$

For MUXs2

$$cond.4_B = \left(b_1\overline{b_2} + \bar{b}_1 b_2\right) \cdot \bar{c} \tag{2.8}$$

$$cond.8_B = \left(\overline{b_1}\overline{b_2}\, b_3 + b_1b_2\overline{b_3}\right) \cdot \bar{c} \tag{2.9}$$

$$cond.5_D = \left(b_0b_1 + b_2\overline{b_3}\right) \cdot c \tag{2.10}$$

$$cond.10_D = (b_3) \cdot c \tag{2.11}$$

$$Inv.2_B = (b_3) \tag{2.12}$$

Where $cond.2_B$ abbreviates condition for choosing binary multiplicand $2A$; $Inv.1_B$ abbreviates condition for choosing negative binary multiplicand for MUXs1; $Inv.2_B$ abbreviates condition for choosing negative binary multiplicand for MUXs2, and similarly for other abbreviations.

2.3.3. Partial Products Accumulation Stage

The partial products accumulation stage is divided into two parts: shared binary column tree and split binary/decimal addition. The main part is the shared one. The shared binary CSA column tree design, based on Dadda's column tree [48], does not cause any area or delay overhead in both binary and decimal paths. Its output column Sums and $Carry$s are rearranged twice for binary and decimal split paths to multiply each bit by its relevant weight. Shared binary column tree produces four vectors for binary rearrangement and three vectors for decimal. Clock gating is used at the beginning of split binary/decimal partial products addition.

2.3.3.1. Binary Column Tree

The 33 partial products with the two signed vectors are added using a CSA binary column tree as shown in Figure 2.6. Each 4-bits column represents a binary number from 0 to 15 or a BCD-8421 number from 0 to 9 depending on $B/D\ control$ signal. Whether binary or BCD-8421, a binary tree is used to add each column digits to generate Sum and $Carry$ for each column. Each column has a different number of digits; the maximum number of digits per columns is 33 digits plus two sign bits. Each column's Sum and $Carry$ does not exceed 8-bits and is represented in a binary format. “Column 15” tree which has the maximum number of digits is shown in Figure 2.7. “Column 15” tree is a 33 digit CSA binary column tree. The two sign bits are entered to the first bit of shifted carry digits inside the tree. A binary tree is implemented here to save the correction delays of decimal path due to the six invalid BCD-8421 digits. The column tree does not pass the carry bit to the next digit. Where, carry bit is of order 16 in binary addition and of order 10 in decimal addition. The column tree outputs 33 Sums and 33 $Carry$s with bit lengths ranging from 4-bits to 8-bits. In the next step, the column tree outputs are rearranged twice, one for binary split path and another for decimal split path. Rearranging of the column tree output digits multiplies each digit by its relevant weight for binary and decimal paths.

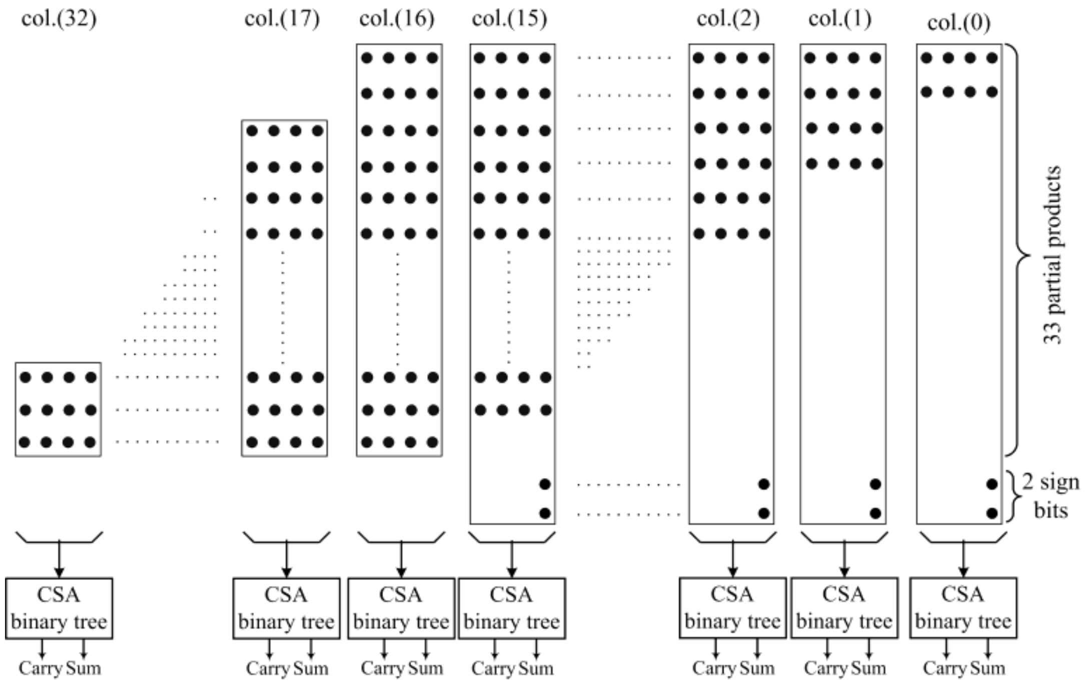


Figure 2.6 Binary column tree scheme.

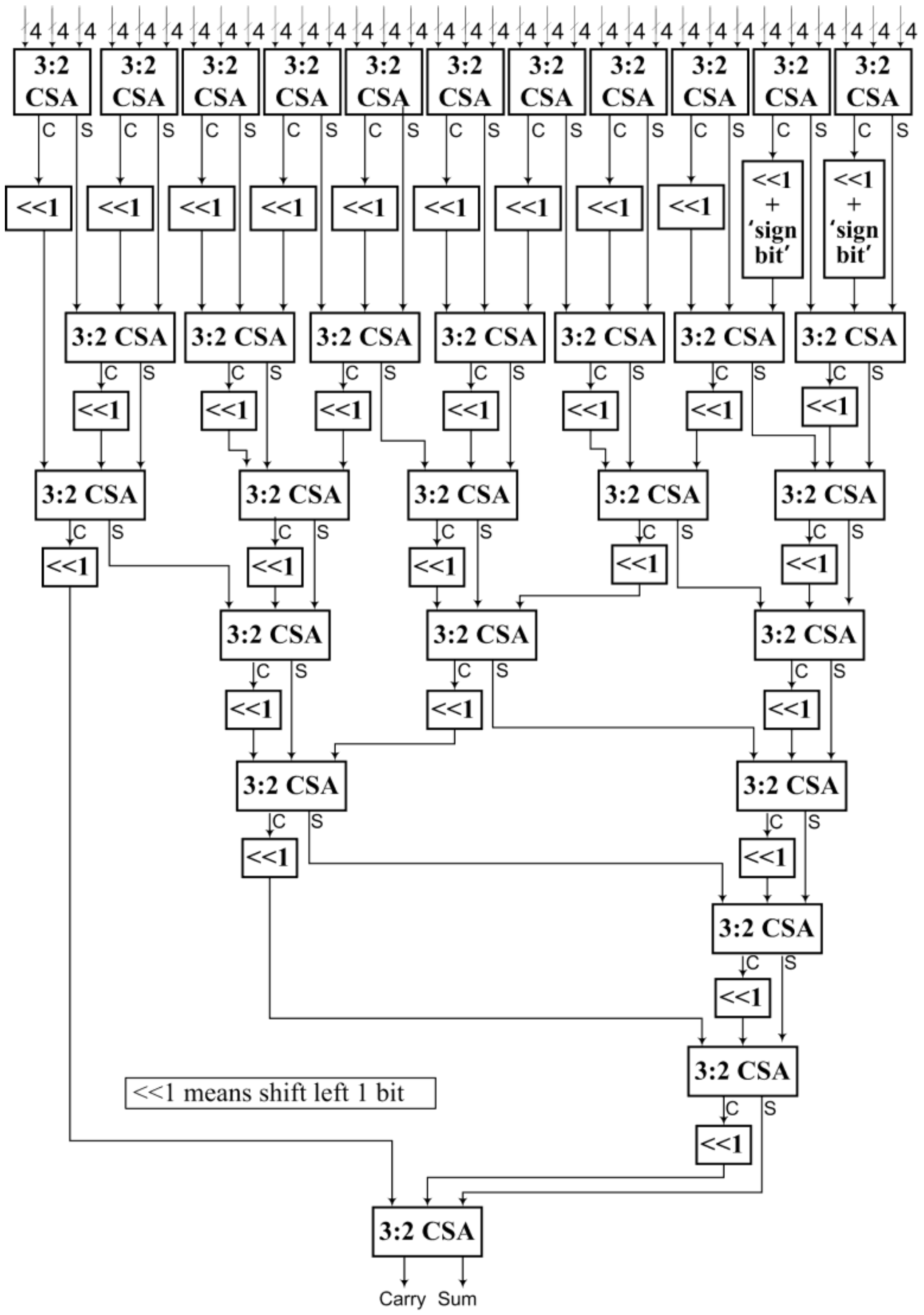
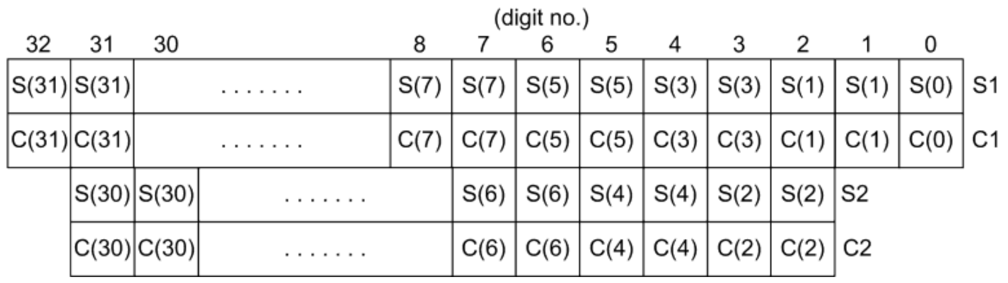


Figure 2.7 "Column 15" binary CSA tree

2.3.3.2. Split Binary Addition

The output of the column tree is rearranged into two major partial Sums and two major partial Carrys. The four binary bit-vectors output after rearranging are shown in Figure 2.8. A small tree is used to add the four binary bit-vectors. The tree consists of two CSAs and one Carry Propagate Adder (CPA). The scheme of the CPA based on Kogge-Stone's design [53] is shown in Figure 2.9.



(digit no.)
32 31 30 8 7 6 5 4 3 2 1 0
S(31) S(31) .... S(7) S(7) S(5) S(5) S(3) S(3) S(1) S(1) S(0) S1
C(31) C(31) .... C(7) C(7) C(5) C(5) C(3) C(3) C(1) C(1) C(0) C1
S(30) S(30) .... S(6) S(6) S(4) S(4) S(2) S(2) S2
C(30) C(30) .... C(6) C(6) C(4) C(4) C(2) C(2) C2

Figure 2.8 The four binary bit-vectors after rearranging.

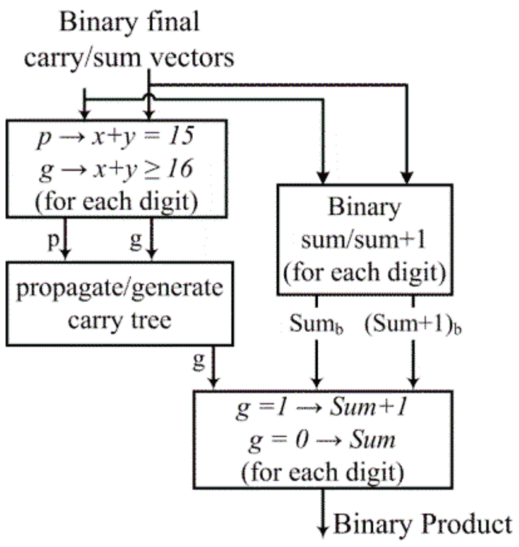


Figure 2.9 Scheme of the final Binary CPA.

2.3.3.3. Split Decimal Addition

The CSA column tree outputs Sum and $Carry$ for each column. First, a simple binary Kogge-Stone CPA is used to add each column Sum and $Carry$. Binary CPA is used to decrease the number of decimal additions which reduces area, delay, and subsequently energy. The output of the binary Kogge-Stone CPAs is 33 Binary SUMs having a range from 5-bits to 9-bits. Then, a binary to BCD converter is used to convert each Binary SUM output to a Decimal SUM, BCD-8421 [54]. Then, each Decimal SUM is converted to BCD-4221 to be ready for the next decimal addition. BCD-4221 format has valid decimal values for all 4-bits combinations which allows the use of binary CSA without large decimal correction [46]. The Decimal SUMs are rearranged to multiply each column output digit by its relevant weight.

At the end, a decimal CSA based on [46] followed by a decimal CPA adds the final three decimal bit-vectors. A scheme of the implemented decimal CPA design is shown in Figure 2.10and the three decimal bit-vectors generated after rearranging are shown in Figure 2.11.

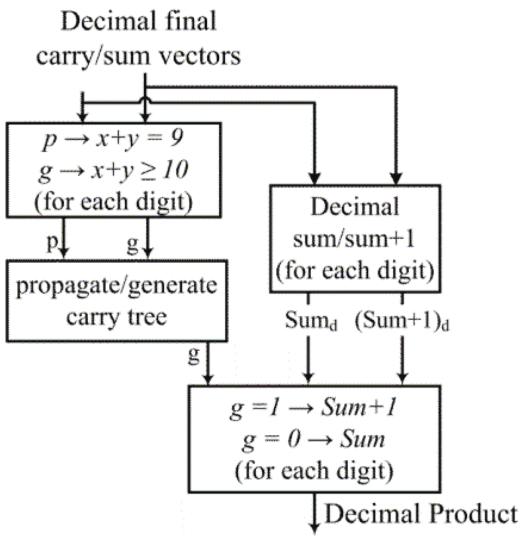
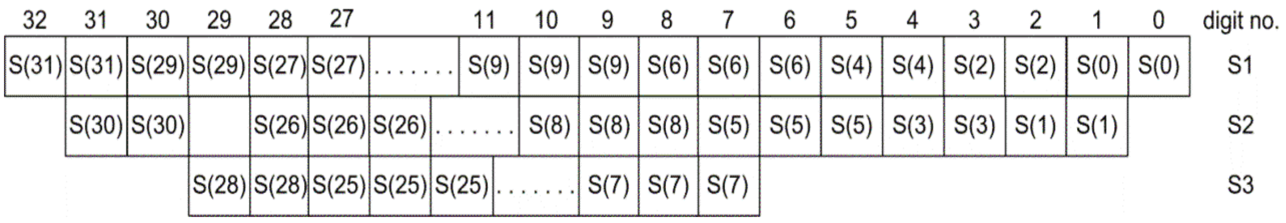


Figure 2.10 Scheme of the final Decimal CPA.



32	31	30	29	28	27		11	10	9	8	7	6	5	4	3	2	1	0	digit no.
S(31)	S(31)	S(29)	S(29)	S(27)	S(27)	....	S(9)	S(9)	S(9)	S(6)	S(6)	S(6)	S(4)	S(4)	S(2)	S(2)	S(0)	S(0)	S1
	S(30)	S(30)		S(26)	S(26)	S(26)		S(8)	S(8)	S(8)	S(5)	S(5)	S(5)	S(3)	S(3)	S(1)	S(1)		S2
			S(28)	S(28)	S(25)	S(25) S(25)			S(7)	S(7)	S(7)								S3

Figure 2.11 The three decimal, BCD, bit-vectors after rearranging.

2.4. Summary

This chapter introduces the available combined binary/decimal multipliers with details explanation for the non-pipelined binary/decimal DBM multiplier design. Then various pipelined binary/decimal multiplier designs are suggested with different pipeline stages division to be tested for lower energy consumption. Then clock gating is presented for the best performance pipelined DBM design for lower energy consumption.

Chapter 3 : DBM Multiplier Comparison and Results

3.1. Introduction

The binary/decimal multiplier designs of Vázquez et al. [46], Hickmann et al. [49], non-pipelined suggested design [50], and suggested pipelined designs are implemented using VHDL. First, for fast/preliminary evaluation of our design, the first suggested pipelined design with two binary stages and three decimal stages along with the previous designs are synthesized and implemented on FPGA. ISE Design Suite 12.1 and Xilinx XPower analyzer tool are used to simulate the designs on Virtex-5 xc5vlx50ff676 FPGA. A C program is written to generate random test cases with 50% binary and 50% decimal multiplication operations. 10,000 random test cases are used for FPGA simulations, where it takes about 10 hours to complete. Then, a simulation activity file, SAIF file, is generated using Xilinx ISim simulator for multipliers' dynamic switching activity simulation.

Then, for more accurate simulation results, designs are simulated for ASIC implementation. Thus, the VHDL designs are synthesized in Synopsis Design Compiler B-2008.09 with TSMC 65nm low power CMOS standard cell library, tcbn65lpbwp7t, operating at 1.2V. Moreover, the designs are synthesized with the open cell libraries NanGate 45nm [55] and NanGate 15nm [56] at their typical operating voltage 1.1V and 0.8V, respectively. Thereby, the performance of area, delay, power, and energy of implemented designs are studied at smaller technology nodes. All simulation results are generated using default design compiler timing script with clock, input, output, load, and wire load constraints per technology file. For power analysis, the switching activity for 120,000 test cases is used. For pipelined design, the clock uncertainty is set to 10% of the clock period. The previously published combined binary/decimal multipliers are non-pipelined. Thus, for impartial comparison, the power and energy of the non-pipelined DBM design is studied against those of the other previously published multiplier designs first. Then, the different pipeline schemes are simulated for optimum area, energy, and throughput tradeoffs and the choice of the suggested pipelined DBM design with two binary stages and two decimal stages is discussed. Then, the simulation results of the pipelined DBM design are presented. At the end, the total power distribution among leakage and dynamic power for all multiplier designs at different technology nodes is presented. Dynamic power is divided into switching and internal power for interconnect and gates' internal dynamic power, respectively. Also, the power breakdown among multiplier components is analyzed.

3.2. FPGA Simulation Results

Table 3.1 shows area, delay, and power values for the two previously proposed combined binary/decimal multiplier designs and the proposed pipelined multiplier design with two binary pipelining stages and three decimal pipelining stages. The total delay of one multiplication operation of the proposed combined binary/decimal design is approximately equal to the delay caused by the previous two combined binary/decimal designs. The suggested design has a tiny extra delay for pipelining registers. The maximum clock frequency of the suggested multiplier is 75MHz. The area of the suggested design is less than that of Vázquez et al. and that of Hickmann et al. by 22.5% and 30.4% respectively. The power consumption of the suggested design is less than that of Vázquez et al. by 31.8% and less than that of Hickmann et al. design by about 56.4%.

Table 3.2 shows the detailed power dissipation for the three designs running at frequencies of 25 MHz, 25 MHz, and 77 MHz for Vázquez et al., Hickmann et al., and the suggested multiplier design, respectively. The suggested pipelined design is divided into three decimal stages thus its operating frequency is three times the non-pipelined ones. Hickmann et al. IOs power dissipation is very high because it duplicates the number of output pins due to split tree. Hickmann et al. IOs' power dissipation is almost double that of Vázquez et al. IOs power dissipation. In our suggested multiplier design, clock gating decreases switching which obviously decreases IOs power dissipation. Logic, signal, and leakage power dissipation in our proposed design are slightly less than in previous designs due to area reduction.

Table 3.1 Worst path delay, area and power consumption of the proposed design and the previously published designs

Multiplier Design	**Vázquez et al. design** [46]	**Hickmann et al. design** [49]	**Proposed pipelined design (Scheme C)**
Worst path delay (ns)	39.8	Decimal path ≈ 37.6 Binary path ≈ 32.4	min T_{clk} = 13
FPGA utilization	48.1%	53.6%	37.3%
Power consumption (W)	2.895	4.533	1.975

Table 3.2 Detailed power dissipations distribution (in Watts) for the proposed pipelined design and the previously published designs

	Vázquez et al. design [46]	**Hickmann et al. design** [49]	**Proposed pipelined design (Scheme C)**
Clocks	-	-	0.053
Logic	0.196	0.268	0.250
Signals	0.643	1.023	0.779
IOs	1.497	2.659	0.347
Leakage	0.559	0.583	0.546
Total	2.895	4.533	1.975

3.3. ASIC Implementation Simulation Results For The non-pipelined Designs

Area, power, and PDP (Power Delay Product) comparisons among the three non-pipelined designs at 100MHz simulating frequency are observed for different technology nodes. Results are depicted in Table 3.3 showing that the non-pipelined DBM architecture design provides significant decrease in area, power, and energy. For Hickmann et al. design [49], and non-pipelined DBM design [50], the average of the binary and decimal delay is used for calculating the PDP. Shared binary column tree greatly decreases the area of the DBM and consequently power. Binary and decimal paths split in the last addition steps attain good delay for both binary and decimal multiplication. That leads to best energy efficiency among the non-pipelined designs.

Minimum path delay is measured by synthesizing the design with very small clock cycle to get the output negative slack. Adding the absolute value of the slack to the small clock cycle gives an estimate value of the maximum operating frequency. Then, the design is synthesized at estimated maximum operating frequency with technology constraints to double check the maximum operating frequency. The minimum path delay is the reverse of the maximum operating frequency. The minimum binary and decimal path delays experimented at current technology used by industry, NanGate 45nm, and the smallest technology, NanGate 15nm are shown in Table 3.4. All non-pipelined designs have almost the same decimal path delay. The non-pipelined DBM's binary path delay is smaller than that of Vázquez et al. and almost the same as that of Hickmann et al. The non-pipelined DBM design allows an operating frequency of up to 4 GHz at 15nm technology with less area and energy than previously published combined binary/decimal multipliers.

Table 3.3 Area, total power dissipation, and PDP for non-pipelined designs at 100MHz.

	Area (mm^2)			Power dissipation (mW)			Average PDP (pJ)		
	TSMC 65nm	**NanGate 45nm**	**NanGate 15nm**	**TSMC 65nm**	**NanGate 45nm**	**NanGate 15nm**	**TSMC 65nm**	**NanGate 45nm**	**NanGate 15nm**
Vázquez et al. design [46]	59.11	41.14	11.19	19.69	10.134	2.63	187.06	61.37	1.88
Hickmann et al. design [49]	62.87	46.91	12.51	22.19	12.604	3.22	175.95	60.06	1.7
non-pipelined DBM design [50]	40.33	29.59	8.09	11.31	7.267	1.90	79.93	35.36	0.93

Table 3.4 Minimum path delay for non-pipelined designs at maximum operating frequency

	NanGate 45nm		NanGate 15nm	
	Decimal path (ns)	Binary path (ns)	Decimal path (ns)	Binary path (ns)
Vázquez et al. design [46]	3.5	3.5	0.251	0.251
Hickmann et al. design [49]	3.3	1.5	0.249	0.155
non-pipelined DBM design [50]	3.1	1.47	0.248	0.161

3.4. Binary/Decimal Pipeline Stages Selection Using NanGate-45 nm Technology

The delay of the different multiplier stages in the non-pipelined DBM design is shown in Table 3.5. The delay of the binary column tree is considered our reference for desired pipeline stage delay and the pipelining stages are decided accordingly. Pipelining allows one operation to be executed every clock cycle after first operation finished. So, dividing the multiplication operation into more pipeline stages will increase throughput, but will result in adding a large amount of registers. That will cause area and power increments. Throughput is calculated as follows

For non-pipelined designs $\text{Throughput} = 1/min.path\ \ delay$ (3.1)

For pipelined designs $\text{Throughput} = 1/min.clock\ cycle$ (3.2)

Various attempted pipelined structures, Figure 2.2, were implemented and simulated to get the most appropriate pipelined design. The pipelining attempts start by one binary stage and two decimal stages for scheme A. The slowest stage of scheme A is divided into two stages in scheme B. The slowest stage of scheme B is divided into two stages in scheme C and the same for scheme D. Pipelined designs are synthesized, before and after clock gating, at a simulating frequency of 100 MHz to compare area, delay, throughput, power, and energy as shown in Table 3.6. The previously published non-pipelined binary/decimal multipliers are included in Table 3.6 for comparison.

For pipelined schemes before clock gating, more pipelining stages lead to an increase in area and power due to the used registers and their switching activity effect on power. On the other hand, more pipelining stages allow the use of higher frequencies. The energy consumption combines these trade-offs and is the main factor for the chosen design. An extra pipeline stage at the output of the partial products selection stage in scheme D adds a large number of registers. These registers are needed to store all multiplication partial products, thereby, increasing area and power. Thus, scheme D design is excluded. Clock gating decreases the power consumption with negligible increase in area. It is started at the registers after the “Binary Column Tree” block. Scheme A is not fully clock gated as the “Split Binary Final Addition” is not clock gated. The clock gated scheme B gives minimum power consumption as it is fully clock gated with minimum extra registers. It also has a suitable delay so it gives good energy consumption. Clock gated schemes C and D increase power due to more registers' switching activity with almost the same effect of clock gating as in scheme B. Consequently, scheme B is chosen to be the pipelined DBM.

Table 3.5 Delay breakdown of each stage in the non-pipelined DBM design for NanGate-45 nm technology at 100MHz

Stage		Critical binary path delay (ns)	Critical decimal path delay (ns)
Multiples Generation		0.160 ns	0.102 ns
Partial Products Selection		1.385 ns	1.442 ns
Partial Products Accumulation	**Binary column tree**	1.198 ns	1.198 ns
	Split Binary part	0.329 ns	-
	Split Decimal part	-	1.491 ns
Final CPA	**Final Binary CPA**	1.133 ns	-
	Final Decimal CPA	-	1.079 ns

Table 3.6 Comparison among different pipelining schemes for NanGate 45nm technology at 100MHz

Non-pipelined Designs		Area (mm^2)		Power (mW)	Min. path delay (ns)	Average PDP (pJ)	Throughput (M operation per second)
Vázquez et al. [46]		41.14		10.134	6.056	61.37	165
Hickmann et al. [49]	Bin.	46.91		12.604	4.05	60.06	246
	Dec.				5.48		182
non-pipelined DBM [50]	Bin.	29.59		7.267	4.313	35.36	231
	Dec.				5.419		184
Pipelined Designs		**Area (mm2)**		**Power (mW)**	**slowest pipeline stage delay (ns)**	**PDP (pJ)**	**Throughput**
		Comb.	**Seq.**				
without clock gating	Scheme A	29.36	3.52	5.942	5.273	31.33	189
	Scheme B	29.56	5.30	6.08	3.861	23.48	259
	Scheme C	29.71	6.62	6.358	3.833	24.37	260
	Scheme D	29.92	16.52	7.27	3.091	22.47	323
after clock gating	Scheme A	29.85	3.53	5.38	5.273	28.37	189
	Scheme B	30.05	5.32	5.197	3.882	20.18	247
	Scheme C	30.16	6.64	5.482	3.833	21.01	260
	Scheme D	30.41	16.63	6.252	3.091	19.33	323

3.5. Pipelined DBM Design Simulation Results For Different Technologies

The pipelined DBM design with two Binary stages and two Decimal stages (scheme B) provides best tradeoff compared to the other multipliers design. On the other hand, the pipelined DBM design is synthesized at different technology nodes. Area, power, and energy of the design at 100 MHz are shown in Table 3.7. By comparing it with results in Table 3.3, the area is a little higher than that of the non-pipelined DBM design. However, the pipelined DBM is smaller than other previously published non-pipelined designs with regards to area. The total power dissipation of the pipelined DBM design for TSMC 65nm, NanGate 45nm, and NanGate 15nm technologies are lower than that of the non-pipelined DBM design by 30%, 28%, and 27%, respectively. Nevertheless, the pipelined design can operate at a higher frequency. So, the energy of the pipelined DBM is around 31%, 43%, and 44% less than that of the non-pipelined DBM design for TSMC 65nm, NanGate 45nm, and NanGate 15nm technologies, respectively. The pipelined DBM allows the use of higher frequencies, up to 500 MHz and 6 GHz, with energy consumptions of 78 pJ and 19 pJ, for NanGate 45nm and NanGate 15nm technologies respectively. The breakdown of power dissipation analysis of the DBM designs in terms of combinational and sequential components for NanGate 45nm technology at 100 MHz is shown in Table 3.8. The pipelined DBM design without clock gating is included to differentiate between pipelining and clock gating effect in the DBM design. Total leakage power of the pipelined designs is slightly increased as the area is higher due to added registers. Nevertheless, pipelining and clock gating in partial products accumulation stage reduce the total dynamic power dissipation.

Table 3.7 Area, power, and PDP for pipelined DBM design at 100MHz for different technologies

	TSMC 65nm	**NanGate 45nm**	**NanGate 15nm**
Area (mm^2)	46.77	35.37	9.60
Power dissipation (mW)	7.84	5.197	1.384
PDP (pJ)	55.469	20.18	0.523

Table 3.8 Detailed power dissipation for pipelined DBM design for NanGate 45nm at 100 MHz

	Internal power (mW)			**Switching power (mW)**			**Leakage power (uW)**		
	Comb.	**Seq.**	**Total**	**Comb.**	**Seq.**	**Total**	**Comb.**	**Seq.**	**Total**
non-pipelined DBM [50]	2.774	-	2.774	3.766	-	3.766	728	-	728
pipelined DBM (without clk-gating)	2.436	0.308	2.744	2.408	0.112	2.520	786	31	817
pipelined DBM	2.107	0.213	2.320	2.00	0.053	2.053	786	31	817

3.6. Power Distribution Analysis

A power comparison among multiplier components is presented in Figure 3.1. Multiples generation and partial products selection stages have similar power dissipation for Vázquez et al., and Hickmann et al. designs. Those of the DBM designs consume less power due to the fast generation of 8A and 9A. Hickmann et al. design has the largest power dissipation in the accumulation stage due to duplicated addition stages caused by split binary and decimal paths. Besides, the use of binary column tree in the DBM designs' accumulation stage decreases the power dissipation significantly. Also, clock gating in pipelined design causes more power reduction. Finally, a study of the power distribution among power consumption components for all designs at different technology nodes is shown in Table 3.9 at 100MHz frequency. Leakage power is dissipated when design cells are turned off, switching power is the power consumed when cells inputs and outputs are switching, and internal power is cells power dissipation. As technology size decreases, leakage power to dynamic power percentage increases. This is observed in simulation results. Although the pipelined DBM design has a small area increment than the non-pipelined DBM design causing a small increment in leakage power. Also, the pipelined DBM has the lowest dynamic power for all used technologies.

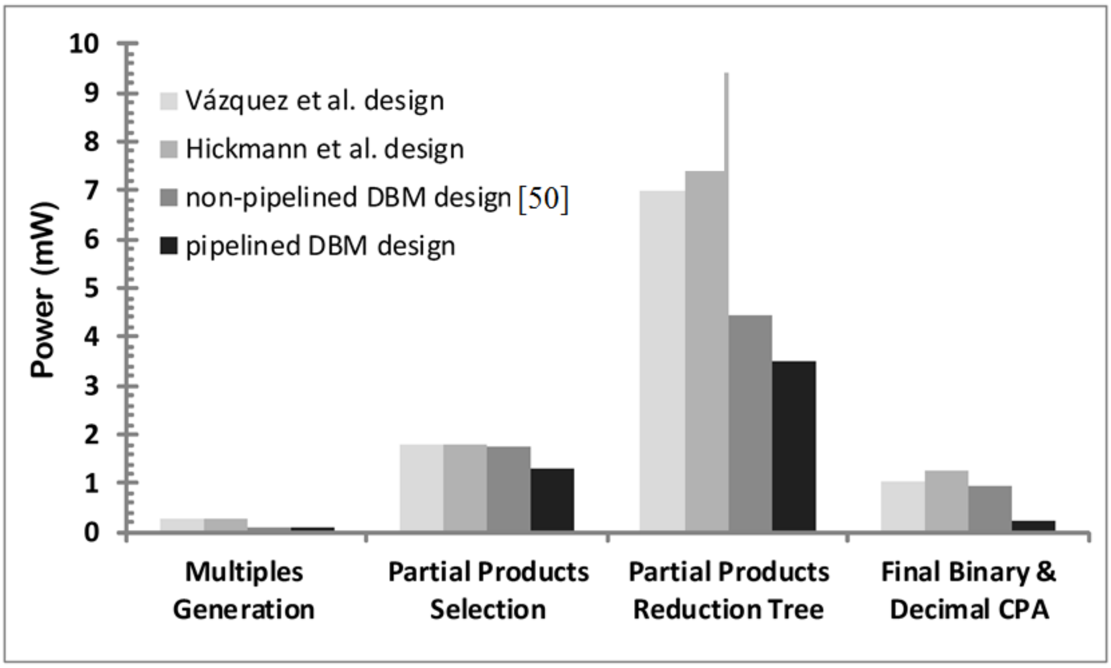


Figure 3.1 Total power breakdown in terms of the structure for 45nm technology at 100 MHz

Table 3.9 Power distribution of the pipelined DBM at 100 MHz for different technologies

		Internal Power (mW)	**Switching Power (mW)**	**Leakage Power**
TSMC 65nm	Vázquez et al. design [46]	14.58	5.11	0.83 uW
	Hickmann et al. design [49]	15.75	6.44	0.93 uW
	non-pipelined DBM design [50]	8.17	3.14	0.64 uW
	pipelined DBM design	6.47	1.37	0.83 uW
NanGate 45nm	Vázquez et al. design [46]	3.78	5.34	1.01 mW
	Hickmann et al. design [49]	4.76	6.71	1.14 mW
	non-pipelined DBM design [50]	2.77	3.766	0.73 mW
	pipelined DBM design	2.52	2.193	0.82 mW
NanGate 15nm	Vázquez et al. design [46]	1.14	1.02	0.47 mW
	Hickmann et al. design [49]	1.33	1.35	0.54 mW
	non-pipelined DBM design [50]	0.82	0.75	0.35 mW
	pipelined DBM design	0.6	0.39	0.40 mW

3.7. Summary

This chapter introduces comparisons and simulation results for the different suggested pipelined binary/decimal multiplier designs and non-pipelined binary/decimal multiplier designs in literature. Firstly the designs are tested using FPGA then simulated for ASIC in Synopsis Design Compiler with TSMC 65nm low power CMOS standard cell library, NanGate 45nm and NanGate 15nm libraries. The performance of area, delay, power, and energy are studied for the different designs. The suggested low energy clock-gated pipelined DBM multiplier shows high reductions in area, power, and energy consumption with acceptable latency. It provides 43% energy reduction if compared to the lowest energy non-pipelined design in the literature.

Chapter 4 : Low Energy Design for Main Memory (Literature in Data Compression)

4.1. Introduction

4.1.1. Memory Hierarchy

Recent features in computer system technology, design, and architecture, such as parallelism and multithreading, increase the computer performance. Nevertheless, more data bandwidth and capacity are required. Memory bandwidth can be defined as the rate of information read/write from/into memory by the processor; and memory capacity as the amount of information that is held by the memory. Therefore, system memory performance and energy became significant bottlenecks in computer systems framework [57]. Data temporal and spatial locality are the main factors behind modern memory hierarchy system. Temporal locality supposes that when memory data is referenced, it will be referenced again shortly. Spatial locality presumes that when an address is referenced, data whose addresses are close by will be referenced soon [58]. Accordingly, memory hierarchy and levels are decided according to technology, speed, capacity, and cost. The primary technologies used today in memory hierarchies are shown in Table 4.1 [58]. SRAM (static random access memory) technology is the fastest memory technology and is used closer to the processor in the cache level memory. Nevertheless, it is the most expensive and is highest in energy consumption. The main memory uses DRAM (dynamic random access memory) technology which has larger capacity for same silicon area with lower speed. Secondary storage is the largest level in the hierarchy, so lower-cost and slower technologies are used, such as Magnetic disk. Flash memory is a nonvolatile memory that is used as the secondary memory in personal mobile devices.

Table 4.1 Memory technologies' at 2012 [58]

Memory technology	Typical access time (ns)	Price/GB at 2012
SRAM semiconductor (Cache Memory)	0.5 – 2.5	\$500 - \$1000
DRAM semiconductor (Main Memory)	50 – 70	\$10 - \$20
Flash semiconductor	5,000 – 50,000	\$0.75 - \$1.00
Magnetic disk (Hard Disk Drive)	5,000,000 – 20,000,000	\$0.05 - \$0.10

SRAM technology is a memory arrays that typically consists of six transistors per bit to avoid disturbance of information when read. It has a fixed read/write access time although the read and write access times may differ. SRAM's access time is very close to the clock cycle time and it needs one clock cycle for read/write as it doesn't need to be refreshed. Nowadays, PCs and server systems' cache levels, last level cache (LLC), level 2 (L2), and level 3 (L3) are integrated onto the processor chip unlike where the past, they used separate SRAM chips for each level [58]. SRAM cells are volatile but as long as power is connected, the cell values are kept indefinitely. DRAM memory consists of an array of charge storage cells; that contains one capacitor and a transistor for each bit. The data is stored as a charge in a capacitor. DRAMs are cheaper and much denser per bit than SRAM but they need to be refreshed regularly to recharge the capacitor. The refresh process is done by just read the cell contents and write them back in it; this process is repeated every several milliseconds. Actually, DRAMs use two-level decoding structure which allows refreshing an entire row, word line, at one time. Refreshing process of DRAM is the reason for calling it dynamic, as opposite to the SRAM static storage. Besides, flash Memory technology is a mix of erasable programmable read-only memory (EPROM) and electrically erasable programmable read-only memory (EEPROM) technologies. As flash memory technology can erase a large amount of memory at one time, it called flash; as opposite to EEPROM that erases each byte individually. The flash cell typically consists of one floating gate transistor like EPROM cell; whereas, EEPROM has two transistors per cell. Nevertheless, they are different in geometry, and materials density that allows the flash memory to be programmed and erased electrically [59]. Furthermore, most flash memories use wear leveling technique that distributes writing on all its blocks to be wearing evenly which avoid writing frequently on the same cells. Wear leveling increases the memory life time of the flash memory. However, it needs a controller to spread the writes in the memory blocks and higher level software to monitor the blocks wear. Hard disk drive (HDD) is an electromechanical technology that store data in magnetic materials similar to the cassette or videotape materials. HDD consists of a set of metal platters, covered with magnetic material on both sides, rotate on a spindle. Above each platter surface, a read/write head is placed which is a movable arm containing a small electromagnetic coil. HDD technology is less expensive than other technologies but its access time is very high because it has a mechanical rotations. [58]

Lately, secondary storage is implemented using NAND flash memory-based solid state drives (SSD) which are still much faster than HDDs technology. While SSD has less area and extra cost, SSD is a semiconductor based technology which makes a significant performance improvement over the rotational HDDs [60]. For recent technologies, DRAM access times are about 10 ns and secondary storage SSD latencies are about 10 μs [60] [61] [62]. Figure 4.1 shows the structure of a memory hierarchy according to capacity and closeness to the processor. Data accesses that hit in the highest level of the hierarchy can be processed quickly due to faster memory. Data accesses that miss go to the lower levels of the hierarchy, which are slower. [58]

Nowadays more cores are integrated onto the same chip so more applications are run concurrently and applications become more data intensive which make memory compression an urgent need. Memory compression grows the capacity of a memory system, reduces page fault, and reduces energy and bandwidth demands. The main memory bandwidth has been a critical shared resource for multiprocessors chip. Most prior techniques have been designed to focus on the capacity metric which has complex logic and few prior works focus on reducing energy or bandwidth. Nowadays, they focus on energy and bandwidth [63]. Compression is utilized to increase cache and main memory capacity, where large capacity memory systems constrain energy and bandwidth [64]. Main memory compression allows more data to be stored near the processor and reduces the number of accesses to the secondary storage or flash memory for portable devices.

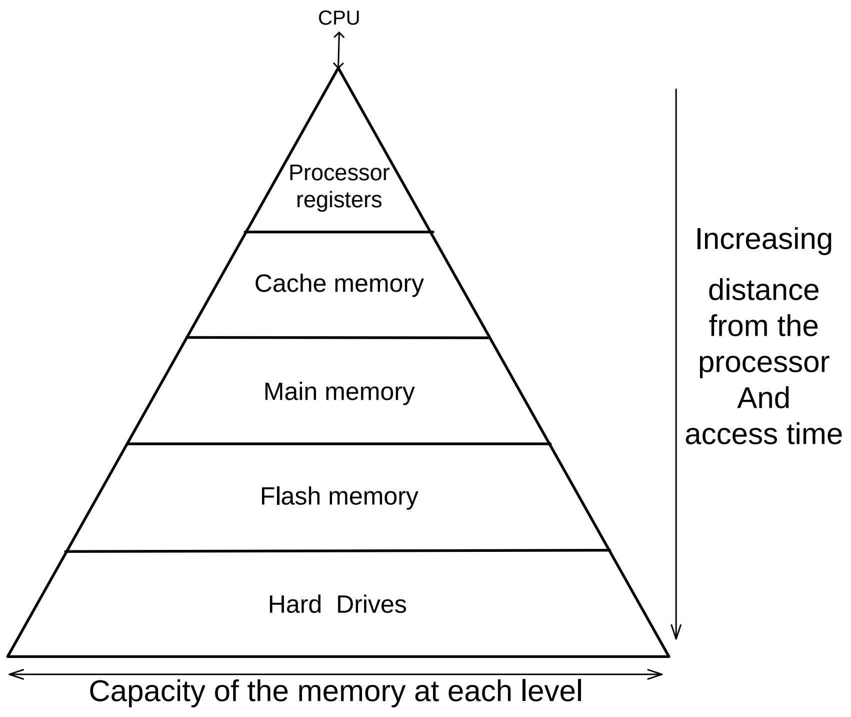

CPU
Processor registers
Cache memory
Main memory
Flash memory
Hard Drives
Increasing distance from the processor And access time
Capacity of the memory at each level

Figure 4.1 Levels in memory storage hierarchy [58]

4.1.2. Lossless Compression

Information theory studies the quantification, storage, and communication of information. It is the basis of many techniques used in data compression. Quantifying information measure information that is included in a piece of data [65]. The toss of a coin is a simple event that can easily be used to study the meaning of information in binary representation. It has two results, thus the result of any toss is initially uncertain. When the coin is thrown, the result is head or tail, yes or no, or 1 or 0; so its uncertainty can be solved in one bit. This example can easily be generalized for binary representation of information. Many real-life problems' solutions can be expressed in the form of several bits; the problem is to find the minimum number of bits. As one bit can express the answer of a yes/no question, the number of bits that express any problem's result are equivalent to the minimum number of yes/no questions that must be answered to reach this solution. Another example of a problem is deck of 64 playing cards and person A hold one card and person B have to guess it. The cards can be numbered 1 to 64, for simplicity. The minimum number of yes/no questions that are needed to guess the card are the number of bits to express the problem's solution. The cards numbers 1 to 64 should be divided into two intervals 1-32 and 33-64. The first question is "is the solution in the first interval 1-32?". If the answer is no, the solution is in the interval 33-64. Then the interval 33-64 is divided into two sub-intervals and the next question is "is the solution in the interval 33-48?", and so on until the sub-interval reduces to a single number. This technique is called binary search. The number of required questions is exactly six to reach the solution. Mathematically, the answer is equal to $log_2 64$. That is why the logarithm is the mathematical function that quantifies information [65]. Similarly, the information contained in a deck of 52 playing cards is $log_2 52 = 5.7$.

For decimal representation, the problem is to compute the number of digits to express a positive integer N. With N number increase, more digits are needed. Two decimal digits can represent the first 100 positive integers (0-99) and three digits can represent the first 1000 numbers, and so on. The number of digits required to represent positive integer N number are approximately $\log N$. Where, the base of the logarithm is 10. Accordingly, the information content in N numbers is proportional to the number of digits it takes to express N, and the function that can measure this information is the logarithm with base equal to the base of the used digits [65]. And it is noticeable that as the base of the used digits increase, one digit can hold more information. And it can be expressed that the information included in one base-n digit equals that included in $\log_2 n$ bits [65] (i.e. information included in one decimal digit = information included in $\log_2 10 = 3.322$ bits).

In this thesis, lossless compression is chosen for main memory data as we can't lose memory data. Lossless compression allows us to save the data in less space without cost any loss of information. The target is to reach the minimum number of bits to store the information included in a set of data. The three major compression factors and tradeoffs considered in this work for the memory data are hardware complexity, compression ratio, and compression/decompression latencies; specially decompression where it is in the critical path of the processor execution time of instructions.

4.2. Literature Review on Lossless Compression Algorithms

4.2.1. Definitions

Data compression is usually called source coding where coding in signal processing seek to encode a data message to transmit the same information using fewer bits. Coding can be defined as; the process of generating a binary representation for data elements to store it in less storage space than the original data. Finding the optimum way to compress data is one of the challenging problems in the field of source coding [66][67]. Probabilities are used in compression algorithms to choose the suitable code lengths for each data symbol. These probabilities are derived from the data message, from similar messages, or upon human experience and knowledge.

Assume data of n symbols $S = [s_1, s_2, \ldots s_n]$, and its corresponding probability estimates $P = [p_1, p_2, \ldots p_n]$. After coding, we have a source code codewords $C = [c_1, c_2, \ldots c_n]$ and the length of codewords, number of bits in each codeword, $L(C) = [l_1, l_2, \ldots, l_N]$. The length of codeword is sometimes called cost. Table 4.2 shows three different source codes and their expected length [68]; the coding problem is $n = 6$, $S = [1, 2, 3, 4, 5, 6]$, $P = [0.67, 0.11, 0.08, 0.07, 0.04, 0.03]$. C_1 is the simple binary code. C_3 codewords are ["0", "100", "101", "110", "1110", "1111"]. Its codewords is chosen with lengths $L(C_3) = [1, 3, 3, 3, 4, 4]$ to gain from the different probabilities of occurrence of symbols. A code with a small number of bits is defined to the most frequent symbol. The details of different source codes will be discussed later; the following are some important definitions

Minimum-Redundancy Coding; it is a set of codeword lengths which best matches the probability distribution of the data symbols thus gives the minimum code length for certain data message.

Prefix-free coding; it refers to the coding techniques which ensures that each codeword is not a prefix of any other codeword of the data symbols codes.

Expected length or Expected cost; the average number of bits per source data symbol. The expected length $L(C)$ of a source code C with probabilities p_i and codeword lengths l_i is defined as

$$L(C) = \sum_{i=1}^{n} p_i l_i \tag{4.1}$$

Table 4.2 Three simple codes and their expected length

s_i	p_i	Code 1 C_1	Code 2 C_2	Code 3 C_3
1	0.67	000	00	0
2	0.11	001	01	100
3	0.08	010	100	101
4	0.07	011	101	110
5	0.04	100	110	1110
6	0.03	101	111	1111
Expected length (bits/symbol)		3.00	2.22	1.73

Kraft-McMillan inequality; It concerns the relation between the codewords c_i and their lengths l_i to establish a unique prefix-free code. Kraft at 1949 observes that if the probability of data symbols is negative power of two, $p_i = 2^{-k_i}$ where k_i is an integer number. So setting each codeword c_i to its corresponding k_i bits, $l_i = k_i$, results in a minimum-redundancy coding and the code should be a prefix-free [69]. As the summation of the supposed code symbols probabilities should be less than or equal 1

$$\sum_{i=1}^{n} p_i \leq 1 \tag{4.2}$$

So **Kraft inequality** says that if the quantity

$$K(C) = \sum_{i=1}^{n} 2^{-l_i} > 1 \tag{4.3}$$

then the code can't be prefix-free. For example, assume that n codewords are all of length $l_i = 1$, then $K(C) = n/2$, and a prefix-free code is only possible when $n \leq 2$.

Based on this work, McMillan at 1956 showed that if $K(C) \leq 1$ for code C, then there always exists another code C' which is a prefix-free with the same cost of C and is uniquely decodable in a left-to-right manner. [69]

4.2.2. Basic Coding Techniques

4.2.2.1. Unary Coding

The data symbols are numbered as in Table 4.2 first column then each symbol s_i is represented as $s_i - 1$ of ‘1’ bits, followed by a single ‘0’ bit.

4.2.2.2. Binary Coding

In Table 4.2, C_1 is the standard binary representation using $\lceil log_2 n \rceil$ for the number of bits of codewords, which equal three in this example. A shorter codewords, equal to $\lfloor log_2 n \rfloor$, can be assigned to the frequent symbols as in C_2 with the prefix-free property. So the total expected length is reduced. Generally, it can be assigned if $log_2 n$ is not an integer number. **Minimal binary code** contains $(2^{\lceil log_2 n \rceil} - n)$ codewords that are $\lfloor log_2 n \rfloor$ bits long, and the remaining $2n - 2^{\lceil log_2 n \rceil}$ are $\lceil log_2 n \rceil$ bits long [68]. Another example of minimal binary coding is shown in Table 4.3. However, the standard binary coding representation is also a prefix-free.

4.2.2.3. Codes with Selector Part

It consists of a $selector$ part that indicates a range of values that compose a bucket, b, and a binary part that indicates a precise value within the specified bucket, Table 4.3.

Elias Coding, called also "Exponential and Binary Search" [1976]

The second code in Table 4.3 is C_γ. Its codewords consist of two portions. The first portion is a unary code for the binary magnitude of s_i, which takes 1 + $\lfloor \log_2 s_i \rfloor$ bits. The second part of each codeword is a simple binary code within the range of $\lfloor \log_2 s_i \rfloor$ bits. Thus the total codeword length is $1 + 2\lfloor \log_2 s_i \rfloor$ bits. This algorithm was firstly described by Bentley and Yao [1976]. The codewords of the second Elias code C_δ is also divided into two portions. The first portion is coded using C_γ instead of unary code, and the codeword length is equal $1 + 2\lfloor log_2 log_2(2s_i) \rfloor + \lfloor \log_2 s_i \rfloor$ bits. [68]

For small number of symbols, Elias coding appear to have high cost. Nevertheless, for large number of symbols, both Elias codes are exponentially better than unary coding where their bucket sizes grow exponentially, $b = (1, 2, 4, 8, \ldots, 2k, \ldots)$.

Table 4.3 minimal binary, Elias, Golomb, and Rice codes
(i.e. the blanks in the codewords do not appear in the coded bit stream)

s_i	p_i	Minimal binary code	Elias C_γ code	Elias C_δ code	Golomb code, $b=5$	Rice code, $k=2$
1	0.32	000	0	0	0 00	0 00
2	0.21	001	10 0	100 0	0 01	0 01
3	0.11	010	10 1	100 1	0 10	0 10
4	0.1	011	110 00	101 00	0 110	0 11
5	0.09	100	110 01	101 01	0 111	10 00
6	0.07	101	110 10	101 10	10 00	10 01
7	0.05	110	110 11	101 11	10 01	10 10
8	0.03	111 0	1110 000	11000 000	10 10	10 11
9	0.02	111 1	1110 001	11000 001	10 110	110 00
Expected length (bits/symbol)		3.05	3.18	3.55	3.38	3.28

Golomb Coding [1966]

It also consists of two portions with a fixed parameter b for the bucket size. The first portion is a unary code where each bucket, b, symbols has new unary code. The second portion is a minimal binary code where its short codewords is assigned to the least values. A division operation used in the encoder to generate the integer quotient of (n/b). The first portion of the codewords length is $\lceil(s_i/\mathrm{b})\rceil$ and the second portion length is equal to minimal binary coding.

Rice Coding [1979]

It is a special case of Golomb code. Rice coding designate the parameter b equal to 2^k where k is an integer value to reduce the design complexity. The value s_i is shifted right k bits to get the value of the unary code part, then the low-order k bits of the s_i are coded as a k-bit binary value.

4.2.2.4. Run Length Encoding (RLE) [1967]

Its main idea is to replace n consecutive occurrences of data symbol s_i with the $\acute{r}ns_i$. Where ŕ is a reserved value to indicate that the following two values is for repeated symbols. Only runs longer than three characters get compressed as the compression take three bytes; reserved value, number of repetition of the symbol, and the symbol. RLE is supported by some bitmap file formats, such as TIFF, and BMP. The advantage of RLE lies on implementation simplicity and low latency.

4.2.3. Statistical Coding Techniques

Statistical compression has two main operations: modeling, and coding. Modeling phase is the process of learning about the structure of the data being compressed to generate a statistical model. Coding is used to map data to the compressed form. Statistical coding also has variable-length codewords to assign shorter codewords to symbols that have higher probability of occurrence. In the next sections, statistical compression algorithms are presented, such as Shannon-Fano, Huffman, and arithmetic coding.

4.2.3.1. Shannon-Fano Coding [1948]

It is independently discovered by Claude Shannon in 1948 and Robert Fano in 1949, and is known as Shannon-Fano coding [68]. The list of symbol probabilities are divided into two parts, with probability as close to 0.5 as possible to each part. Each part takes one bit prefix bit, '0' or '1', as the first bit of their codewords. Then each part is sub-divided into subparts of probability of weights that as close as possible to half of its parent part weight. Code 3 in Table 4.2 is a prefix-free Shannon-Fano coding that reduces the codeword length to 1.73 bits per symbol. However, Shannon-Fano algorithm is not always effective as it not always gives the minimum redundancy codewords. That is the results of its top-down structure scheme. For example, if the probability distribution of data symbols $P = [0.35, 0.15, 0.1, 0.1, 0.1, 0.1, 0.1]$, the Shannon-Fano codewords $C = [\text{“}00\text{”}, \text{”}01\text{”}, \text{”}100\text{”}, \text{”}101\text{”}, \text{”}110\text{”}, \text{”}1110\text{”}, \text{"}1111\text{"}]$ with lengths $L(C) = [2, 2, 3, 3, 3, 4, 4]$ and 2.7 bits/symbol for the expected length. This approach assigns a 2-bit codeword to the second symbol, while it should have a codeword of the same length as symbols 3, 4, 5, 6, and 7.

4.2.3.2. Huffman Coding [1952]

A bottom-up mechanism is employed rather than Shannon-Fano top-down algorithm. It sorts the symbols according to their probability distribution in descending order. Then at each step of the algorithm, the last two symbols take one bit prefix bit, '0' and '1', as the first bit of their codewords and the two symbols probabilities are combined and the symbols probabilities are reordered. Table 4.4 shows an example of Huffman coding steps for $P = [0.35, 0.15, 0.1, 0.1, 0.1, 0.1, 0.1]$. The length of the generated codewords is $L(C) = [2, 3, 3, 3, 3, 3, 3]$ and the expected length is 2.65. The Huffman scheme is simple, and gives the best codes for data symbols when the probabilities of the different symbols making up the message are known. However, it produces ideal variable-length codes when the symbols probabilities are negative powers of two such as $1/2$, $1/4$, or $1/8$. [65]

Afterward, several minimum redundancy codes are performed depending on the idea of Huffman coding. Canonical Huffman coding was introduced by Schwartz and Kallick in 1964, Connell in 1973, Hirschberg and Lelewer in 1990, Zobel and Moffat in 1995, and Moffat and Turpin in 1997 [68]. The canonical Huffman codes are a specific type of Huffman code that store codewords efficiently by generating “normal” Huffman codewords lengths and then change codewords preserving their lengths. The objective is to find codes that are simple, and fast to be encoded and decoded with the same expected length/cost of codewords.

Table 4.4 Huffman coding example

P	initial code
0.35	$c_1 = \lambda$
0.15	$c_2 = \lambda$
0.1	$c_3 = \lambda$
0.1	$c_4 = \lambda$
0.1	$c_5 = \lambda$
0.1	$c_6 = \lambda$
0.1	$c_7 = \lambda$
initial stage	

P	initial code
0.35	$c_1 = \lambda$
0.2	$c_6 = 0,$ $c_7 = 1$
0.15	$c_2 = \lambda$
0.1	$c_3 = \lambda$
0.1	$c_4 = \lambda$
0.1	$c_5 = \lambda$
First step	

P	initial code
0.35	$c_1 = \lambda$
0.2	$c_6 = 0,$ $c_7 = 1$
0.2	$c_4 = 0,$ $c_5 = 1$
0.15	$c_2 = \lambda$
0.1	$c_3 = \lambda$
Second step	

P	initial code
0.35	$c_1 = \lambda$
0.25	$c_2 = 0,$ $c_3 = 1$
0.2	$c_6 = 0,$ $c_7 = 1$
0.2	$c_4 = 0,$ $c_5 = 1$
Third step	

P	initial code
0.4	$c_6 = 00,$ $c_7 = 01,$ $c_4 = 10,$ $c_5 = 11$
0.35	$c_1 = \lambda$
0.25	$c_2 = 0,$ $c_3 = 1$
Fourth step	

P	initial code
0.6	$c_1 = 0$ $c_2 = 10,$ $c_3 = 11$
0.4	$c_6 = 00,$ $c_7 = 01,$ $c_4 = 10,$ $c_5 = 11$
Fifth step	

P	initial code
1	$c_1 = 00$ $c_2 = 010,$ $c_3 = 011$ $c_6 = 100,$ $c_7 = 101,$ $c_4 = 110,$ $c_5 = 111$
Sixth step	

Nowadays, Huffman coding is widely used with the mainstream compression formats such as ZIP. And sometimes it is used with other compression techniques; multimedia codecs such as JPEG and MP3 use a front-end model and quantization followed by Huffman coding.

4.2.3.3. Arithmetic Coding

It is firstly proposed in 1960s by Peter Elias [65]. Arithmetic coding gives one codeword (typically long) to the input message. It encodes more symbols using only one codeword of fixed length. It assigns a certain initial interval to the input message then read it symbol by symbol and narrow the interval according to the probability of symbols. The number of bits that represent the interval is larger as the interval is narrower, so the higher probability symbols narrow the interval less than the lower probability ones. The interval is specified by lower and upper limits or by lower limit and width. The initial interval is [0, 1), which means a range of real numbers from 0 to 1 including 0 but not including 1. However, the arithmetic coding output is a real number in the range [0, 1) although the 0. part is not included; the number 0.97654 is encoded to 97654.

After the initial interval is set to [0, 1), the interval is divided into subintervals whose sizes are proportional to the symbols' probabilities, then the message is read symbol by symbol. However, the order of symbols according to their probability distribution are ascending or descending. At each step, new symbol is processed and the current interval gets smaller so it takes more bits to express it. The output is a unique number that identify the input message. Table 4.5 shows an example of arithmetic coding. The input message is $a_2a_3a_1a_4a_1a_5a_2a_6a_1a_7$. Table 4.5(a) shows the symbol's probability of occurrence according to input message. To encode the message, the initial interval is set to $[0, 1)$. The first symbol a_2 reduces this interval to the subinterval from its 50% point to its 70% point. It gets the new interval [0.5,0.7) of size 0.2. For the second symbol a_3, the new interval is $[0.58, 0.64)$, it is calculated by $0.5 + (0.7 - 0.5) \times 0.4 = 0.58$ and $0.5 + (0.7 - 0.5) \times 0.7 = 0.64$. So it reduces the interval from size 0.2 to size 0.06. The final code of this method can be any number from the final range. The Low and High values of the interval [NewLow, NewHigh) is calculated using the following equations

$$\mathrm{NewLow} = \mathrm{OldLow} + \mathrm{old\_interval\_Range} * \mathrm{LowRange}(\mathrm{s_i}) \tag{4.4}$$

$$\mathrm{NewHigh} = \mathrm{OldLow} + \mathrm{old\_interval\_Range} * \mathrm{HighRange}(\mathrm{s_i}) \tag{4.5}$$

where $old\_interval\_Range = OldHigh - OldLow$,
$LowRange(s_i)$, and $HighRange(s_i)$ indicate the low and high limits of the subinterval of the symbol s_i, respectively.

The output stream consists of the symbols and its frequencies (probabilities) then the bits of the compressed code. After coding all the input data message $a_2a_3a_1a_4a_1a_5a_2a_6a_1a_7$, the final code is a number in the interval [0.628796836, 0.6287968684). We can choose 62879684, $(11101111110111011111000100)_b$. The average length of codeword here is 2.6 instead of 2.7, and 2.65 for Shannon-Fano, and Huffman coding, respectively. As the length of output code is 26-bits for 10 symbols.

Table 4.5 Example of arithmetic coding for message of 10 symbols (a) statistical model of the data, (b) coding of the input message

s_i	p_i	symbol's subintervals
a_1	0.3	[0.7, 1)
a_2	0.2	[0.5, 0.7)
a_3	0.1	[0.4, 0.5)
a_4	0.1	[0.3, 0.4)
a_5	0.1	[0.2, 0.3)
a_6	0.1	[0.1, 0.2)
a_7	0.1	[0, 0.1)

(a)

Message	Current interval
Initial interval	$[0, 1)$
a_2	$[0.5, 0.7)$
a_3	$[0.58, 0.64)$
a_1	$[0.622, 0.64)$
a_4	$[0.6274, 0.6292)$
a_1	$[0.62866, 0.6292)$
a_5	$[0.628768, 0.628822)$
a_2a_6	$[0.62879608, 0.62879716)$
a_1a_7	$[0.628796836, 0.6287968684)$

(b)

4.2.4. Dictionary Based Coding Techniques

Using statistical model for data coding makes the quality of the model affect the quality of the compression. Dictionary based coding approaches choose strings of symbols and encode each one as used in a dictionary. The statistical coding is permanent, sometimes allowing the addition of strings but no removals, while the dictionary based coding takes strings previously found in the input stream, permitting for additions and removals of strings while new data is being read.

4.2.4.1. LZ77 and LZ78 [Lempel and Ziv, 1977 and 1978]

LZ77 [70] and LZ78 [71] are also known as LZ1 and LZ2 respectively. It knows the used characters by certain data stream and generates new coding for it depending on number of used characters. Every data stream has its dictionary based on the data stream. It moves on all input data to build the dictionary codes that will be compressed. It uses fixed length code for input patterns. After generating the dictionary model for the input data, compression and decompression is done according to look up tables, LUTs, with buffer move through all bytes and a comparator to match dictionary entries. Compression ratio depend on input data, large number of used characters will give bad compression ratio and vice versa. Its delay is proportional to input data, large number of bytes gives high latency. LZ77 maintains a sliding window during compression. LZ78 decompression allows random access to the input as long as the entire dictionary, while LZ77 decompression must always start at the beginning of the input. These two algorithms form the basis for many variations including LZW, LZMA, LZS, LZSS, and others. These algorithms formed the basis of several worldwide compression schemes, including GIF and the DEFLATE algorithm used in PNG.

4.2.5. Differential Coding

It is a different approach results in a significantly less complex system. It encodes the first data element of the source code, and then encodes the differences between the next source elements and the first element. It is used for sources such as speech and images which have a correlation between its samples. [67]
For example, the source code $[6.2, 9.7, 13.2, 5.9, 8, 7.4, 4.2, 1.8]$ is coded into $[6.2, 3.5, 7, -0.3\ , 1.8, 1.2, -2, -4.4]$. The original code is recovered by adding the difference values to the first element. One of the common usages of this coding is differential pulse code modulation (DPCM) and delta modulation. While differential coding does not provide high compression as some other coding techniques, it is very simple to implement. It is recommended for speech and image applications.

4.2.6. Conclusion

Binary, minimal binary, and run length coding are independent on input data. Minimal binary coding is like binary coding but save some bits when the number of data symbols is not in a power of two, 2^k. Run length coding is used with other coding approaches, after or before another coding, due to its simplicity and good compression depending on repeated patterns of the input data.

Table 4.6 shows a comparison between the different basic coding techniques with an example of 60 symbols where comparing numbers is easier than equations. Both Golomb and Rice codes are extensively used in compression applications due to their low expected length/cost with a large number of symbols. And Rice code is preferred because it's simpler. Elias C_δ average number of bits is higher than Elias C_γ at equal probability distribution of input symbols, but when the probabilities of some symbols are higher than the others, the cost of C_δ will be less. Generally, the choice of the coding technique depends on the probability distribution of the data symbols which lead to the expected cost, and the implementation complexity of the coding technique. Statistical and dictionary based coding encode symbols in non-fixed length codewords depending on their probability of repetition in the input data. While they have high compression ratio compared to basic coding techniques, they need more processing on input data before coding, and need extra storage space. The storage space is for the statistics codes or the dictionary data to be able to recode them.

For our target here, ASIC compression/decompression design for main memory data, the choice of the compression approach depends on the memory data types and the implementation feasibility of the approach in hardware. Differential coding and basic coding approaches like unary, minimal binary, Elias, and rice coding have simple hardware implementation in contradiction of the Golomb and arithmetic coding which has division in its encoder. Despite the hardware implementation of Shannon-Fano and Huffman statistical coding is simple, the statistical coding and Run length coding should move on the input data firstly before coding which not practicable in hardware. However, run length coding is the simplest as it can be implemented by reading piece by piece of the input data. Also, dictionary based coding firstly study the input data, however, it need large LUTs in its hardware implementation.

Table 4.6 Comparison between different basic coding techniques

Codes	Maximum/last codeword length (bits)	Average codewords Length assume equal probability distribution (bits)	Complexity	Latency
Unary (n symbols)	n	$1+2+3+\cdots..+n \;=\; n(n+1)/2$	Simple	Small (subtraction by 1 delay)
(60 symbols)	60	1830		
minimal binary	$\lceil \log_2 \mathrm{n} \rceil$	$\left(2^{\lceil log_2 n \rceil} - n\right) * \lfloor \log_2 \mathrm{n} \rfloor + 2n - 2^{\lceil log_2 n \rceil} * \lceil \log_2 \mathrm{n} \rceil$	Simple	Small (like binary coding)
(60 symbols)	6	(64-60) * 5 + 56 * 6 = 356		
Elias C_γ (n symbols)	$1 + 2\lfloor \log_2 \mathrm{n} \rfloor$	$n + 2(\lfloor \log_2 1 \rfloor + \lfloor \log_2 2 \rfloor + \lfloor \log_2 3 \rfloor + \cdots)$	Simple	Small (unary code + binary coding)
(60 symbols)	11	60+2(2*1+4*2+8*3+16*4+29*5) = 546		
Elias C_δ (n symbols)	$1 + 2\lfloor \log_2 \log_2 2\mathrm{n} \rfloor + \lfloor \log_2 \mathrm{n} \rfloor$	$n + (\lfloor \log_2 1 \rfloor + \lfloor \log_2 2 \rfloor + \lfloor \log_2 3 \rfloor + \cdots) +$ $2(\lfloor \log_2 \log_2 2 \rfloor + \lfloor \log_2 \log_2 4 \rfloor + \lfloor \log_2 \log_2 6 \rfloor + \ldots)$	Simple	Medium (Elias C_γ + binary coding)
(60 symbols)	8	60+(2*1+4*2+8*3+16*4+29*5)+2(4*1+8*2+16*3+29*4) = 671		
Golomb (fixed-size b)	$\lceil (\mathrm{n}/\mathrm{b}) \rceil + \lceil \log_2 \mathrm{b} \rceil$	$(\lceil (1/7) \rceil + \lceil (2/7) \rceil + \lceil (3/7) \rceil + \cdots) + n * \lceil \log_2 \mathrm{b} \rceil$	Complex	Medium (division/ multiplication)
(60 symbols, b=7)	9 + 3 = 12	(7*1+ 7*2+7*3+7*4+7*5+7*6+7*7+7*8+4*9) +60*3 = 288		
Rice ($b=2^k$)	$\lceil (\mathrm{n}/\mathrm{b}) \rceil + \mathrm{k}$	$(\lceil (1/8) \rceil + \lceil (2/8) \rceil + \lceil (3/8) \rceil + \cdots) + n * \lceil \log_2 \mathrm{b} \rceil$	Simple	Small (unary code + shift + binary coding)
(60 symbols, k=3,b=8)	8 + 3 = 11	(8*1+ 8*2+8*3+8*4+8*5+8*6+8*7+4*8) +60*3 = 256		

Chapter 5 : Suggested Approach for Low Energy ASIC Design for Main Memory Data Compression/Decompression

5.1. Recent Literature in Memory Compression

Since applications become more data intensive, capacity and bandwidth of memory system become critical in application performance. At architecture level of abstraction, memory solutions can be classified into reference-based solutions which don't reduce the amount of data stored and value-based solutions which reduce the amount of data stored. Reference based solutions like placement and replacement solutions [72], and prefetching predict reference stream [73]. Memory compression which reduces the size of memory blocks and memory data deduplication [74] that eliminates the redundant block copies are considered value-based solutions. Memory redundancy is a result of constants, some application inputs (i.e. multiple adjacent pixels may have the same color in image processing), and some operations such as copying and assignment. Also, some applications provide worst-case scenario so it uses large size data types whereas most values could fit in smaller data types. Besides, the differences between values stored within the cache line are small in some applications so they can be represented in a compressed form using a base value and an array of differences [64].

The basic coding techniques used in compression algorithms are Huffman coding which works by analyzing the data and the elements that has high probability are coded with small number of bits. It is a variable length coding approach [65] [75]. LZ dictionary based compression algorithms work by replacing repeated occurrences of data elements with references to a single copy of that element existing earlier in the uncompressed data. It uses fixed length codes. It used in literature by Ekman and Stenstrom [76] and by IBM [77]. They will be discussed in details in next sections. Zero-content compression suggested by Dusser et al. in [78] concern null data blocks in cache memory and propose Zero-Content Augmented cache (ZCA) which consists of a conventional cache augmented with a specialized cache for null blocks, the Zero-Content cache (ZC). In the ZC cache, the data block is represented by its address tag and a validity bit and several null blocks are associated with a single address tag in the ZC cache. Its decompression is very simple. Dusser and Seznec in [79] apply null data compression in main memory and attempt to manage null blocks throughout the whole memory hierarchy. The limitation of their approach is that it cannot compress data with other patterns. Base-delta compression is used for similar data. So it can be represented using a base value and an array of differences.

IBM propose a memory expansion technology (MXT) [77] for main memory compression. They add an uncompressed large L3 cache, 32 MB, in front of the compressed memory and use LZ dictionary based compression algorithm. The critical decompression hardware is parallel implemented on special purpose ASIC circuit. The number of memory accesses are reduced but the latency of memory access is increased significantly, where the decompression block takes 64 cycles. Benini et al. [80] reduce the memory traffic using an on-the-fly variable-length compression solution, depending on delta coding. Their design depends on dividing the 16B cache line into four words, and then compresses it to at least 12B. The compressed line stores the nonzero bits of the differences between first word and the other three words as well as the first word.

The compression and decompression hardware implementations have a simple design complexity, Figure 5.1. The latency of the decompression of one compressed cache line is 4-byte addition for 16B cache line.

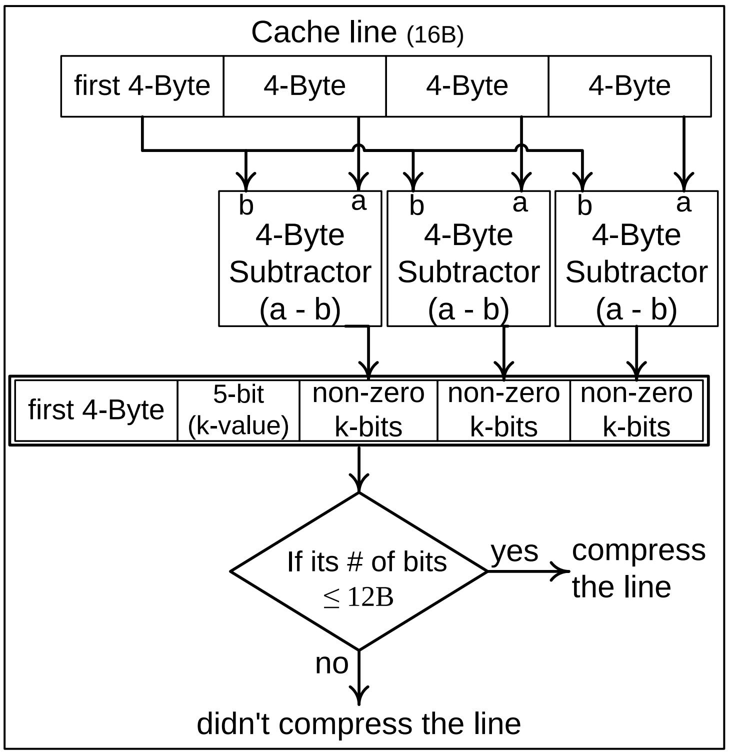


Figure 5.1 Benini et al. basic compression block diagram [91]

Ekman and Stenstrom [76] propose an approach to reduce main memory blocks fragmentation due to variable-length compression schemes. Moreover, they overcome the performance loss of MXT technique due to indirect access with a low-latency frequent-pattern compression, FPC, algorithm [81]. Besides, their approach provides a significant compression ratio with a simple zero-content compression algorithm. Nevertheless, sometimes cache miss depends on the size of compressed block. Pekhimenko et al. in [82] suggest a new cache compression algorithm depending on delta coding, base-delta-immediate compression. Figure 5.2 shows a general block diagram for "base-delta-immediate compression" Compressor Unit (CU). They observe that each cache line has low dynamic range. For each cache line, they use eight parallel compressor units; six for different base and delta sizes and two for zeros and repeated values. Then select the best one of them to output the compressed cache line, Figure 5.3. The decompression latency is similar to an integer vector addition. However, the design complexity is higher than that used in [80] due to the parallel six units. Moreover, Pekhimenko et al. [83] note that variable-length compressions provide address computation overhead which lead to more complicated address translation. They propose a memory framework that compresses the cache lines of each page to the same length. Their linearly compressed pages approach simplifies the physical address computation of the cache line to only shift operation.

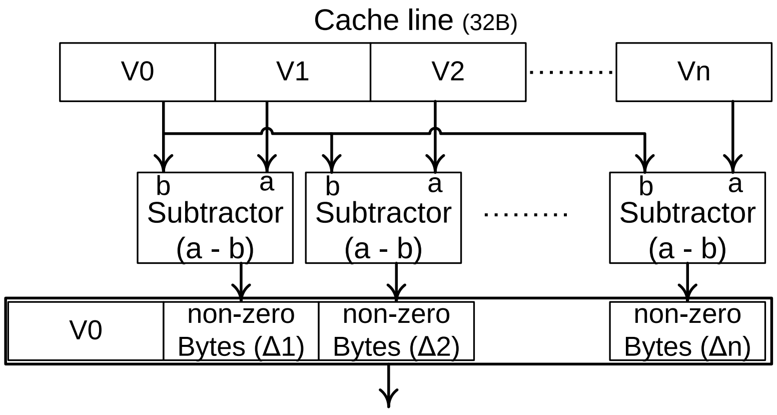


Figure 5.2 General block diagram for "base-delta-immediate compression" Compressor Unit (CU)

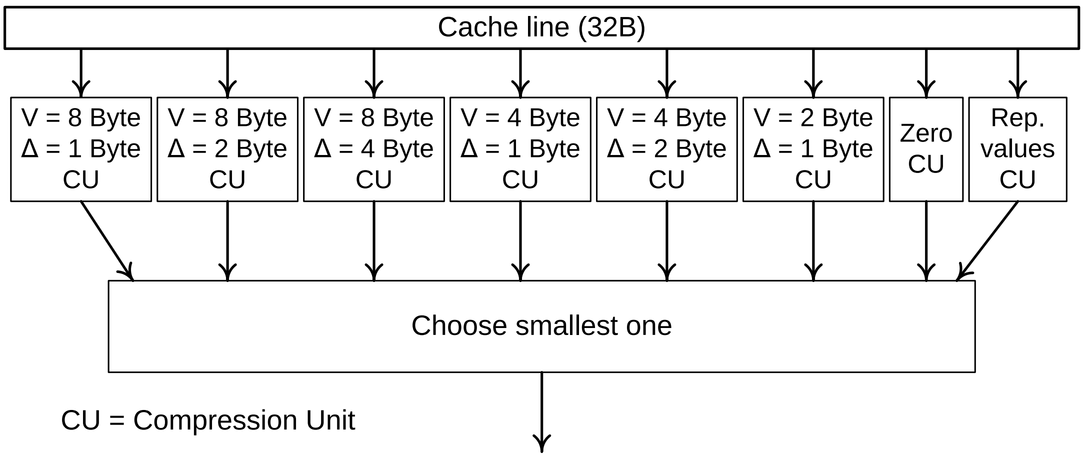


Figure 5.3 "Base-delta-immediate compression" block diagram

On the other hand, Miguel et al. in [84] and [85] observe that many similar or identical memory lines are stored in the cache which may be a redundant data and waste cache capacity. They suggest saving area and power by storing similar cache lines one time where some applications can tolerate exactness. They assume that when every element in the cache line is within a specified threshold, T, of its corresponding element in another cache line, the two cache lines are approximately similar. And they can be stored in the same cache location. Their maximum T value is 10% of each application data range of values. For example, data range from 1 to 150 gives maximum threshold value of 15. They tradeoff acceptable inexact data and energy saving with performance.

In this thesis, a lossless compression/decompression design is suggested between main memory and cache interface. The proposed approach depends on the delta coding and the observation that, for many applications, the lines of the main memory pages are mostly similar. The target is to achieve a simple low power compression design for main memory data without any approximation in data storage. The design depends on simplicity and low latency, which in turn reduces the energy consumption with proper compression ratio. The suggested design reduces decompression latency which is more critical as data pass through it when processor misses data in cache memory. Whereas, compression is necessary in returning data back from cache to main memory or getting data from secondary storage.

5.2. Suggested Methodology for Low Energy Main Memory Compression

Main memory data is periodically changed which will make statistical and dictionary based coding techniques have high overhead if used in main memory compression. They depend on the estimated probabilities of data symbols or complex arithmetic calculations as it includes multiplication or division operations. Contrary to this thesis work which has a significant constraints in compression/decompression processing time and energy. That led to choose delta coding which needn't high processing like Huffman, LZ, and arithmetic compression techniques. The target is to propose a generic low energy compression/decompression approach for main memory data. A low latency, low complexity, and fixed-length compression approach is designated for main memory compression/decompression to reduce the costly accesses of secondary storage, when page fault occur. Delta compression is chosen due to its simple architecture design and low decompression latency. In many applications, data stored in each main memory page are probably similar. In this work, two memory lines are similar within x% means they are different within $(100 - x)$%. The similarity here is defined as the difference between each and every corresponding 1-Byte elements of the two lines, $(l_i - f_i)$, are less than or equal $(2^8 * x/100 - 1)$ where i is the element index, f_i is the ith element of first line, and l_i refer to the ith element of certain line in the memory, as shown in Figure 5.4. The suggested design stores the first line, 32B, of the page then for each next line; differentiate between each byte, 8-bits, of that line and its corresponding byte in the first page line to get a set of deltas, Δs. If all the Δs of the line are less than or equal to 6-bits, the line is compressed into 24B fixed-length compressed line. Figure 5.5 shows the compression block diagram for a 32B memory line. The design consists of 32 parallel 8-bits subtractors, each one handling one byte of memory line. The compressed/uncompressed flag (C/U flag) value is the selector for the multiplexer. It selects whether the line will be compressed or not. If all 32 subtractors outputs can be stored in 6-bits each, then the C/U flag will be set to '1' and the 24B compressed line will be output. Otherwise, the line is not compressed and the 32B line will be output. An extra one bit for the C/U flag is added at the beginning of the compressed line "Tag bit". The subtractors are implemented based on the parallel prefix Kogge-Stone carry look-ahead adder which has low fan-out at each stage to increase the performance. The subtractors' scheme of two 8-bits elements $(a - b)$ is shown in Figure 5.6.

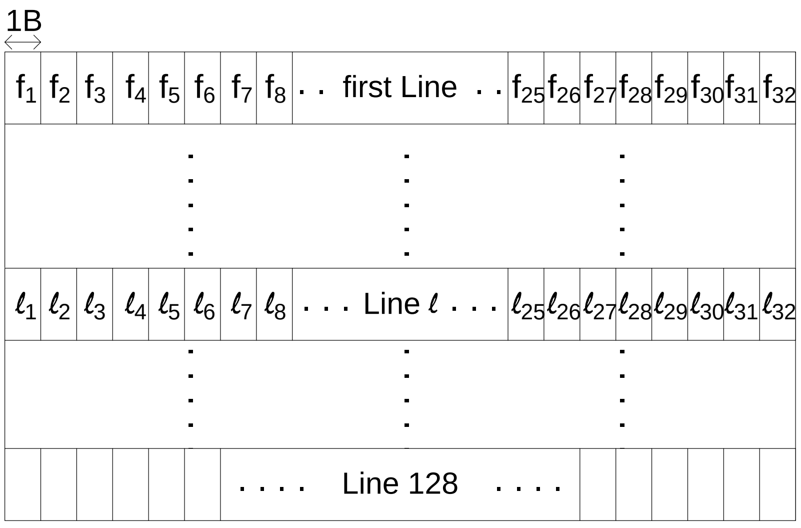

1B
f1 f2 f3 f4 f5 f6 f7 f8 . . first Line . . f25 f26 f27 f28 f29 f30 f31 f32
ℓ1 ℓ2 ℓ3 ℓ4 ℓ5 ℓ6 ℓ7 ℓ8 . . . Line ℓ . . . ℓ25 ℓ26 ℓ27 ℓ28 ℓ29 ℓ30 ℓ31 ℓ32
. . . . Line 128 . . . .

Figure 5.4 Block diagram of 4KB memory page

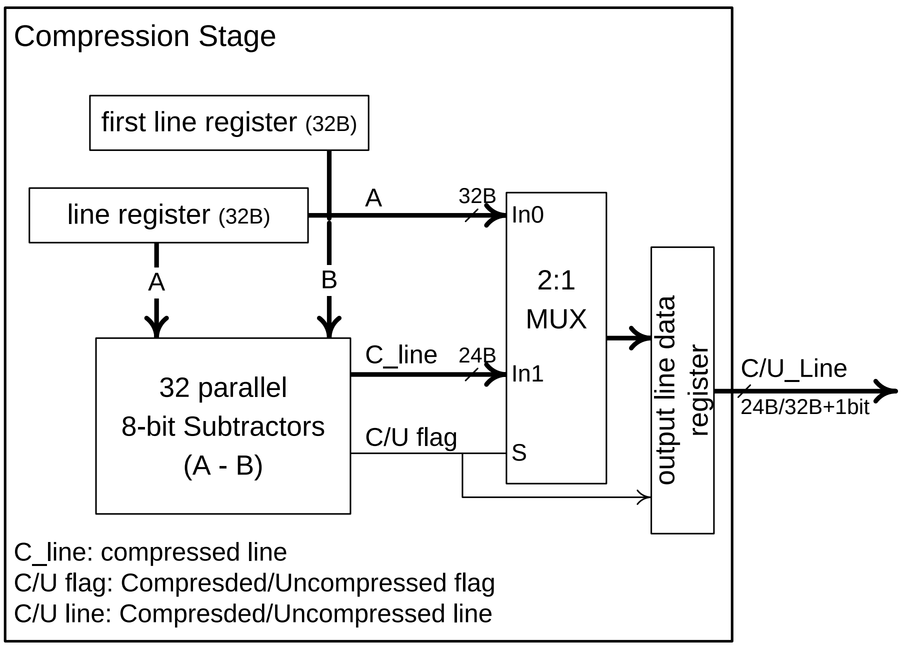

Compression Stage
first line register (32B)
line register (32B)
A
32B
In0
A
B
2:1 MUX
32 parallel 8-bit Subtractors (A - B)
C_line
24B
In1
C/U flag
S
output line data register
C/U_Line
24B/32B+1bit
C_line: compressed line
C/U flag: Compresded/Uncompressed flag
C/U line: Compresded/Uncompressed line

Figure 5.5 Compression block diagram for a 32B memory line

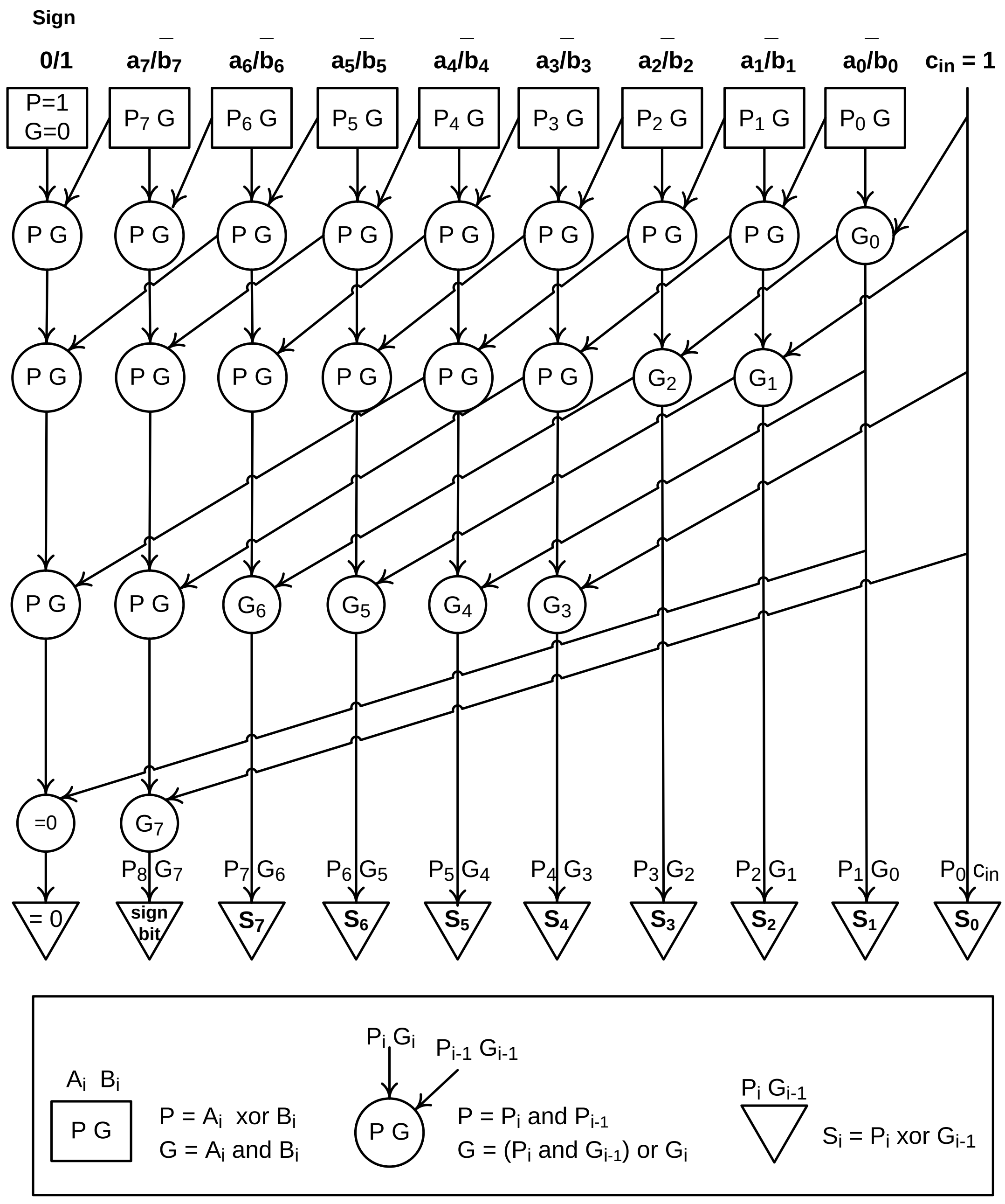


Figure 5.6 Details of the 8-bits subtractor, the basic unit of the compression design

In the decompression stage, Figure 5.7, firstly, the first bit of the memory line is checked. If its value is ‘1’, the line is compressed and the line data is stored in the next 24B. So, each byte of the compressed line data is added to its corresponding byte of the first line of this memory page. The critical latency of the decompression hardware design is simply 8-bits Kogge-Stone addition followed by a multiplexer.

Also, a variable-length design is implemented for the suggested approach. It has five checks after the subtractors’ block. It checks that all the Δs of the line are less than or equal 3-bits, 4-bits, 5-bits, 6-bits, and 7-bits, then compresses the line in the smallest possible length. However, a 3-bits tag is added to each compressed line.

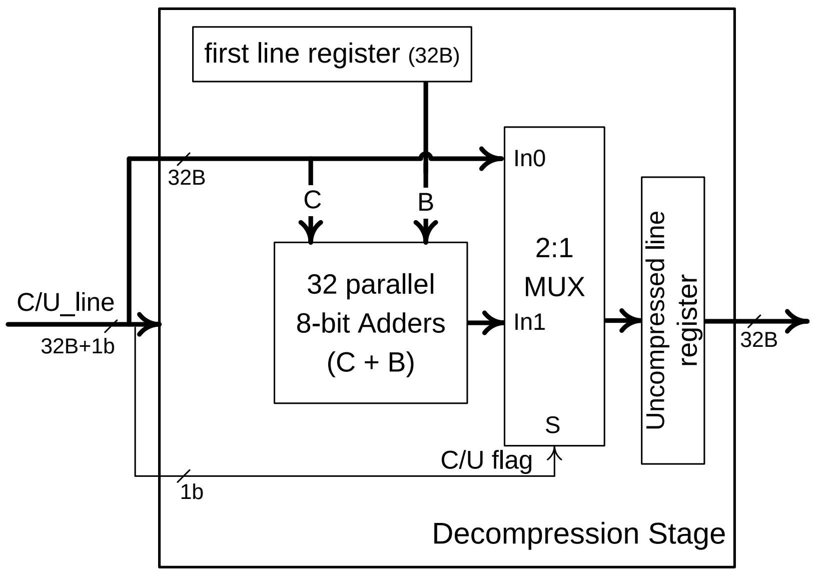


Figure 5.7 Decompression block diagram for 32B memory line

5.3. Comparison and Results

Compression and decompression have latency, energy and area overheads, which may affect the compression impact. In other words, the main tradeoffs for compression and decompression are compression ratio, latency and hardware complexity. The suggested design's advantages lie in its low latency and energy consumption if compared to available designs in literature. Also, the compressed line is fixed-length which provides less number of tag bits. Furthermore, delta compression has a good tradeoff between compression ratio, latency, and hardware complexity. The work of approximate computing in cache memory in [84] and [85] points similar cache lines to the same location in cache memory at the expense of small variations in applications last output. Nevertheless, the percentage of similarity, which is up to a maximum of 10%, in [84] is calculated by the average of differences between the two lines corresponding elements, $[(\Sigma(l_i - f_i))/32]$ according to Figure 5.4 symbols. Thus, it is possible to have more than 10% difference between two corresponding line elements.

This work suggests storing delta between similar lines in the main memory with similarity up to 37.5% between memory lines, and retrieves exact data values for main memory storage.

A comparison between suggested approach and literature approaches, Benini et al. [80] and base-delta-immediate [82], is obtained. They both use delta compression but depend on similarity within each data line. Benini et al. divide the 32B memory line to eight parts, each of length 4-Bytes/32-bits, then calculate the delta between the first part and the other parts. Furthermore, Base-delta-immediate methodology uses eight different compression blocks with different division of the memory line and delta length then chooses the smallest compression line.

Firstly, a statistics for the applications data similarity between memory lines is performed for 64B memory line using MATLAB and the results is shown in Table 5.1. A sample of different applications data is chosen. From PARSEC benchmark [86], the blackscholes application, an option pricing kernel that uses the blackscholes partial differential equation (PDE), data is a combination between floating point numbers and characters. The canneal application, a simulated cache-aware annealing kernel which optimizes the routing cost of a chip design, data is small integers and characters. PERFECT application 1 (Pa1) kernel benchmark is used for images binary file testing [87]. The applications data size is up to 16MB and the chosen page size is 4K page.

Then, the designs are implemented using VHDL for 32B line and synthesized in Synopsis Design Compiler B-2008.09 with NanGate 45nm Open Cell Library [55] at its typical operating voltage 1.1V. All simulation results are generated using default design compiler timing script with clock, load, and wire load constraints. Clock uncertainty is set to 10% of the clock period. For power analysis, a switching activity toggle rate of 75% is used. Table 5.2 shows that the suggested fixed-length approach decreases the latency by 67% and 83% than the two previous designs. The latency of the proposed compression stage is proportional to the latency of 8-bits parallel prefix subtractor delay and some logic for MUX and compression decision while the decompression has only 8-bits parallel prefix adder latency. However, the Latency of Benini et al., and Base-delta-immediate compression approaches are proportional to 32-bits, and 64-bits subtraction, respectively. And the logic for the compression decision according to their approaches. The proposed design low latency leads to high reduction in energy consumption, 66% and 43% less than the two previous designs. The frequency of operation is increased from 150 MHz, and 300MHz for Base-delta- immediate, and Benini et al. designs to 800MHz. The area of the suggested design and consequently power is increased slightly according to the usage of Kogge-Stone subtraction. Also, working on two lines has small effect in non-combinational area increment. Nevertheless, the fan-out is decreased which has noticeably an effect in latency reduction. The complex logic of the compression decision, which compares between five different outputs, increases the suggested variable-length design latency and energy. It is also one of the reasons of the high latency of Benini et al. and Base-delta-immediate approaches.

The compression ratio of the proposed approach for different applications is shown in Table 5.3 for alternative number of fixed-lengths compressed line and the variable-length design.

$$Compression\ Ratio\ = \frac{Uncompressed\ file\ size\ (Byte)}{Compressed\ file\ size\ (Byte)} \tag{5.1}$$

The simulations show good compression ratio of 1.3x for small integers and character data and of 1.16x for image applications. However, blackscholes application has a very low percentage of similarity, so extra tag bits in fact increase the file size. Also, comparisons between the compression ratios of the suggested design with the literature ones are shown in Table 5.4 for different benchmark applications.

Table 5.1 Similarity between memory lines (reference: 1st line of each 4K page)

	Similarity between lines within					approximate computing (T = 10%)
	100%	**62.5%**	**50%**	**37.5%**	**25%**	
Compressed Byte	**0-bit**	**4-bit**	**5-bit**	**6-bit**	**7-bit**	
Blackscholes application	0.4%	0.4%	0.4%	0.5%	1.1%	0.4%
Canneal application	0%	0%	2%	99%	99%	2.2%
Pa1 kernel benchmark	0%	7%	11%	23%	51%	23%

Table 5.2 Compression performance for different designs.

	Design	Area (mm^2)	Latency (ns)	Power (mW)	Energy (pJ)
variable-length	Benini et al. [80]	5.66	3.55	1.2	4.26
	Base-delta-immediate [82]	4.64	6.77	1.06	7.18
	Proposed variable-length	7.52	2.41	2.24	5.39
Proposed fixed-length		7.04	1.16	2.11	2.44

Table 5.3 Compression ratios for different benchmark applications for the suggested approach.

	fixed-length approach				variable-length approach
	Similarity between lines within				
	62.5%	**50%**	**37.5%**	**25%**	
Compressed Byte	**4-bit**	**5-bit**	**6-bit**	**7-bit**	
Blackscholes application	1.00	1.00	1.00	1.00	1.00
Canneal application	1.00	1.00	1.32	1.13	1.30
Pa1 kernel benchmark	1.02	1.04	1.06	1.06	1.11

Table 5.4 Compression ratios for the different designs

	Benini et al. [80]	**Base-delta-Immediate [82]**	**Proposed fixed-length**	**Proposed variable-length**
Blackscholes application	1.00	1.00	1.00	1.00
Canneal application	1.00	1.00	1.32	1.3
Pa1 kernel benchmark	1.04	1.00	1.06	1.11

5.4. Preparing design for fabrication

In order to prepare the design for fabrication, the compression block diagram is designed as shown in Figure 5.8. The memory lines are read 8-bits by 8-bits so every 32 clock cycles, new line (row) is read. The design has three inputs; reset, clk, and line element (L_e). It has five outputs; first line and its enable bit, compressed/uncompressed line (C/U_Line) and its enable bit, and flag bit shows that the output (C/U_Line) is compressed or not (C/U_f). Six extra outputs is added for testing.

The compression block diagram design post translate functionality is firstly checked with FPGA post translate simulation model to edit any faults in the design after adding the gates delays. Then the design is checked after the post synthesis ASIC simulation where the output files from the synopsis design compiler are linked with the UMC-130nm technology library files. The hdl file, which contain hardware implementation of the design using technology gates description, and the sdf file, which contain the technology gate delays is simulated using ModelSim. The following simulation results are for four memory lines from the Canneal application. The four testing lines are

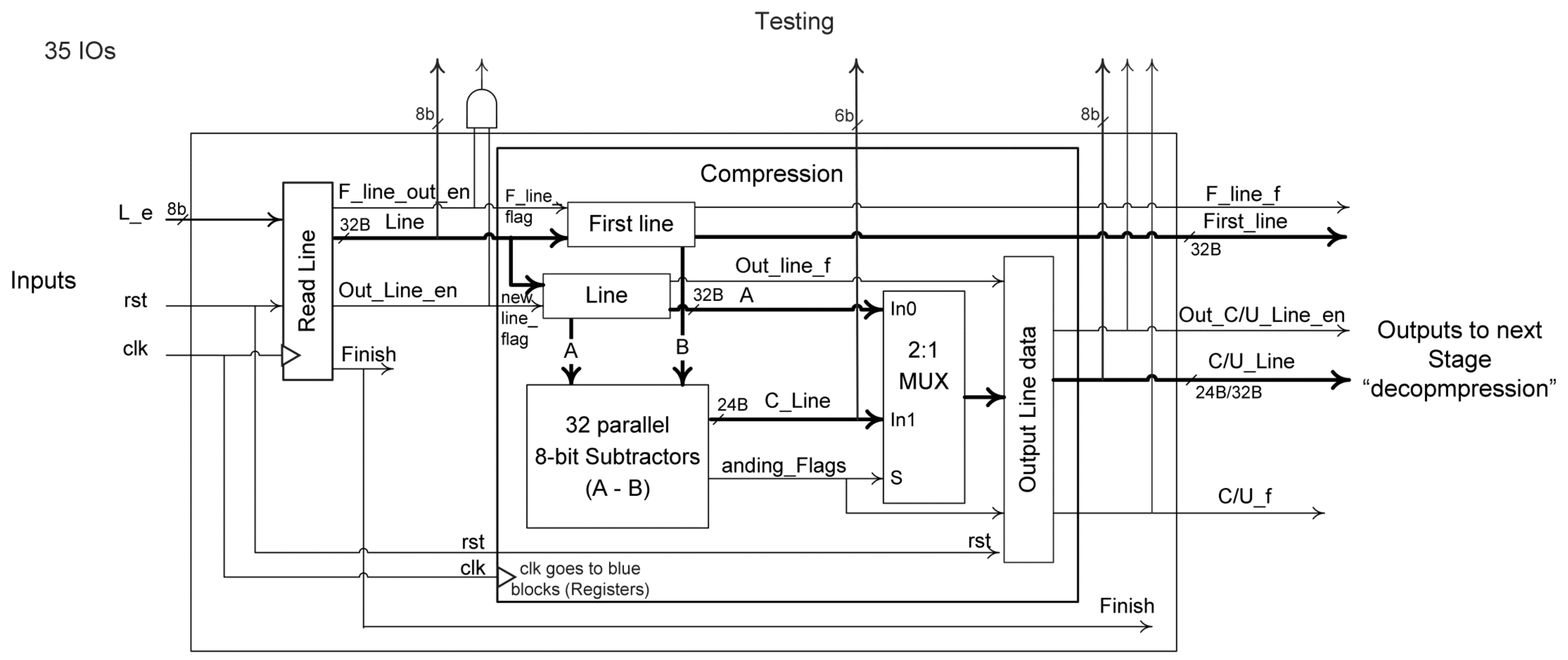
1) 61 02 67 61 68 64 68 62 02 63 6A 67 6A 61 63 02 68 65 63 61 66 64 01 6A 67 62 66 6A 00 00 00 00; first line

2) 65 01 69 67 63 64 63 66 01 61 63 6A 61 67 67 02 64 65 62 68 6A 68 01 66 63 65 62 63 00 00 00 00; will be compressed

3) 69 01 65 63 69 63 68 6A 02 69 66 68 61 64 61 02 67 61 68 64 68 62 02 63 6A 67 6A 61 00 00 FF 00; can’t be compressed

4) 69 01 65 63 69 63 68 6A 02 69 66 68 61 64 61 02 67 61 68 64 68 62 02 63 6A 67 6A 61 00 00 00 00; will be compressed

After reading the last element of a line, the compression takes one clock cycle and the result is stored in the output register (output line data). Then the outputs, C/U_f, C/U_Line and its enable, are output in the next clock as shown in Figure 5.9 and Figure 5.10 for FPGA and ASIC simulation results, respectively. Lastly, Figure 5.11 shows the ASIC post synthesis simulation results for the above four testing memory lines of Canneal application.



L_e => line_element
F_line_flag => First line flag

C_Line -> Compressed_Line

A -> A (uncompressed line)

C/U_Line -> Compressed/Uncompressed Line

Figure 5.8 Compression design block diagram after editing for fabrication

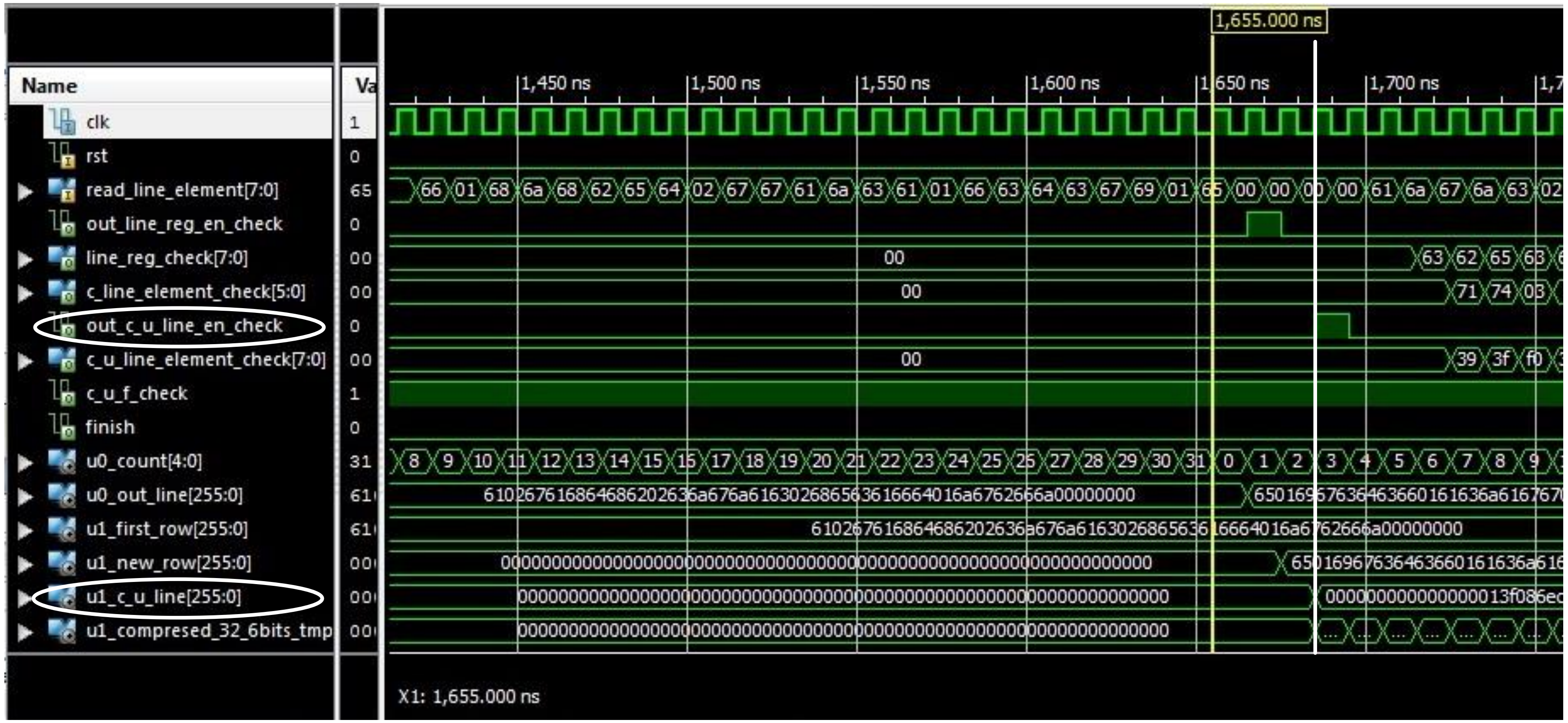


Figure 5.9 FPGA post translate simulation results

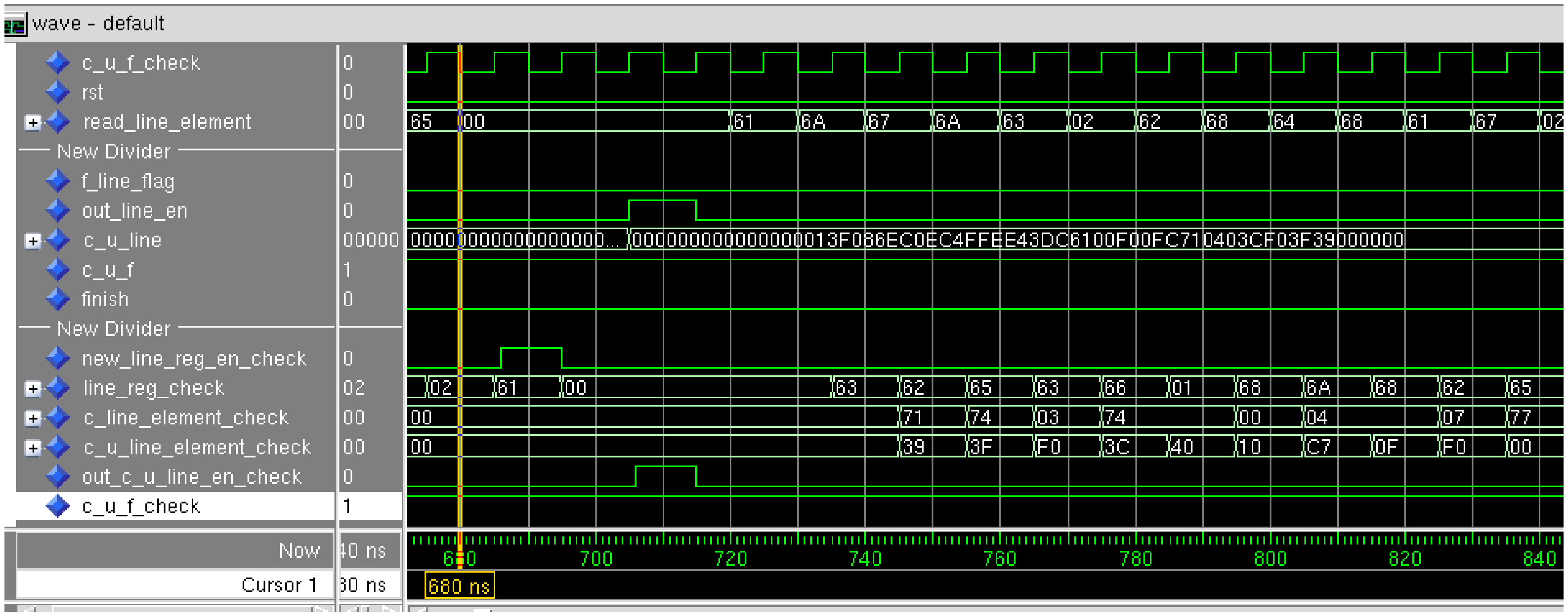


Figure 5.10 ASIC post synthesis simulation results

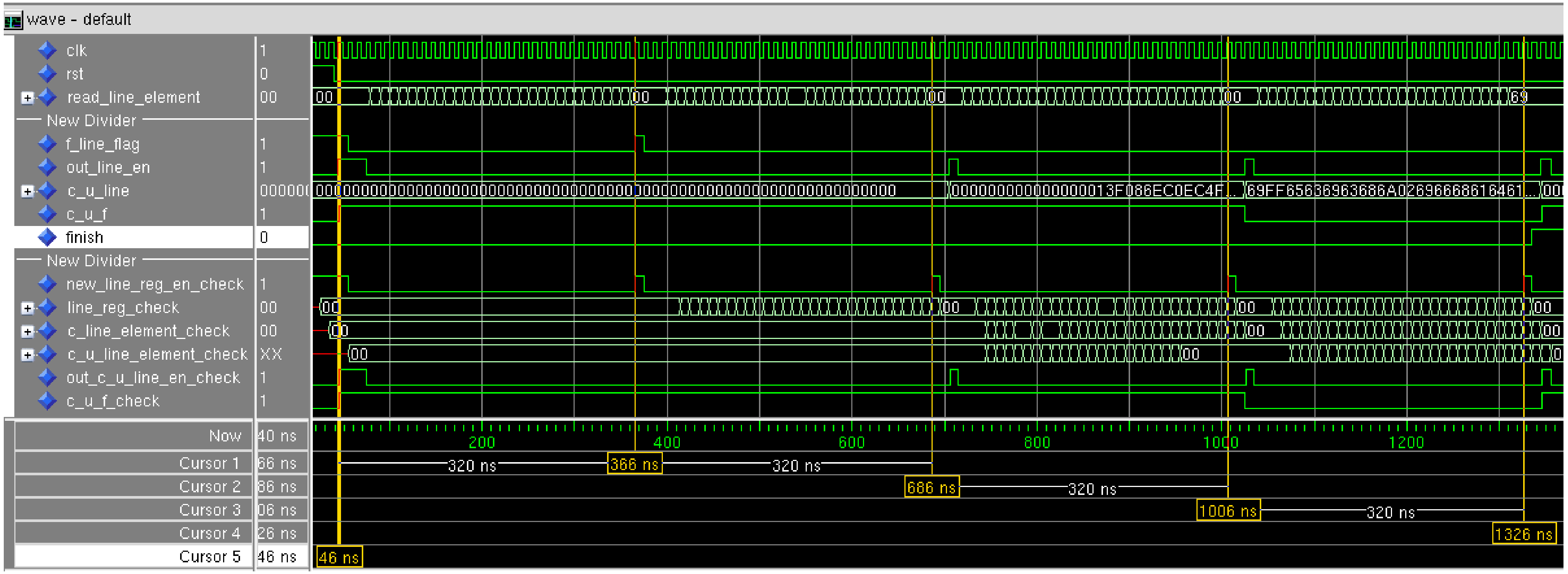


Figure 5.11 ASIC post synthesis simulation results for four memory line

5.5. Summary

This chapter introduces a detailed explanation for the recent memory compression approaches in literature. Then the suggested methodology for main memory compression and decompression is discussed. A comparison for the compression ratio and performance of the different designs shows reduction in energy up to 66% for the suggested design when compared to literature designs. And the frequency of operation is increased from 300 MHz to 800 MHz. Also, the new design allows the main memory to store up to 30% more data. Finally, the preparation of the suggested design for fabrication is presented.

Conclusions and future work

Low power/energy has been a major design constraint. The explosion in digital communications and the desire to preserve battery life time, improve system reliability, and reduce cooling costs has pushed for extensive research in low power/energy digital designs. Processing unit is the dominant part of our Information and Communications Technologies (ICT). ICT is mainly composed of our computers, smart phones and digital TVs along with its supporting computer server farms which support the cloud. The real cost of computation in our exploding ICT world is simply the cost of energy (power * latency) consumption. Making this cost cheaper is very important. This thesis proposes two low energy designs at circuit, and architecture levels of abstraction for multiplier in the arithmetic and logic unit (ALU), and main memory storage data, respectively.

The first suggested design is a low energy dual base multiplier design. It modifies existing architectures at the partial products accumulation stage. Partial products accumulation stage is the most significant stage of the multiplier as it has the largest area and delay. The suggested DBM divide the accumulation stage into two parts. The first part is the main tree, which is designed using a binary column tree. Binary column tree usage in dual binary/decimal accumulation stage is very efficient. Binary addition in the main accumulation part for binary and decimal accumulation decreases the area and power of the multiplier. The second part is a clock gated split binary/decimal tree. Splitting the accumulation stage at the last few addition levels decreases the area of the accumulation stage, and separates decimal overhead delays from the binary multiplication path. Decimal correction blocks are gathered and implemented after the binary column tree, at the beginning of split decimal path. The decimal correction blocks are represented in binary to decimal conversion blocks, maximally 9-bits B/D conversion tree. Besides, pipelining and clock gating decreases the dynamic power dissipation further. The proposed pipelined multiplier design provides significant reductions in area, power, and energy with acceptable delay. For more power reduction, power gating is suggested to significantly reduce internal and leakage power. Also, adopting one of the low power/delay full adder cell designs in the critical path will decrease the total power/delay of the multiplication operation even more.

The second suggested design is a low latency and low energy compression/decompression approach for main memory data. The performance of the proposed algorithm is evaluated according to compression ratio and implementation complexity which affects latency and energy overhead. If compared to other designs in literature, the design offers small latency and high reduction in energy consumption. The design achieves good compression ratios for applications that have integer or character data types with small to medium differences between its data values. An extra hardware can be added to increase the compression ratio if the data type is taken into consideration. For example, the memory line might be divided into elements of 4B for floating point data. Taking into consideration that the Kogge-stone addition will guarantee a small increase in latency. Also, run length encoding can be used after the delta compression to improve the compression ratio.

There are various solutions for low power/energy designs at all levels of abstraction such as system and architecture framework, circuit design, power gating, and new low energy materials and technologies. The improvement at higher levels of abstraction needs lower-cost design engineers and affects all subsequent abstraction levels to comply with the changes at that higher level. However, new technologies take us to a new significant level of performance improvement despite it needs higher-cost design engineers and a long period of time compared to higher levels of abstraction.

References

[1] G. E. Moore, “Cramming more components onto integrated circuits,” *Electronics*, pp. 52–59, 1965.

[2] G. E. Moore, “Progress In Digital Integrated Electronics,” in *Technical Digest, International Electron Devices Meeting*, 1975, pp. 11–13.

[3] G. E. Moore, “Lithography and the future of Moore’s law,” in *Symposium on microlithography (SPIE’S)*, 1995, vol. 2438, pp. 2–17.

[4] P. R. Panda, B. V. N. Silpa, A. Shrivastava, and K. Gummidipudi, *Power-efficient system design*. Springer, Boston, MA, 2010.

[5] A. Jain *et al.*, “Concise loads and stores: The case for an asymmetric compute-memory architecture for approximation,” in *Proceedings of the Annual International Symposium on Microarchitecture, MICRO*, 2016.

[6] Q. Xu, T. Mytkowicz, and N. S. Kim, “Approximate Computing: A Survey,” *IEEE Des. Test Test*, vol. 33, no. 1, pp. 8–22, 2016.

[7] D. A. F. El-dib, “Low Power Register Exchange Viterbi Decoder for Wireless Applications,” Ph.D. dissertation, University of Waterloo, 2004.

[8] M. Anis and M. Elmasry, *Multi-threshold CMOS digital circuits-managing leakage power*. Kluwer Academic Publishers, 2003.

[9] S. Gupta and S. Padave, “Power Optimization for Low Power VLSI Circuits,” *Int. J. Adv. Res. Comput. Sci. Softw. Eng.*, vol. 6, no. 3, pp. 96–99, 2016.

[10] R. Santhiya and M. T. Thamaraimanalan, “Power Gating Based Low Power 32 Bit BCD Adder using DVT,” *Int. J. Sci. Res. Dev.*, vol. 3, no. 02, pp. 802–805, 2015.

[11] Y. Li, “Memory-centric low power digital system design,” Ph.D. dissertation, Rensselaer Polytechnic Institute, 2012.

[12] D. J. Moni and P. E. Sophia, “Design of low power and high speed configurable booth multiplier,” in *3rd International Conference on Electronics Computer Technology (ICECT)*, 2011, vol. 6, pp. 338–342.

[13] C. Sharma, “Power Reduction in VLSI chips by Optimizing Switching Activity at Test Process , Architecture & Gate Level,” *Int. J. Eng. Sci. Technol.*, vol. 3, no. 4, pp. 3256–3259, 2011.

[14] J. Han and M. Orshansky, “Approximate computing: An emerging paradigm for energy-efficient design,” in *18th IEEE European Test Symposium (ETS)*, 2013, pp. 1–6.

[15] Q. Guo, K. Strauss, L. Ceze, and H. S. Malvar, “High-Density Image Storage Using Approximate Memory Cells,” in *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, 2016, pp. 413–426.

[16] B. Holt, J. Bornholt, I. Zhang, D. Ports, M. Oskin, and L. Ceze, “Disciplined Inconsistency with Consistency Types,” in *Proceedings of the Seventh ACM Symposium on Cloud Computing*, 2016, pp. 279–293.

[17] J. Von Neumann, “Probabilistic logics and the synthesis of reliable organisms from unreliable components,” *Autom. Stud.*, vol. 34, no. 34, pp. 43–98, 1956.

[18] A. Alaghi and J. P. Hayes, “Survey of stochastic computing,” *ACM Trans. Embed. Comput. Syst.*, vol. 12, no. 2s, p. 92, 2013.

[19] E. Schkufza, R. Sharma, and A. Aiken, “Stochastic Optimization of Floating-point Programs with Tunable Precision,” in *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2014, pp. 53–64.

[20] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, “Low-Power Digital Signal Processing Using Approximate Adders,” *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 32, no. 1, pp. 124–137, Jan. 2013.

[21] D. Mohapatra, V. K. Chippa, A. Raghunathan, and K. Roy, “Design of voltage-scalable meta-functions for approximate computing,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2011, pp. 1–6.

[22] J. Miao, K. He, A. Gerstlauer, and M. Orshansky, “Modeling and Synthesis of Quality-energy Optimal Approximate Adders,” in *Proceedings of the International Conference on Computer-Aided Design*, 2012, pp. 728–735.

[23] N. Zhu, W. L. Goh, W. Zhang, K. S. Yeo, and Z. H. Kong, “Design of low-power high-speed truncation-error-tolerant adder and its application in digital signal processing,” *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 18, no. 8, pp. 1225–1229, 2010.

[24] Y. Kim, Y. Zhang, and P. Li, “Energy Efficient Approximate Arithmetic for Error Resilient Neuromorphic Computing,” *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 23, no. 11, pp. 2733–2737, 2015.

[25] 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, 2015.

[26] A. Yazdanbakhsh *et al.*, “Axilog: Language support for approximate hardware design,” in *Design Automation and Test in Europe (DATE)*, 2015, pp. 812–817.

[27] K. Nepal, Y. Li, R. I. Bahar, and S. Reda, “ABACUS: A technique for automated behavioral synthesis of approximate computing circuits,” in *Design Automation and Test in Europe (DATE)*, 2014, pp. 1–6.

[28] C. Li, W. Luo, S. S. Sapatnekar, and J. Hu, “Joint precision optimization and high level synthesis for approximate computing,” in *52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2015, pp. 1–6.

[29] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, “Architecture Support for Disciplined Approximate Programming,” in *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2012, pp. 301–312.

[30] U. R. Karpuzcu, I. Akturk, and N. S. Kim, “Accordion: Toward soft Near-Threshold Voltage Computing,” in *IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, 2014, pp. 72–83.

[31] L. Benini, A. Bogliolo, and G. De Micheli, “A survey of design techniques for system-level dynamic power management,” *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 8, no. 3, pp. 299–316, Jun. 2000.

[32] J. Antony and J. Pathak, “Design and implementation of high speed baugh wooley and modified booth multiplier using cadence RTL,” *Int. J. Res. Eng. Technol.*, vol. 3, no. 8, pp. 56–63, 2014.

[33] J.-T. Yan and Z.-W. Chen, “Low-power multiplier design with row and column bypassing,” in *IEEE International SOC Conference SOCC*, 2009, pp. 227–230.

[34] M.-C. Wen, S.-J. Wang, and Y.-N. Lin, “Low-power parallel multiplier with column bypassing,” in *IEEE International Symposium on Circuits and Systems ISCAS*, 2005, vol. 2, pp. 1638–1641.

[35] Z. Abid, H. El-Razouk, and D. A. A. El-Dib, “Low power multipliers based on new hybrid full adders,” *Microelectronics J.*, vol. 39, no. 12, pp. 1509–1515, Dec. 2008.

[36] M. A. Valashani and S. Mirzakuchaki, “Two new energy-efficient full adder designs,” in *24th Iranian Conference on Electrical Engineering (ICEE)*, 2016, pp. 655–660.

[37] I. S. Abu-Khater, A. Bellaouar, and M. I. Elmasry, “Circuit techniques for CMOS low-power high-performance multipliers,” *IEEE J. Solid-State Circuits*, vol. 31, no. 10, pp. 1535–1546, 1996.

[38] J. B. Kim, “An area efficient multiplier using current-mode quaternary logic technique,” in *10th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*, 2010, pp. 403–405.

[39] P. Kimfors *et al.*, “Custom layout strategy for rectangle-shaped log-depth multiplier reduction tree,” in *16th IEEE International Conference on Electronics, Circuits, and Systems*, 2009, vol. 1, no. c, pp. 77–80.

[40] Q. Tong, K. Choi, and J. D. Cho, “A review on system level low power techniques,” *Pervasive Technol.*, vol. 1, no. 1, 2015.

[41] “Decimal Arithmetic FAQ, Part 1 – General Questions.” [Online]. Available: http://speleotrove.com/decimal/decifaq1.html. [Accessed: 17-Jan-2016].

[42] M. F. Cowlishaw, “Decimal floating-point: algorism for computers,” in *16Th IEEE Symposium On Computer Arithmetic*, 2003, pp. 104–111.

[43] E. M. Schwarz, “Decimal Multiplication with Efficient Partial Product Generation,” in *Proceedings of the 17th IEEE Symposium on Computer Arithmetic*, 2005, pp. 21–28.

[44] M. A. Erle and M. J. Schulte, “Decimal multiplication via carry-save addition,” in *Proceedings IEEE International Conference on Application-Specific Systems, Architectures, and Processors. ASAP 2003*, 2003, pp. 348–358.

[45] T. Lang and A. Nannarelli, “A Radix-10 Combinational Multiplier,” in *Asilomar Conference on Signals, Systems and Computers*, 2006.

[46] A. Vázquez, E. Antelo, and P. Montuschi, “A new family of high performance parallel decimal multipliers,” in *18th IEEE Symposium on Computer Arithmetic*, 2007, pp. 195–204.

[47] R. D. Kenney, M. J. Schulte, and M. A. Erle, “A high-frequency decimal multiplier,” in *IEEE International Conference on Computer Design: VLSI in Computers and Processors, 2004. ICCD 2004. Proceedings.*, 2004, pp. 26–29.

[48] L. Dadda, “Multioperand parallel decimal adder: A mixed binary and BCD approach,” *IEEE Trans. Comput.*, vol. 56, no. 10, pp. 1320–1328, 2007.

[49] B. Hickmann, M. Schulte, and M. Erle, “Improved combined binary/decimal fixed-point multipliers,” in *IEEE International Conference on Computer Design*, 2008, pp. 87–94.

[50] M. Mahmoud and H. A. H. Fahmy, “A parallel combined binary/decimal fixed-point multiplier with binary partial products reduction tree,” in *21st International Conference on Computer Theory and Applications (ICCTA)*, 2011.

[51] A. Vazquez and E. Antelo, “Improved Design of High-Performance Parallel Decimal Multipliers,” *IEEE Trans. Comput.*, vol. 59, no. 5, 2010.

[52] G. Jaberipur and A. Kaivani, “Improving the Speed of Parallel Decimal Multiplication,” *IEEE Trans. Comput.*, vol. 58, no. 11, pp. 1539–1552, 2009.

[53] P. Kogge and H. S. Stone, “A parallel algorithm for the efficient solution of a general class of recurrence equations,” *IEEE Trans. Comput.*, vol. C-22, no. 8, pp. 786–793, 1973.

[54] B. M. Benedek, "Developing large binary to BCD conversion structures," in *IEEE 3rd Symposium on Computer Arithmetic (ARITH)*, 1975, pp. 188–196.

[55] "NanGate 45nm Open Cell Library." [Online]. Available: http://www.nangate.com/?page_id=2325. [Accessed: 17-Jan-2016].

[56] "NanGate 15nm Open Cell Library." [Online]. Available: http://www.nangate.com/?page_id=2328. [Accessed: 17-Jan-2016].

[57] O. Mutlu, J. Meza, and L. Subramanian, "The main memory system: challenges and opportunities," *Communications of the Korean Institute of Information Scientists and Engineers*, pp. 16–41, 2015.

[58] D. A. Patterson and J. L. Hennessy, *Computer organization and design, the hardware/software interface*, 5th ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2013.

[59] H. Geng, *Semiconductor Manufacturing Handbook, Second Edition*. McGraw-Hill Education, 2017.

[60] I. Corporation, "Intel® Solid-State Drives in Server Storage Applications," 2014.

[61] S. Lehmann and F. Gerfers, "Channel analysis for a 6.4 Gb/s DDR5 data buffer receiver front-end," in *15th IEEE International New Circuits and Systems Conference (NEWCAS)*, 2017, pp. 109–112.

[62] J. He, S. Kannan, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "The Unwritten Contract of Solid State Drives," in *Proceedings of the Twelfth European Conference on Computer Systems*, 2017, pp. 127–144.

[63] A. Shafiee, M. Taassori, R. Balasubramonian, and A. Davis, "MemZip: Exploring unconventional benefits from memory compression," in *Proceedings of the International Symposium on High-Performance Computer Architecture*, 2014, pp. 638–649.

[64] S. Mittal and J. S. Vetter, "A Survey Of Architectural Approaches for Data Compression in Cache and Main Memory Systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 9219, no. c, pp. 1–14, 2015.

[65] D. Salomon and G. Motta, *Handbook of Data Compression*, Fifth Edit. Springer-Verlag London Ltd, 2010.

[66] Z. M. D. Dasgupta, *Evolutionary Algorithms in Engineering Applications*. Springer-Verlag Berlin Heidelberg, 1997.

[67] K. SAYOOD, *INTRODUCTION TO DATA COMPRESSION*. MORGAN KAUFMANN PUBLISHER, 2017.

[68] A. T. Alistair Moffat, *Compression and coding algorithms*. Springer Science+Business Media, LLC, 2002.

[69] M. Borda, *Fundamentals in Information Theory and Coding*. Springer-Verlag Berlin Heidelberg, 2011.

[70] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Trans. Inf. Theory*, vol. 23, no. 3, pp. 337–343, 1977.

[71] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE Trans. Inf. Theory*, vol. 24, no. 5, pp. 530–536, 1978.

[72] G. Pekhimenko *et al.*, "Exploiting compressed block size as an indicator of future reuse," in *21st IEEE International Symposium on High Performance Computer Architecture, HPCA*, 2015, pp. 51–63.

[73] B. Falsafi and T. F. Wenisch, *A Primer on Hardware Prefetching*. Morgan & Claypool Publishers, 2014.

[74] Y. Tian, S. M. Khan, D. A. Jiménez, and G. H. Loh, "Last-level Cache Deduplication," in *Proceedings of the 28th ACM International Conference on Supercomputing*, 2014, pp. 53–62.

[75] A. Arelakis and P. Stenstrom, "SC2: A Statistical Compression Cache Scheme," in *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, 2014, pp. 145–156.

[76] M. Ekman and P. Stenstrom, "A robust main-memory compression scheme," in *Proceedings - International Symposium on Computer Architecture*, 2005, pp. 74–85.

[77] B. Abali *et al.*, "Memory expansion Technology (MXT): software support and performance," *IBM J. Res. Dev.*, vol. 45, no. 2, pp. 287–301, 2001.

[78] J. Dusser, T. Piquet, and A. Seznec, "Zero-content Augmented Caches," in *Proceedings of the 23rd International Conference on Supercomputing*, 2009, pp. 46–55.

[79] J. Dusser and A. Seznec, "Decoupled zero-compressed memory," in *HiPEAC'11 - Proceedings of the 6th International Conference on High Performance and Embedded Architectures and Compilers*, 2011, pp. 77–86.

[80] L. Benini, D. Bruni, A. Macii, and E. Macii, "Memory energy minimization by data compression: algorithms, architectures and implementation," *IEEE Trans. Very Large Scale Integr.*, vol. 12, no. 3, pp. 255–268, 2004.

[81] A. Alameldeen and D. Wood, "Frequent Pattern Compression : A Significance-Based Compression Scheme for L2 Caches," 2004.

[82] G. Pekhimenko, V. Seshadri, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Base-delta-immediate Compression: Practical Data Compression for On-chip Caches," in *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*, 2012, pp. 377–388.

[83] G. Pekhimenko *et al.*, "Linearly compressed pages: A low-complexity, low-latency main memory compression framework," in *Proceedings of the 46th International Symposium on Microarchitecture*, 2013, pp. 172–184.

[84] J. S. Miguel, J. Albericio, A. Moshovos, and N. E. Jerger, "Doppelgänger: A Cache for Approximate Computing," in *Proceedings of the 48th International Symposium on Microarchitecture*, 2015, pp. 50–61.

[85] J. S. Miguel, J. Albericio, N. E. Jerger, and A. Jaleel, "The Bunker Cache for spatio-value approximation," in *49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016, pp. 1–12.

[86] C. Bienia and K. Li, " PARSEC 2.0: A New Benchmark Suite for Chip-Multiprocessors ," in *Proc. of the 5th Annual Workshop on Modeling, Benchmarking and Simulation*, 2009.

[87] K. Barker *et al.*, "PERFECT (Power Efficiency Revolution For Embedded Computing Technologies) Benchmark Suite Manual." 2013.

Appendix A: RTL synthesis flow commands using Synopsys Design Compiler (DC)

To execute any UNIX command in dc shell:

```
» sh <UNIX_command>
```

Main Steps:

1. Library setup

```
» set search_path “$search_path”
```

# DC looks for specified design and library files in the search_path directories
# Default path is the current work directory (CWD): The directory that dc_shell is invoked from

```
» link_library=target_library={65nm.db}
```

or

```
» set target_library 65nm.db

» set link_library “* $target_library”
```

2. Analyze.

Load all VHDL code

```
» define_design_lib WORK –path ./work

» read -f vhdl {arbiter.vhd top.vhd}
```

# read_vhdl command creates several intermediate files and directories which collectively form the “VHDL Design Library”.

3. Elaborate.

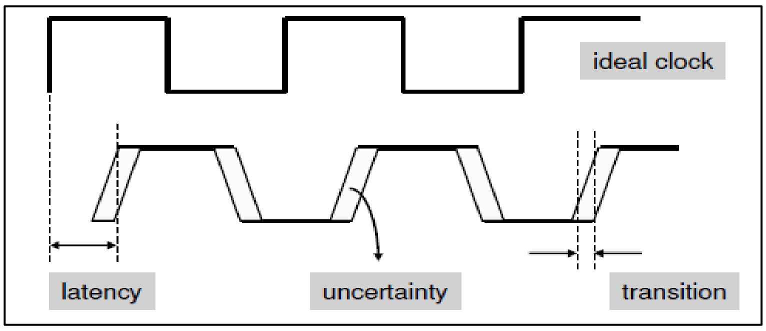
Choose top level vhdl file.

```
» current_design TOP
```

4. Link Design with library

```
» link
```

5. Setting up the clock



If the design doesn't have a clock:

```
» create_clock -period 40 -waveform {0 20} -name sys_clk
```

If the design does have a clock pin:

```
» create_clock clk -period 40 -waveform {0 20}
```

```
» set_clock_uncertainty 0.14 clk
```

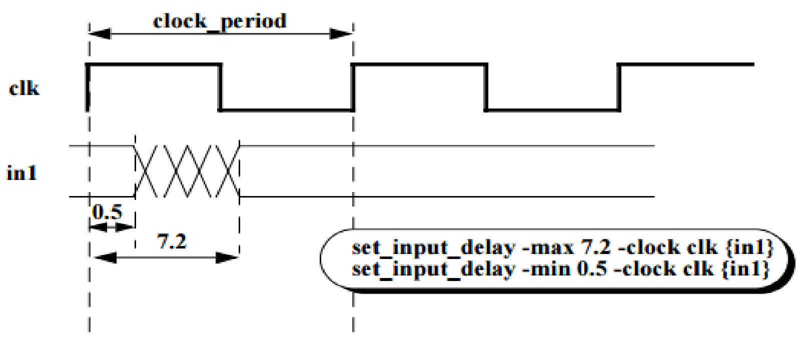
# uncertainty models the maximum delay difference between the clock network branches, known as clock skew, but can also include clock jitter and margin effects

```
» set_clock_latency -max 0.3 clk
```

# Network latency models the average 'internal' delay from the create_clock port or pin to the register clock pins
# it is useful when clock generation circuitry is not part of the design, or for derived clocks

```
» set_clock_transition 0.08 CLK
```

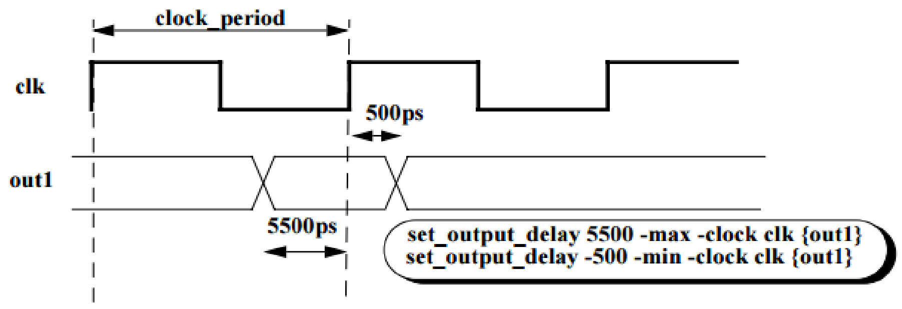
# Transition models the rise and fall times of the clock waveform at the register clock pins


clock_period
clk
in1
0.5
7.2
set_input_delay -max 7.2 -clock clk {in1}
set_input_delay -min 0.5 -clock clk {in1}

```
» set_input_delay 2.0 -clock clk [all_inputs]
```

# depend on the slowest DFF in the library (i.e. The DFF has a clock-Q delay of 1.75ns + 0.25ns for wiring delay = 2.0 ns)

» remove_input_delay Clk



clock_period
clk
500ps
out1
5500ps
set_output_delay 5500 -max -clock clk {out1}
set_output_delay -500 -min -clock clk {out1}

» set_output_delay 1.65 -clock clk [all_outputs]

# depend on the slowest DFF in the library (i.e. the DFF has a setup time of 1.4ns + 0.25 for wiring delay = 1.65ns)

» set_load 0.1 [all_outputs]

#for Registered Outputs, load capacitance

» set_max_fanout 1 [all_inputs]

» set_fanout_load 8 [all_outputs]

6. Constraints

» dont_touch_network B_D_control

» set_wire_load_mode enclosed

» set_wire_load_model –name 1.6MGates

# Technology library may have a default wire load model specified, which will be used if no wire load model is manually or automatically applied. To find out what the default model is use one of the following commands:

» get_attribute <lib_name> default_wire_load
» report_lib <lib_name>

7. Check Design

» check_design

8. compile_ultra

» compile_ultra –scan –retime –timing

9. Store all of the results.

» write -format ddc -output project_name.ddc

» read_ddc project_name.ddc

» set_switching_activity -toggle_rate 0.75 -select inputs -hier -clock clk

#75% toggle rate of inputs with all nets annotation in saif file

or

» read_saif -input name.saif

» report_power -analysis_effort high > name_power_report_log.txt

» report_power -analysis_effort high > name_log_file.txt

» report_power -analysis_effort high -hierarchy -levels 2 > output_log_file.txt

» report_timing -path_type full -max_paths 10 -significant_digits 5 > name_timing_power_report.txt

» report_area

10. reset_design

الملخص

إن الزيادة المستمرة في تكامل الرقائق واستهلاك الطاقة المرتبطة بها جعل تصميم الدوائر منخفضة الطاقة/القدرة من أهم التحديات التي تواجه الدوائر الالكترونية المتكاملة فائقة الاختزال. حيث أن العائق الرئيسي للتحرك نحو أجهزه قابلة للنقل هو ارتفاع استهلاك الطاقة/القدرة. إدارة الطاقة لها استراتيجيات متعدده على جميع مراحل تصميم الدوائر المتكاملة. ومع ذلك، تحسين مستوى الطاقة/القدرة في مراحل تصميم الدوائر أو بنيتها أو نظامها يحقق تقليل كبير في الطاقة/القدرة مقارنة بتقنيات المستويات الأخرى. في هذه الرسالة، نقدم تصميمين مختلفين من التصميمات في مرحلة التصميم والبنية الخاصة بالأجهزه لتقليل استهلاك الطاقة/القدرة لمضاعف ثنائي/عشري ولنظام الذاكرة الرئيسية.

تجميع النظام الثنائي والعشري في تصميم الوحدات الحسابية للحاسب هو الأمثل للحصول على تطبيقات عالية السرعة ومنخفضة الطاقة. تم اقتراح تصميم لمضاعف منخفض الطاقة معالج بالتجزئة ذو اساس مزدوج ثنائي/عشري. أُجريت دراسة شاملة على التصميم المقترح والتصميمات السابقة والتي أثبتت انخفاض هائل في استهلاك الطاقة والقدرة والمساحة. تم اختيار طريقة تجزئة التصميم للحصول على طاقة/قدرة منخفضة. التصميم يسمح باستخدام ترددات تصل إلى 4 جيجاهرتز باستخدام تكنولوجيا ذات طول 15 نانومتر. التصميم المقترح للمضاعف قدم تخفيض في المساحه يصل إلى 37% وتخفيض للطاقة يصل إلى 43%.

أيضا، تم اقتراح نهج لضغط الذاكرة الرئيسية باستخدام تصميم بسيط منخفض الطاقه لتخزين البيانات بجودة عالية للتطبيقات التي تحتاج إلى نتائج دقيقة. المساهمة الرئيسية للتصميم المقترح هو انخفاض استهلاكها للطاقة عن التصميمات السابقة بحوالي 66% نظراً لبساطته وانخفاض الزمن بين المدخلات والمخرجات. تم تحسين تردد التصميم المقترح عن التصميمات السابقة من 300 ميجاهرتز إلى 800 ميجاهرتز. التصميم يسمح للذاكرة الرئيسية بتخزين بيانات أكثر بنسبة تصل إلى 30% وفقا لمؤشرات الأداء الخاصة ببيانات التطبيقات.

تصميمات منخفضة الطاقة لبنية الحاسب

اعداد
مرفت محمد عادل محمود

رسالة مقدمة إلى كلية الهندسة – جامعة القاهرة
كجزء من متطلبات الحصول على درجة **دكتوراه الفلسفة**
في
هندسة الالكترونيات والاتصالات الكهربية

تحت اشراف

أ.د. حسام علي حسن فهمي
قسم الالكترونيات والاتصالات،
كلية الهندسة، جامعة القاهرة

د. داليا عبد الواحد فؤاد الديب
قسم الهندسة الكهربائية وهندسة
الحاسبات، كلية الهندسة، جامعه
دالهوزي ، كندا

كليــة الهندســة - جامعــة القاهــرة
الجيـزة - جمهوريــة مصـر العربيــة

سنة 2019

تصميمات منخفضة الطاقة لبنية الحاسب

اعداد

مرفت محمد عادل محمود

رسالة مقدمة إلى كلية الهندسة – جامعة القاهرة
كجزء من متطلبات الحصول على درجة **دكتوراه الفلسفة**
في
هندسة الالكترونيات والاتصالات الكهربية

كليـــة الهندســـة - جامعـــة القاهـــرة
الجيـزة - جمهوريـة مصـر العربيــة

سنة 2019