Speed Comparison of Binary Adders Techniques

by

Abdulmajeed Alghamdi
B.Sc, Yanbu Industrial College, 2011

A Project Report Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Engineering

in the Department of Electrical and Computer Engineering

Speed Comparison of Binary Adders Techniques

by

Abdulmajeed Alghamdi
B.Sc, Yanbu Industrial College, 2011

Supervisory Committee

Dr. Fayez Gebali, Supervisor
(Department of Electrical and Computer Engineering)

Dr. Atef Ibrahim, Co-Supervisor
(Department of Electrical and Computer Engineering)

Abstract

The search for developing techniques of a digital circuit that assist in minimizing the challenges that occur nowadays in microelectronics is continuous. Arithmetic circuits especially adder target of a continues effort that presents an interesting research field. Since adder occupies a critical position in arithmetic circuits design, it is important to ensure that the performance of adder meets certain specifications. In this report, we introduce different fast adders topologies with focus on computation speed parameter. In addition, we provide delay modeling and simulation of the fast adders to check their performance using Matlab software environment. The main goal is to provide a comprehensive review of fast adders and some new techniques that have been applied to improve their performance.

Contents

List of Tables

List of Figures

Acknowledgements

First of all, my greatest gratitude is to my country Saudi Arabia and especially the King Abdullah Scholarship Program (KASP) and ministry of education for sponsoring my studies at the University of Victoria. Also, my deep thank is to my supervisor Dr. Fayez Gebali, for his support, feedback, guidance, and encouragement during my study.

I would like to thank my supervisory committee for their essential feedback. I thank all the members and staff of the electrical and computer engineering department at the University of Victoria whom I had interesting discussions allowing me to achieve my goal. Finally, a lot of appreciation to my family and friends for their unceasing support and warm feelings during my study in Canada.

THANK YOU ...

Acronyms

ALU ..... Arithmetic Logic Unit

ASIC ..... Application-specific integrated circuit

BEC ..... Binary to Excess-1 Converter

BKA ..... Brent-Kung Adder

CLA ..... Carry Lookahead Adeer

CMOS ..... Complementary Metal Oxide Semiconductor

CPL ..... Complementary Pass Transistor Logic

CSA ..... Carry Select Adder

DPL ..... Double Pass Transistor Logic

FA ..... Full Adder

FPGA ..... Field Programmable Gate Array

HA ..... Half Adder

HCLA ..... Hierarchical Carry Lookahead Adder

KSA ..... Kogge-Stone Adder

LSB ..... Least Significant Bit

MSB ..... Most Significant Bit

RCA ..... Ripple Carry Adder

VHDL ..... Very High Speed Integrated Circuit Hardware Description Language

VLSI ..... Very Large Scale Integration

Chapter 1

Introduction

Computer industry creates every time faster-computing components due to the advances in Integrated Circuits (ICs) technology. Every time better chip fabrication and sophisticated computing design techniques are available to be used which create architectural innovation with an efficient use of technology improvements. However, with several improvement techniques of integrated circuits, parameters including latency, power consumption, silicon die and temperature of the chip are continuing to be major challenging of integrated circuit designers.

The adder is an essential computing component that is discussed widely in digital circuit design.Various applications including a high-performance processor, embedded processors, and DSP processors rely on the adder component in order to perform algorithm operations in their arithmetic logic units (ALUs). Adders also used in other parts of the processor to calculate addresses, table indices, increment and decrement operators. Due to this reason, addition is the most frequently used operation in a digital circuit that represents the main role for the computational speed of a system.

Since adder occupies a critical position inside ALU of microprocessors, it is very important to ensure that its performance adequately meets given specifications for speed, area, and power consumption of different adders topologies such as ripple carry adder [16], carry lookahead adder [16], carry save adder [10, 23, 27], carry skip adder [32, 17], carry select adder [6, 30, 29], parallel prefix adders [8, 34, 12, 18] and hybrid adder [24, 36] etc.

1.1 Motivation

Arithmetic Logic Unit (ALU) is known as the heart of every microprocessor which determines its throughput. Adder is considered the core of ALU. Since adder is targeting an essential position in every microprocessor, it is important to ensure that the performance of adder meets certain specifications in terms of delay, chip area, and power consumption.

The aim of this project is to provide a thorough study of several adder architectures include Ripple Carry Adder (RCA), Carry Lookahead Adder (CLA), Hierarchical Carry Lookahead Adder (HCLA), Carry Select Adder (CSA), Kogge-Stone Adder (KSA), and Brent-Kung Adder (BKA). The study addresses adders structures, modules and circuits design, models, and algorithms. In addition to that, we introduce recent design techniques that assists to reduce delay, area, power, and enhance the overall performance of these adders.

1.2 Previous Work

Adder is considered one of the most essential units in digital circuits design. An example of the adder that is widely discussed is RCA, which is among the most straightforward implementation [16]. RCA is a commonly used adder because it is simple to implement and an efficient adder in terms of area and power consumption. Although RCA has these advantages, it is very slow which then affects the overall adder performance. The low performance of this implementation is the result of the delay associated with propagating the carry signal through the chain of full adders (FAs) in a sequential manner.

Since RCA has the delay issue, designers have proposed different adder architecture called Carry Select Adder (CSA) [6]. CSA generally consists of two RCAs and multiplexer. Adding two n-bit numbers with a CSA is done with two RCAs to perform the calculation twice, one time with the assuming the carry is being zero, and the other is one. After both results are calculated, the correct sum as well as carry, are then selected by the multiplexer. Thus, CSA saves more time than RCA to produce the output.

Another proposed architecture for reducing RCA delay is Carry Lookahead Adder (CLA). The CLA improves the speed by reducing the amount of time required to determine the carry bits [5]. Instead of timing overhead associated with rippling carry through cascaded FAs, CLA relies on extra logics to carry signals faster. An example of CLA adder is Kogge-Stone Adder (KSA) and Brent-Kung Adder (BKA). Although CLA is preferable adder in terms of speed, it comes with extra logic gates which caused for occupying a huge area of the chip, power consumption, and high-temperature issues.

Hence, hardware designers usually contribute to improve the performance of adders using different approaches. The following paragraphs discuss recent contributions for enhancing these adders with attention to the speed, area and power consumption.

RCA adder has been implemented using different approaches and technologies. The author in [35] designed a powerful and efficient n-bit ripple carry adder by applying a new 3-dot based Quantum-Dot Cellular (QDC) architecture. The proposed 32-bit RCA structure improves 21 % on QDC compared to traditional structure. The author in [3] shows demonstrating on self-checking RCA on how ambipolar design style can assist reducing the hardware overhead. The design shows a reduction of at least 56% in area, and 62% of delay compared with equivalent CMOS process.

Regarding CSA, the author in [37] proposed an area-efficient CSA by sharing the common Boolean logic term, and was able to reduce the area of conventional CSA from 1947 to 960 and power consumption from 1.26mw to 0.37mw. In [26], the work uses a simple and efficient gate-level modification to reduce the area and power of the CSA. The modified CSA produced 17.4 % smaller area and 15.4 % less power consumption compared to traditional CSA. Another technique is used in [19] to eliminate all the redundant logic operations in the conventional CSA and proposed a new logic formulation for CSA. Results show that the proposed CSA involves nearly 35% less area_delay product compared to 48% of CSA based BEC design. The author in [21] modified CSA architecture using D-latch instead of BEC. Besides, the work compared conventional CSA architecture, CSA based BEC and CSA based D-latch. Results show that the CSA based D-latch has less delay and power consumption compared to conventional CSA and CSA using BEC.

CLA adder issues and proposed solutions have also been addressed. In [31], the author used different static and dynamic logic styles such as Standard CMOS, DCVS Pseudo NMOS, PTL and Domino logic style. The results show average power and PDP for sum and carry are found a minimum in case 4-bit standard CMOS CLA implementation. In addition, reference [5] shows implementing and evaluation of CLA using different radix-n to reach the best value of n. Results discussion show that CLA using radix-2 produced 10.6 % less delay, and 10.41 % smaller area compared to commonly used radix-4.

Furthermore, other fast adders such as KSA and BKA have also been taken into consideration. In [7], the author designed carry select adder using kogge-stone adder and compared with the traditional kogge-stone adder. CSA with Kogge Stone network produced 3.53 % less delay compared to traditional KSA. The reference [20] enables the designers to choose the best configuration to meet specifications in terms of delay area and power consumption of KSA. The author in [33] designed KSA and BKA based on degenerate pass transistor logic (PTL) and compared to CPL and DPL pass transistor logic. The results illustrate that BKA has a smaller area and power consumption compared to KSA. Evaluating performance of 4-bit BKA using various transistors gate sizing, basic logic gate and a compound gate is mentioned in [4]. The results show that compound gate has 35% less power, and 19.16% smaller delay compared to the basic logic gate.

1.3 Report Organization

This report is organized as follows:

In Chapter 1, we provide motivation and summary of related work. The related work is targeting different fast adders including recent techniques that have been applied to enhance the performance of these adders.

In Chapter 2, we discuss a basic concept of addition process and provided a comprehensive study of different fast adders. The study includes RCA, CLA, HCLA, CSA, KSA, and BKA. We mainly focused on the architectural environment of these adders, basic module design, structure style, gate level implementation, circuit analysis, modeling, and prefix adders algorithms.

In Chapter 3, we conclude the report and provide suggestions for future work.

Chapter 2

Adders Theory

2.1 Adders Background

An adder is a digital logic circuit that executes an arithmetic operation such as addition, subtraction. It is also used in processor to calculate table indices, addresses, and similar operation. In this chapter, we discusse different adder types such as ripple carry adder, carry lookahead adder, hierarchical carry lookahead adder, carry select adder, kogge-stone adder, and brent-kung adder.

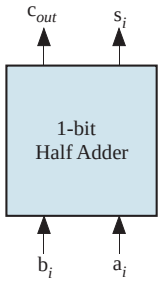
The basic adder types are half adder (HA) and full adder (FA) as shown in Figure 2.1. Half adder basically adds two binary digits a and b and produce two output signals sum s and carry c. The carry signal indicates an overflow into the next digit of a multi-digit addition. HA can be implemented using XOR gate and AND gate according to equations 2.1 and 2.2. In contrast, FA adds three binary digits, often written as a, b and c, and produce two output signals sum s and carry c. FA can be implemented using equations 2.3 and 2.4.

$$s = a \oplus b \tag{2.1}$$

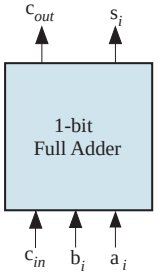
$$c_{out} = a \cdot b \tag{2.2}$$

$$s = (a \oplus b) \oplus c_{in} \tag{2.3}$$

$$c_{out} = (a \oplus b) \cdot c_{in} + a \cdot b \tag{2.4}$$



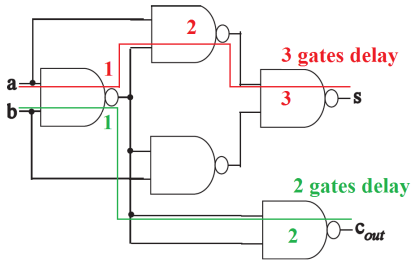
(a) Half adder



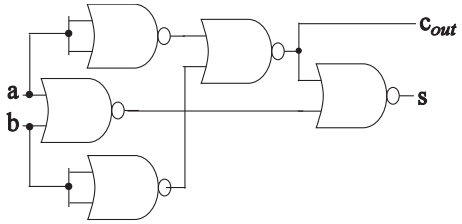
(b) Full adder

Figure 2.1: Basic adders modules

Interestingly, digital arithmetic circuits such as adder are usually constructed with NAND or NOR gates instead of AND and OR gates. NAND and NOR gates are used in all integrated circuit families and easier to fabricate with electronics components. In addition, it is easy to convert NOT, AND, and OR gates into an equivalent NAND and NOR logic diagrams. Figure 2.2 is an example of half adder using NAND and NOR gates implementation. Figure 2.3 shows a full adder implementation using NAND and NOR gates.

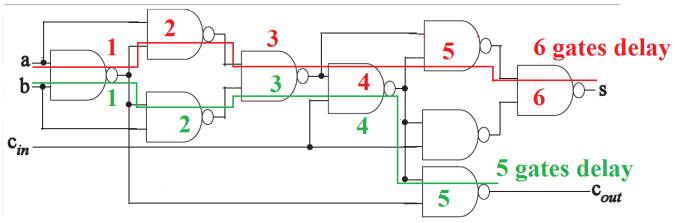


(a) NAND gate implementation

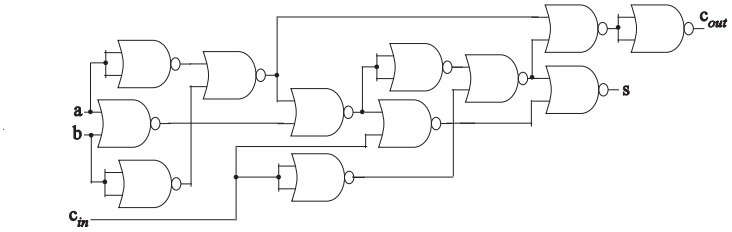


(b) NOR gate implementation

Figure 2.2: Half adder implementation using NAND and NOR gates [1]



(a) NAND gate implementation



(b) NOR gate implementation

Figure 2.3: Full adder implementation using NAND and NOR gates [1]

2.1.1 Delay Modeling

1. We normalize gate delay relative to 2-input NAND gates.
2. The normalized delay for the i-input NAND gate is given by $T_i = \lceil log_2 i \rceil$.
3. The normalized delay for 2-XOR gate ($T_x = 3$), see Figure 2.2 (a).
4. The normalized delay for 1-bit full adder stage ($T_c = 5$), see Figure 2.3 (a).
5. The normalized delay for a multiplexer ($T_m = 3$).

2.1.2 Carry Propagation and Generation

The parallel addition of two binary numbers means that all the bits of the addend and augend are available for computation simultaneously. Therefore, the principle of propagate (p) and generate (g) are commonly used in parallel addition, which are defined in terms of a single digit and do not rely on any other digits in the sum.The generate method occurs when both addend and augend are '1' which express as $g = a{\cdot}b$ while the propagate method occurs if and only if at least one of a or b is '1' which express as $p = a \oplus b$. Given these methods of generation and propagation, it will carry exactly when either the addition generates, or the least significant bit carry and the addition propagates. Written in boolean algebra as $c_{i+1} = g_i + (p_i \cdot c_i)$.

2.2 Ripple Carry Adder (RCA)

RCA is a cascade of full adders that is connected in serial so that the carry can propagate through every full adder before the process of addition is completed. As shown in Figure 2.4, the first full adder's output is connected as an input to the second full adder, and the second full adder's output is connected as an input to the third full adder, etc. This type of adder is called RCA since each carry bit ripples to the next full adder from the least significant bit (LSB) position to the most significant one(MSB).

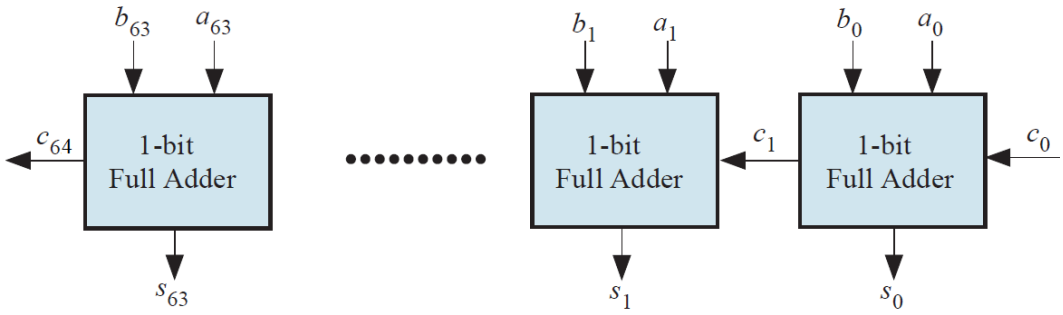


Figure 2.4: 64-bit RCA

The full adder circuit can be implemented by constructing two half adders and one OR gate according to the equation 2.5 and 2.6.The XOR output of the first half adder is considered as an input to the second half adder. The s output of the second half adder is the exclusive-OR of c_{in}.

$$s = (a \oplus b) \oplus c_{in} \tag{2.5}$$

$$c_{out} = (a \oplus b) \cdot c_{in} + a \cdot b \tag{2.6}$$

In the RCA, the output is known after the carry generated by the previous stages. Hence, the sum of the most significant bit (MSB) is valid after rippling the carry signal through the adder from the least significant bit (LSB) stage to the most significant bit stage. So the final sum and carry out will be available after a considerable delay.

This delay is proportional to the number of N-bits which is given approximately by equation 2.7.

$$T_{RCA} = NT_c \tag{2.7}$$

Where, N is the input data, T_c is the delay for full adder single stage which equles to 5, see Figure 2.3 (a). So we can also write the equation of CLA as 5N. In the case of Figure 2.4, the c_{in} is equal to c_0 which is the least significant bit, and the c_{out} is equal to c_{63} which is considered the most significant bit. It is noticed from the equation above that the delay increases linearly with the N-bit. Hence, this kind of adder design can not be used in the high-performance processor which is designed for large N-bit.

2.3 Carry Select Adder (CS)

The Carry Select Adder (CS) consists of two Ripple Carry Aadders (RCAs) and a multiplexer. In order to add two n-bit numbers, we need two ripple carry adders to perform the calculation twice[11]. The first time we assume the carry-in being zero and the second time is one. After the two results are calculated, we get the correct sum and correct carry and then selected by the multiplexer. Figure 2.5 shows 4-bit carry select adder. The 1-bit multiplexer for sum selection can be implemented as Figure 2.6.

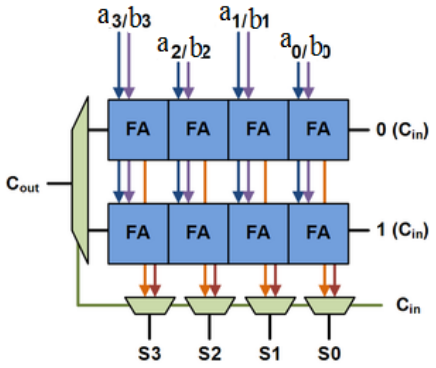


Figure 2.5: 4-bit CS

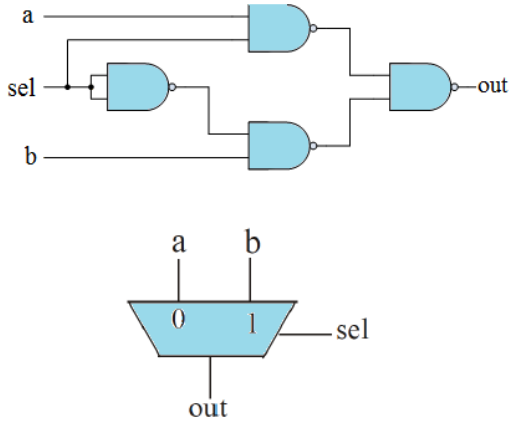


Figure 2.6: 1-bit multiplexer circuit and module

Figure 2.5 is the basic architecture of 4-bit carry select adder. Two 4-bit RCAs are multiplexed together, where the resulting sum and carry bits are selected by the c_{in}. The carry select adder is basically proposed to improve the shortcoming of RCA to remove the linear dependency between computation delay and input word length. CS divides the RCA into M groups, while each group consists of a duplicated (N/n)-bit RCA pair as illustrated in Figure 2.7. Hence, the N-bit CS delay can be calculated as follows:

$$T_{CS} = T_c \times n + (N/n)T_m \tag{2.8}$$

Where, N is the input data, and n is the radix in each stage. T_m is multiplexer delay which equals 3 gates delay.

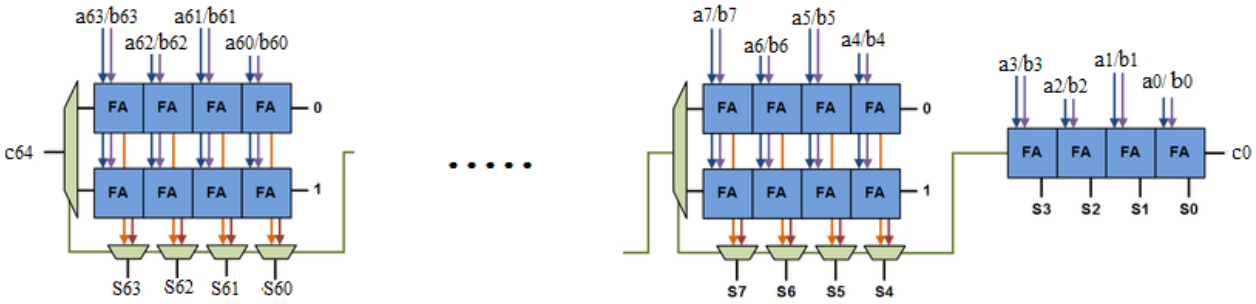


Figure 2.7: 64-bit uniform-size CS

2.4 Carry Lookahead Adder (CLA)

A significant performance improvement for the RCA implementation is a parallel adder was developed by Weinberger and Smith in 1958 called Carry Lookahead Adder (CLA)[22]. The CLA is one of the fastest adder design used for adding two numbers. The CLA delay no longer depends on N-bit. Instead, carry outputs are calculated in advance based on generate (g) and propagate (p) signals. For example, output of 4-bit carries can be computed based on the following equations:

$$\begin{aligned} c_0 &= c_{in} \\ c_1 &= g_0 + p_0.c_0 \\ c_2 &= g_1 + p_1.g_0 + p_1.p_0.c_0 \\ c_3 &= g_2 + p_2.g_1 + p_2.p_1.g_0 + p_2.p_1.p_0.c_0 \end{aligned}$$

We can generalize the above equations as mentioned in [9]. The carry at bit i can be written as:

$$c_i = c_{in} \prod_{j=0}^{i-1} p_j + \sum_{j=0}^{i-1} g_j \prod_{k=j+1}^{i-1} p_k, \qquad 0 \leq i \leq n \tag{2.9}$$

Typically, CLA can be constructed using two levels. The first level is called a Partial Full Adder (PFA) as shown in Figure 2.9. This part is responsible for generate and propagate the carry to the second level.

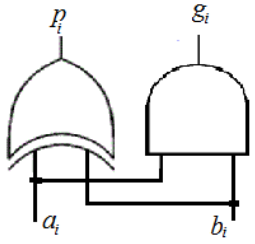


Figure 2.8: Partial full adder circuit

For N-bit CLA, the two n-bit inputs $a[n-1:0]$ and $b[n-1:0]$ to be added are used to generate the carry propagate $p[n-1:0]$ and carry generate $g[n-1:0]$ signals to be supplied to the CLA at bit i according to equations 2.10 and 2.11.

$$p_i = a_i \oplus b_i \tag{2.10}$$

$$g_i = a_i \cdot b_i \tag{2.11}$$

The output sum can be expressed according to equation 2.12, where c_i is the carry output of each stage.

$$s_i = p_i \oplus c_i \tag{2.12}$$

Figure 2.10 shows the second level of CLA which is called carry lookahead module. It is responsible for computing output carries according to equation 2.9.

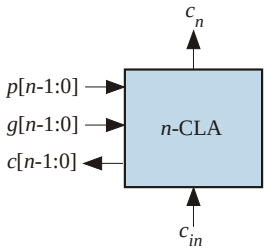


Figure 2.9: Carry lookahead basic module [5]

Figure 2.10 shows n-bit CLA basic module which accepts two n-bit input signals p and g, and the carry-in signal c_{in} and produces $n + 1$ bits carry signal $c_0 - c_n$. The n inside the module indicates the number of radices (bits). For example, let us consider we have 4-bit CLA. The c_{in} here indicates to c_0 which is the least significant bit and c_n indicates to c_4 which is the most significant bit. Figure 2.11 shows gate level implementation of 4-bit carry lookahead module.

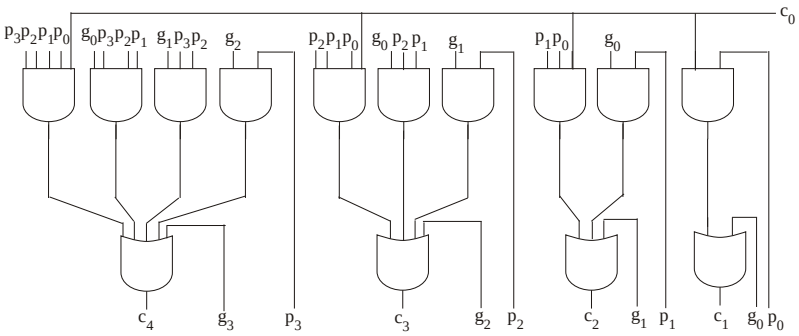


Figure 2.10: Gate level implementation of 4-bit CLA

The entire design of CLA can be constructed using either a conventional or hierarchical structure [5]. The conventional structure is the most common design which can be built according to the equation:

$$N = m \times n \tag{2.13}$$

Where N is the input data of CLA, m is the number of modules, and n is the radix in each module. Figure 2.12 shows an example of conventional 64-bit CLA with two modules using radix-32 in each module.

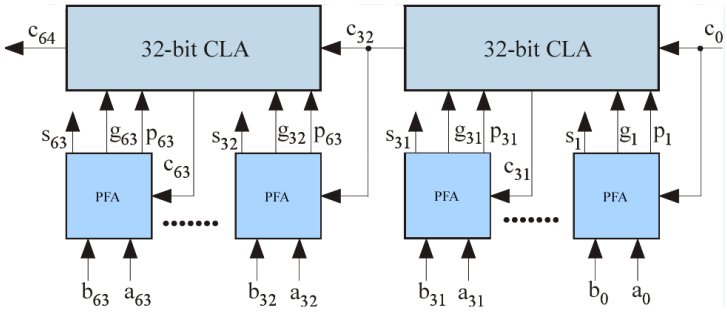


Figure 2.11: 64-Bit conventional CLA using radix-32

However, it is unlikely to create a 64-bit conventional CLA adder using a huge radix-n as shown in Figure 2.12. The number of logic gates used in each module would be significantly large resulting in a huge area and power consumption. The number of logic gates increases every time we calculate the carry-in for higher bit positions. Therefore, designers usually prefer to use a small value of n like radix-4 which is the most common design. Designing CLA with a small radix-n requires few gates implementation resulting in a small area, power consumption and faster operation. Figure 2.13 shows 64-bit conventional CLA using radix-4.

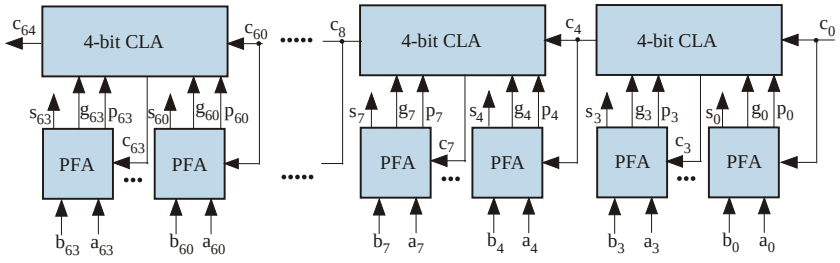


Figure 2.12: 64-Bit conventional CLA using radix-4

It is noticed from the previous paragraphs that adder performance is strongly influenced by the carry-propagating process of CLA. In order to get higher computational speed, it is important to make the sum independent from carry propagation. This is the principle of how conventional structure of CLA works. Figure 2.14 shows CMOS implementation of a 4-bit conventional CLA.

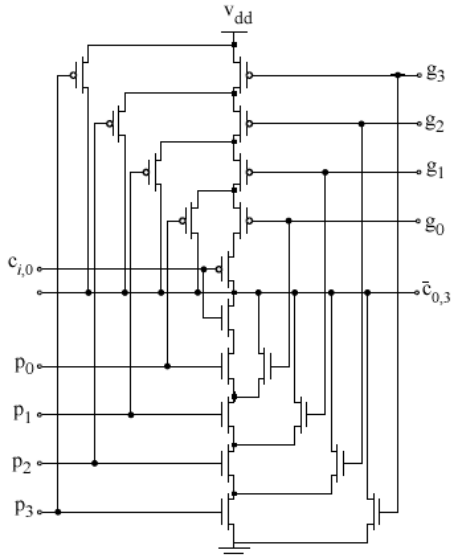


Figure 2.13: CMOS implementation of a 4-bit CLA [14]

On the other hand, when designing faster CLA adder, it is important to get around the rippling effect that occurs from one module to another one. In CLA, designers usually prefer to use radix-4 implementation in order to get faster operation and to avoid area and power consumption issues. In general, the N-bit delay of carry lookahead adder can be obtained using the following equation.

$$T_{CLA} = 2T_x + \lceil N/n \rceil \times log_2(n+1) \tag{2.14}$$

Where, N is the input data, and n is the radix in each module. T_x is the propagation and generation delay which is the same as sum delay T_s, see Figure 2.2 (a). However, the delay of rippling the carry from module to another remains an issue for CLA designers. In order to solve this problem, it is necessary to build propagation and generation into a hierarchical tree (logarithmic) [22]. CLA using hierarchical structure produces a group of carry generate, and a group of carry propagate outputs each module.

In the other words, carry lookahead modules indicate if the carry is generated within the group or the incoming carry will propagate across the group to the next module. Hence, knowing in advance if the carry is generated or propagated inside a large group of bits will assist to decrease the delay due to the carry computation [25].

2.5 Hierarchical Carry Lookahead Adder (HCLA)

A hierarchical structure is another form of constructing CLA adder. Hierarchical carry lookahead adder or simply (HCLA) builds larger adders by combining the carry lookahead modules hierarchically [9]. The entire N-bit HCLA architecture can be divided into three parts.

The first part is the partial full adder (PFA) which is dissucessed in the previous section of CLA. The second part is the n-HCLA module which is different than n-CLA module, see Figure 2.15. The third part is a small circuit that accepts p_{out} and g_{out}, c_{in} and produces c_{out}. The following paragraphs provide more details about these parts.

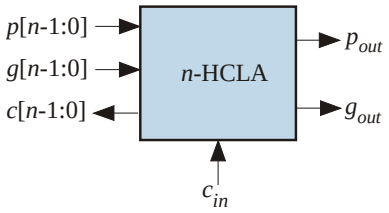


Figure 2.14: n-bit HCLA basic module [9]

In general, the n-HCLA basic module processes two n-bit signals $p[n-1:0]$ and $g[n-1:0]$ to produce two output signals: group carry propagate p_{out} and group carry generate g_{out} according to the equations:

$$p_{out} = \prod_{j=0}^{n-1} p_j \tag{2.15}$$

$$g_{out} = \sum_{i=0}^{n-1} g_i \prod_{j=i+1}^{n-1} p_j \tag{2.16}$$

Figure 2.16 shows the 4-bit HCLA circuit. It is noticeable that we add two outputs to our lookahead circuit in order to compute our carry values: carry group generate and carry group propagate. The group generate and propagate signals are still a single bit each, not a vector.

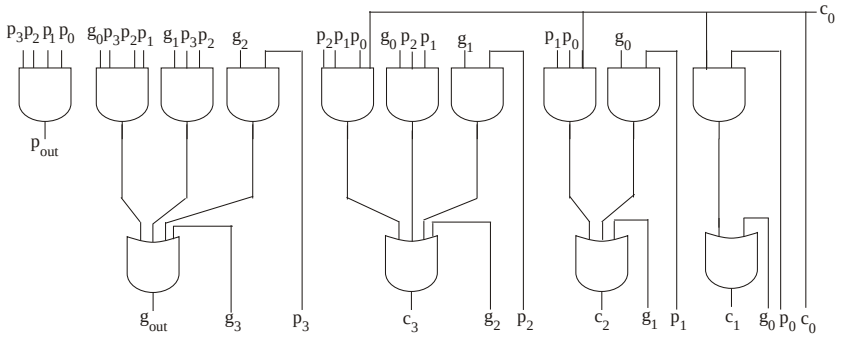


Figure 2.15: Gate level implementation 4-bit HCLA [9]

Figure 2.17 shows 64-bit HCLA. The first part of the bottom is the partial full adder. The second part in the middle is the hierarchical carry lookahead module. We use gg to indicate group generate signals and gp to indicate group propagate signals. The third part of entire N-bit HCLA architecture accepts the p_{out} and g_{out} signals from the top level, and the c_{in} signal to produce the carry out.

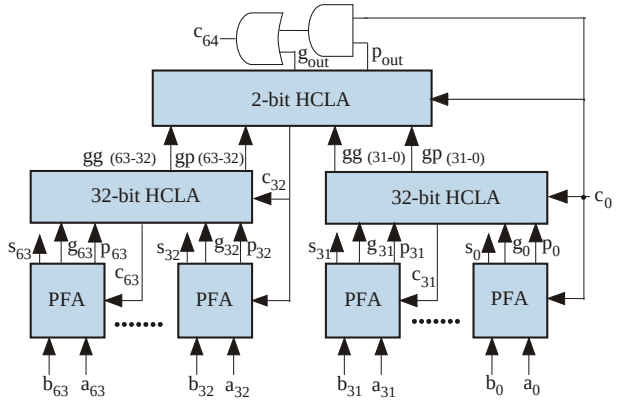


Figure 2.16: 64-bit HCLA using two levels implementation

2.5.1 Identical Radix-n HCLA

The N-bit HCLA adder can have different number of levels (h) as well as different radix-n in each level. In this report, we have only mentioned HCLA using two levels implmentation. For example, Figure 2.18 shows 64-bit HCLA using two levels implementation. The radices in both levels h_0 and h_1 are the same (8-bit); that is the meaning of identical radix-n.

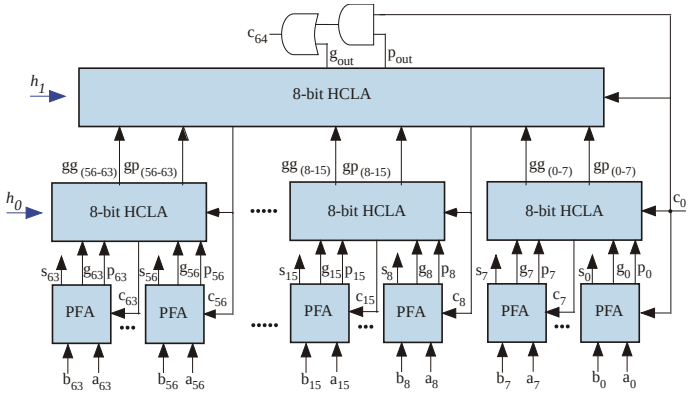


Figure 2.17: 64-bit HCLA using two levels implementation

2.5.2 Nonidentical Radix-n HCLA

The nonidentical radix-n HCLA deals with the case when the radix-n in h_0 and h_1 are not the same. Figure 2.19 shows 64-bit HCLA using nonidentical radix-n. The first level consists of four modules using radix-16 each. The second level consists of one module using radix-4.

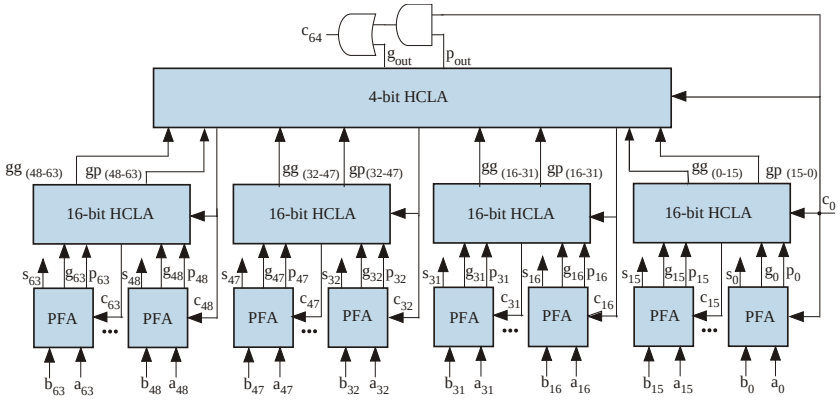


Figure 2.18: 64-bit HCLA using two levels nonidentical radix-n

For both identical and nonidentical radix-n of two levels HCLA, the N-bit HCLA structure can be built according to equation (2.17) where $n_{(h_0)}$ is radix-n in the first level, $n_{(h_1)}$ is radix-n in the second level.

$$N = n_{(h_0)} \times n_{(h_1)} \tag{2.17}$$

However, delay in HCLA is different than CLA. Total delay in HCLA using identical radix-n can be obtained according to the equation 2.18 where T_x is the same in CLA. The n_1 indicates the radix in the first level and n_0 indicates the radix in the second level. We can get the delay of HCLA using identical and nonidentical radix-n using the same equation.

$$T_{HCLA} = 2T_x + 2log_2(n_1 + 1) + log_2(n_0 + 1) \tag{2.18}$$

2.6 Parallel Prefix Adders

Parallel prefix adders classes are based on solving recurrence equations which were introduced by Kogg and Stone [13]. The parallel prefix adder employs 3-strctural stages as shown in Figure 2.20. The first stage at the top is responsible for generate and propagate the signals exactly as CLA. The stage in the middle is where the carry bits are calculated. In this stage, to formulate the operation of the prefix adders, we should know a function called "prefix operator" which represents as ".". The "." function has two essential properties to keep the computational operation faster. The first one is called the associative property and the second one is idempotency property [22]. By looking to the advantage of these properties, c_{out} can be found at a depth proportional to $log_2(N)$ [15]. The last stage in the bottom is where we get our final summation.

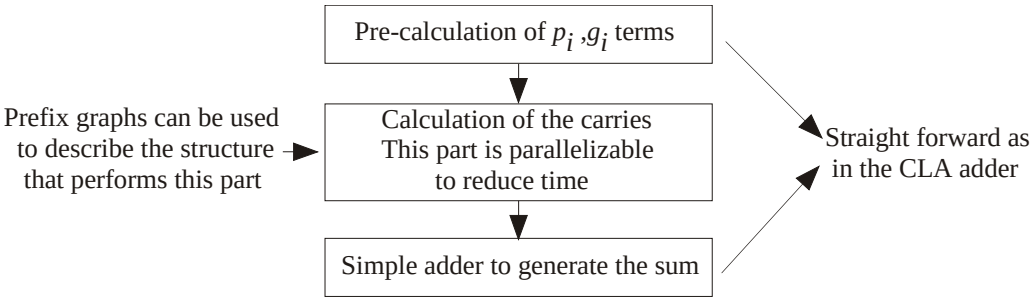
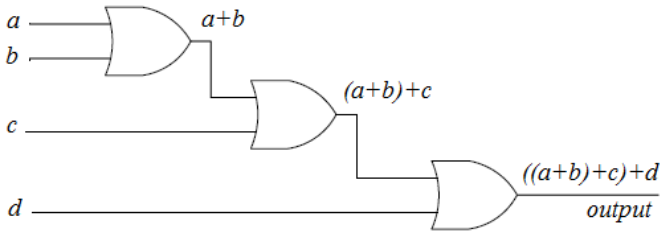


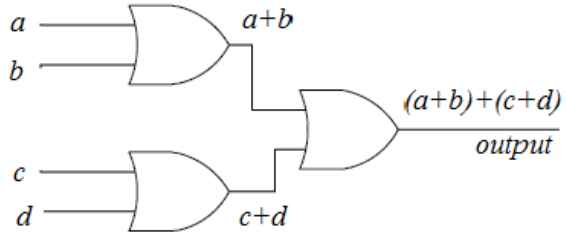
Figure 2.19: Parallel-prefix adder stages

Figure 2.21 shows a practical example of the prefix operator. Part (a) shows a serial OR gate implementation. It is noticeable that the output summation will be ready after three levels (stages). In contrast, part (b) shows a parallel OR gate implementation. We only have two stages to get the output. Hence, the associative property we used it in (b) is able to give us the least delay.

The associative and idempotency properties of "." operator allow carry output to be computed in a different number of levels or simply depth. Therefore, we can design various topologies of prefix adders which are mentioned in the literature and are inspiring to VLSI designers because of their minimum depth and delay. The main advantages of these adders are the good layout, and ability to make trade-offs between fan-in and fan-out, so fan-in can be controlled in no more than two.



(a) OR gate serial implementation



(b) OR gate parallel implementation

Figure 2.20: Associative operations are parallelizable

Figure 2.22 shows the definition of cells including black cell, gray cell, and white cell. These cells and their logical structures below them are used in prefix adders scheme [28]. The logical structure of black cell can propagate and generate signals while the logical structure of gray cell can only genrate signals. The white cell (buffer) is loading the signal out. In parallel prefix method, the generate and propagate signals can be grouped in various fashion to get the same correct carries. Based on different methodologies of grouping these signals, different prefix architectures can be created.

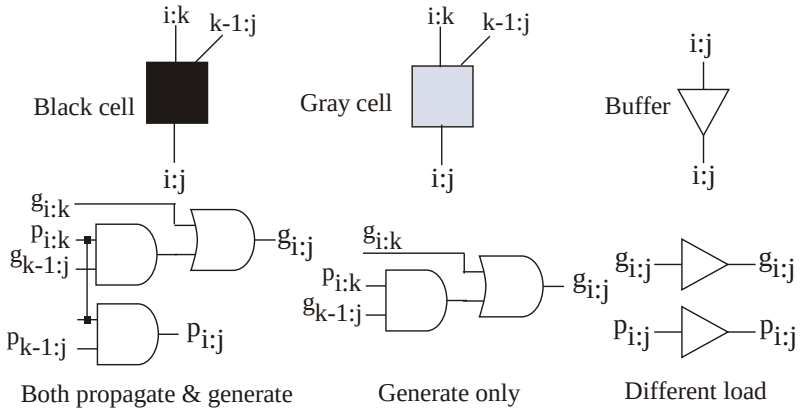


Figure 2.21: Cell definition for parallel-prefix scheme [28]

2.6.1 Kogge-Stone Adder (KSA)

Kogge-Stone Adder (KSA) is a parallel prefix adder that was developed by Peter M. Kogge and Harold S. Stone in 1973 [28]. KSA is widely considered as a standard adder in the industry for high performance arithmetic circuits [2]. The KSA computes the carries in parallel and takes more area to implement, but has a lower fan-out at each stage, which then increases performance adder. KSA can be easily implimented by analyzing it in terms of three parts:

- Pre-processing

This step includes computation of generate and propagate signals that corresponding to each pair of bits in a and b. The generate and propagate signals are given by the equations below:

$$p_i = a_i \oplus b_i \tag{2.19}$$

$$g_i = a_i \cdot b_i \tag{2.20}$$

- prefix carry tree

This part differentiates KSA from other adders and is the reason behind its high performance. This step includes computation of carries that corresponding to each bit. This part uses group propagate and generate signals which are given by the equations below:

$$p_{i:j} = p_{i:k+1} \cdot p_{k:j} \tag{2.21}$$

$$g_{i:j} = g_{i:k+1} + (p_{i:k+1} \cdot g_{k:j}) \tag{2.22}$$

The notations $p_{i:j}$ and $g_{i:j}$ denote to group-propagate and group-generate respectively and for the group that includes bit positions from i to j. k represents the logic level from where the input is produced.

The computation p and g can be performed by extending the tree structure shown in Figure 2.23, so that it is possible to obtain $(p_{i:j}, g_{i:j})$ for every i and j. This process is illustrated in Figure 2.22.

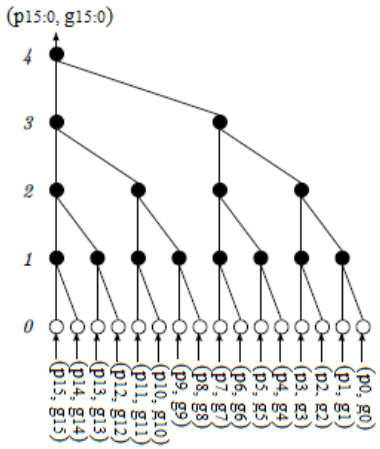


Figure 2.22: Computation of (p_{15}, g_{15}) using a tree structure. [28]

As we have explained in the previous paragraph, the KSA occupies a big area to implement but has a lower fan-out at each stage. The big area of KSA is due to wiring complexity in this design. An example of the 8-bit kogge stone adder tree is shown in Figure 2.24. Every vertical stage produces propagate and generate signals as shown. The carry bits are produced in the last stage, and XOR'd with the previous propagate signals to produce the sum bits. That is more obvious when we see Figure 2.25 of KSA gate level implementation.

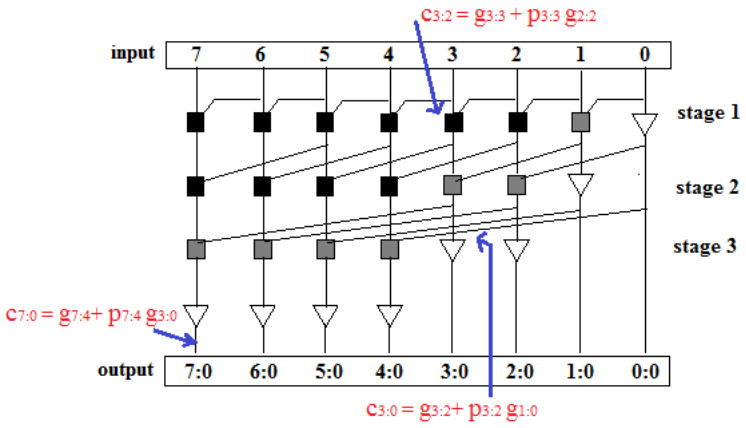


Figure 2.23: 8-bit Kogge-Stone adder tree [28]

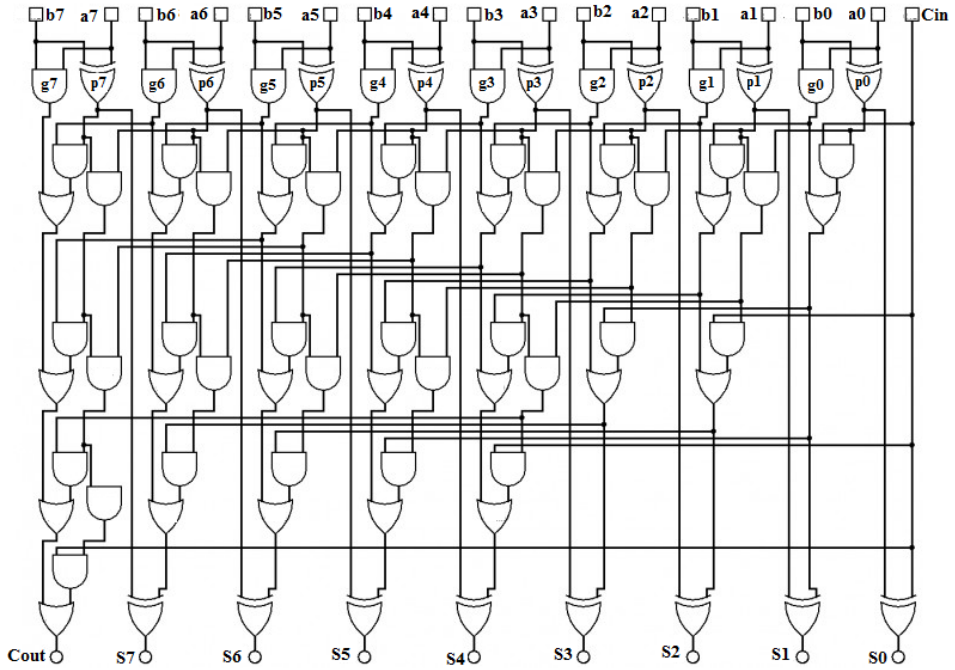


Figure 2.24: Gate implementation of 8-bit Kogge-Stone adder [2]

- post processing

This step is the final step to all adders of the (carry look ahead) family. It includes computation of sum bits which is given by the equation below:

$$s_i = p_i \oplus c_i \tag{2.23}$$

2.6.2 Brent-Kung Adder (BKA)

Brent-Kung Adder (BKA) is a parallel prefix adder that was developed by Brent and Kung in 1982 [28]. The idea of Brent and Kung adder is to combine propagate signals and generate signals into groups of two by using the associative property only [22]. BKA is also belonged to carry lookahead adder family and can be implemented by analysing it into three parts as following:

- Pre-processing

This step includes computation of generate and propagate signals that corresponding to each pair of bits in a and b, and it is the same process of other CLA family. The generate and propagate signals are given by the equations below:

$$p_i = a_i \oplus b_i \tag{2.24}$$

$$g_i = a_i \cdot b_i \tag{2.25}$$

- prefix carry tree

This part includes computation of carries that corresponding to each bit, and it is different from other CLA family. The equations below shows how propagate and generate signals are calculated in BKA. Also the equations are applied to BKA tree in Figure 2.26.

$$p_{i:j} = p_{i:k} \cdot p_{k-1:j} \tag{2.26}$$

$$g_{i:j} = g_{i:k} + (p_{i:k} \cdot g_{k-1:j}) \tag{2.27}$$

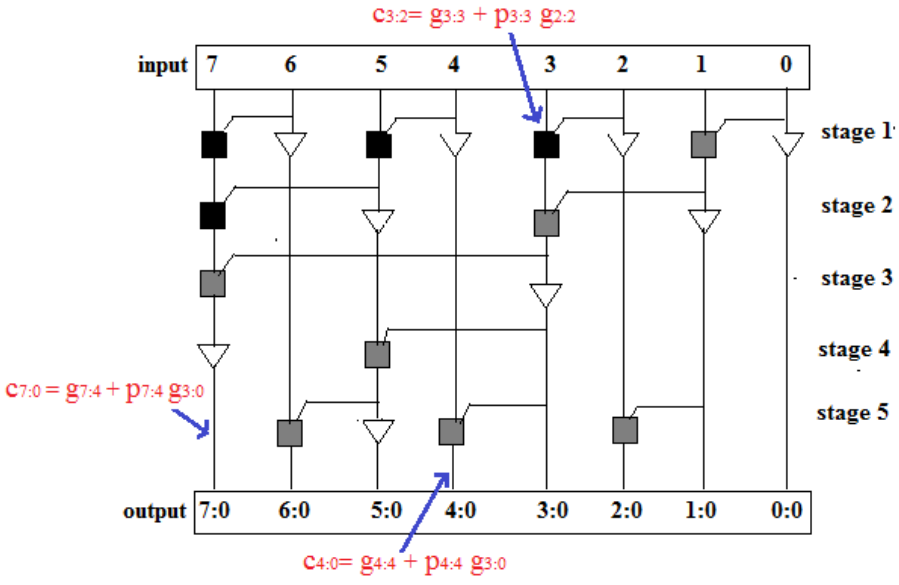


Figure 2.25: 8-bit Brent-Kung adder tree [28]

BKA avoids the complexity of wires resulting in smaller area and power consumption. BKA has maximum logic depth, limited fun-out to two at each stage. The smart idea of this design is to compute prefixes for 2-bit groups first. These are then used to find prefixes for 4-bit groups, and turn to find prefixes for 8-bit groups, etc. The issue of this design is that the propagate and generate signals take more stages than KSA to be calculated. Figure 2.26 shows 8-bit BKA tree and the logic gate implementation of this tree is shown in Figure 2.27.

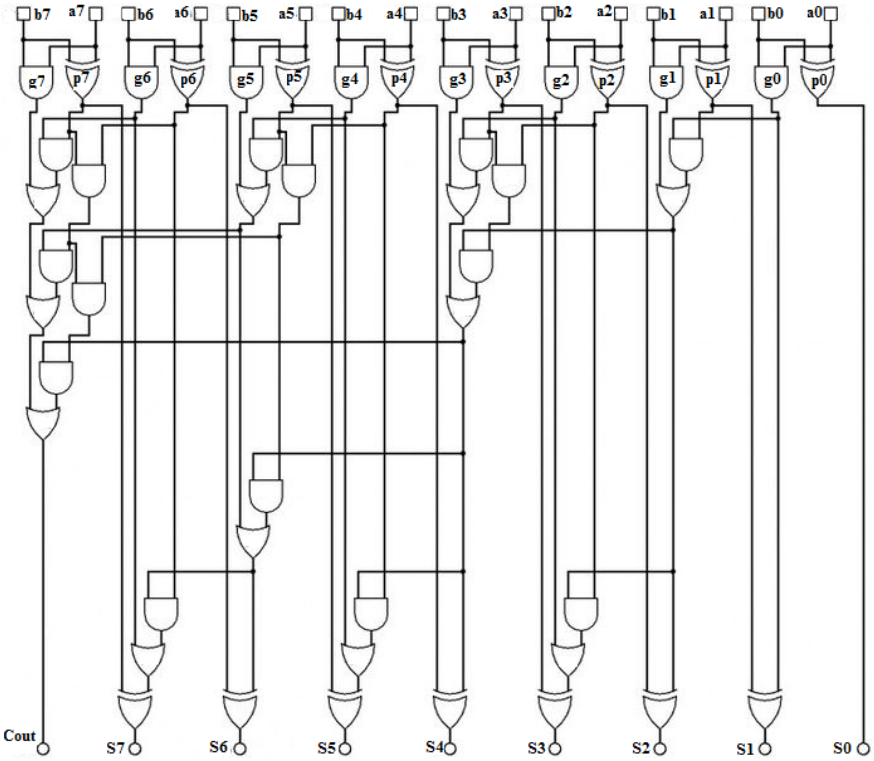


Figure 2.26: Gate implementation of 8-bit Brent-Kung adder [2]

- post processing

This step includes computation of sum bits which is also the same process of another adder of CLA family. This process is given by the equation below:

Brent-Kung Adder features with low network complixity comparing to Kogge-Stone Adder. The low network complexity assists to reduce the area of adder resulting in reducing the power consumption as well. This feature makes BKA more efficient than KSA, which has more black competition nodes and long wires. On the other hand, BKA has more stages (logic levels) compare to KSA. Having more competition stages leads to a slower adder. For example, as shown in Figure 2.24, 8-bit KSA needs only three stages to calculate the carries while BKA in Figure 2.26 needs five stages to get the carries calculated. Hence, KSA is more efficient than BKA in terms of speed.

2.6.3 Prefix Adders Algorithms

The performance of parallel prefix adders makes them particularly inspiring for VLSI implementation. The parallel prefix adder structures that have been proposed are optimized for their logic depth, fan-out area, and interconnect count of the logic circuits. The delay of Kogge-Stone adder and Brent-Kung adder can be obtained according to the equation $T = log_2 N$. Table 2.1 summarizes the algorithms of parallel prefix adder including logic levels, area, fan-out and wire tracks. We are only focusing on KSA and BKA in this literature.

Table 2.1: Parallel Prefix Adders Algorithms [28]

Types	Logic Levels	Area	Fan-out	Wire tracks
Brent-Kung	$2log_2 n - 1$	$2n - log_2 n - 2$	2	1
Kogge-Stone	$log_2 n$	$nlog_2 n - n + 1$	2	$n/2$
Ladner-Fischer	$log_2 n + 1$	$(n/4)log_2 n + 3n/4 - 1$	$n/4 + 1$	1
Knowles	$log_2 n$	$nlog_2 n - n + 1$	3	$n/4$
Sklansky	$log_2 n$	$(n/2)log_2 n$	$n/2 + 1$	1
Han-Carlson	$log_2 n$	$(n/2)log_2 n$	2	$n/4$

The ideal N-bit parallel prefix adder tree would have:

- Logic Level= $\log n$
- Fan-out= 2
- Wire track= 1

2.7 Delay of Binary Adders Topologies

Delay is an important parameter to check the performance of the circuit design. The delay determines how long it takes for data to transfer from the input to the output. It may vary slightly or sharply depending on the number of bits (N), the logic levels, the number of radix-n, and structure style of the adder, etc. In this report, we have generated a Matlab code based on the adders modeling to estimate the normalized delay when we vary the number of bits (N).

2.7.1 Matlab Code

The programming environment for estimating the delay of adders is based on Matlab software. First, we create the main file to define the important parameters include the radix-n (n); we choose radix-4 for all adders design. In addition, we define the total number of bits (N), delay of carry out of full adder (T_c), delay of XOR gate (T_x), and delay of the multiplexer (T_m). Second, we create the main loop of the adders function as shown in Listing 1. The code in Listing 2 is to plot figure to show the normalized delay of adders. The Y-axis shows the normalized delay corresponding to the number of bits in the X-axis.

Listing 1: Matlab Code Main File

```
clc
clear all
close all

% Main parameters
N = [4 8 16 32 64 128]; % Total number of bit
points = length(N);
n = 4;   % radix
n0 = ceil(N/n);
T_c = 5; % delay of carry out of the full adder
T_X = 3; % delay of XOR gate
T_m = 2;
%---------------

T_RCA = zeros(1, points);
T_CS = zeros(1, points);
T_CLA = zeros(1, points);
T_HCLA = zeros(1, points);
T_KS = zeros(1, points);

% main loop
for i = 1:points
    T_RCA(i) = func_T_RCA(N(i),T_c);
    T_CS(i) = func_T_CS(N(i),n, T_c,T_m);
    T_CLA(i) = func_T_CLA(N(i),n, T_X);
    T_HCLA(i) = func_T_HCLA(n, n0(i),T_X);
    T_KS(i) = func_T_KS(N(i),T_X);
end
```

Listing 2: Plot Figure. Main File

```
30
31 -    figure
32 -    hold on
33 -    box on
34
35 -    set(gcf,'DefaultLineLineWidth',2)
36 -    set(gca, 'FontSize', 16)
37 -    semilogy(...
38          (1:points),T_RCA ,'-k ', ...
39          (1:points),T_CS ,'-B', ...
40          (1:points),T_CLA ,'-R', ...
41          (1:points),T_HCLA ,'--G', ...
42          (1:points),T_KS ,'-G ')
43
44 -    set(gca, 'FontSize', 16,'fontweight', 'bold')
45 -    xlabel('Number of bit','FontSize',16)
46 -    ylabel('Delay','FontSize',16)
47 -    legend('RCA','CS','CLA','HCLA','KS');
```

The code below shows adders functions that are developed for different fast adders topologies based on the adders modeling discussed in the previous sections. Each adder function should be in a separate file. However, we combine these functions into one file just to show how the adders models have been created in the Matlab software environment.

Listing 3: Adders Functions

```
1   function T = func_T_RCA(N,T_c)  % RCA function
2
3 -   T = N*T_c;
4
5     %-----------------------------
1   function T = func_T_CS(N,n, T_c,T_m)  % CS function
2
3 -   T = T_c*n+((N/n)-1)*T_m;
4
5     %-----------------------------------
1   function T = func_T_CLA(N,n, T_X) % CLA function
2
3 -   T=  2*T_X+ ceil(N/n)* log(n+1);
4
5     %--------------------------------
1   function T =  func_T_HCLA(n, n0,T_X)     % HCLA function
2
3 -   T=  2* T_X + 2*log(n+1) + log(n0+1) ;
4
5     %-----------------------------------
1   function T = func_T_KS(N,T_X)  % KS function
2
3 -   T=  2* T_X * log (N) + T_X;
4
5     %----------------------------
```

2.7.2 Adders Delays Simulation and Analysis

The purpose of the simulation is to verify the adders performance by evaluating the normalized delay. Each adder circuit is simulated with a different number of bits include 4, 8, 16, 32, 64, and 128. Matlab software is launched to generate the code and to provide the results as shown in Figure 2.28. The results show the normalized delay for different adders topologies include Ripple Carry Adder (RCA), Carry Select Adder (CS), Carry Lookahead Adder (CLA), Hierarchical Carry Lookahead Adder (HCLA), and Kogge-Stone Adder (KS). First, RCA delay increases dramatically with increasing the number of bits and has the worst delay among other adders topologies. The CS adder delay increases gradually with rising the number of bits, but still has better performance than RCA. For carry lookahead adder families, CLA produces the least delay for a small number of bits (8-bit). The KS adder produces smaller delay for a big number of bit (64-bit) compared to CLA. HCLA has a slight increase of delay and considered the most effecient adder topology among all.

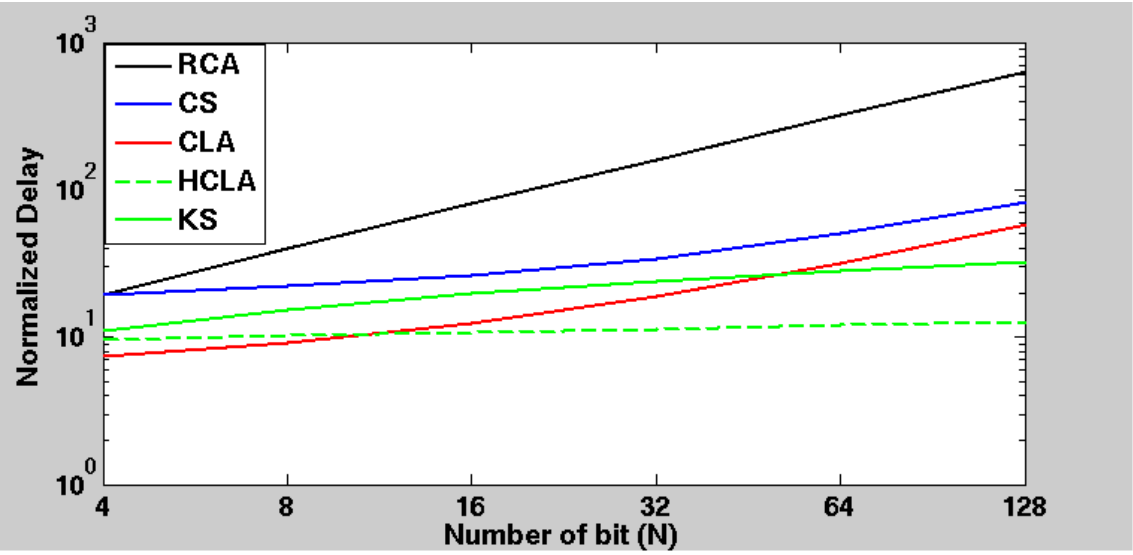


Figure 2.27: Normalized Delay of Adders

Chapter 3

Conclusion

3.1 Conclusion

In this work, various adder architectures including ripple carry adder, carry select adder, and carry lookahead adder are explained in detail. Carry lookahead adder is addressed for both conventional and hierarchical structures styles. Both of these structures are designed for identical and nonidentical radix-n. In addition to that, carry lookahead adder families architectures include conventional carry lookahead adder, hierarchical carry lookahead adder, Kogge-Stone adder, and Brent-Kung adder are reviewed. The literature includes circuits design and analysis, delay modeling, gate level implementation, parallel adders algorithms, and various structure styles in order to meet specifications in terms of computational speed performance. Adders performance are estimated and simulated using Matlab software environment.

3.2 Future Work

- The next step is to address the other binary adders topologies such as carry save adder, Ladner-Fischer adder, Han-Carlson adder, S. Knowles adder, and Sklansky-Conditional Sum adder.

- Designing and implementing of all fast adders for future testing of circuit delay, area, and power consumption are worth mentioning.

Bibliography

[1] http://www.electronicshub.org/half-adder-and-full-adder-circuits/.

[2] http://venividiwiki.ee.virginia.edu/mediawiki/index.php/ClassECE6332Fall12Group-Fault-Tolerant_Reconfigurable_PPA.

[3] Self-checking ripple-carry adder with ambipolar silicon nanowire fet. *Turkyilmaz, Ogun and Clermidy, Fabien and Amarù, Luca Gaetano and Gaillardon, Pierre-Emmanuel and De Micheli, Giovanni*, pages 2127–2130, 2013.

[4] Anas Zainal Abidin, Syed Abdul Mutalib Al Junid, Khairul Khaizi Mohd Sharif, Zulkifli Othman, and Muhammad Adib Haron. 4-bit brent kung parallel prefix adder simulation study using silvaco eda tools. *International Journal of Simulation, Systems, Science and Technology*, 2009.

[5] A Alghamdi and F Gebali. Performance analysis of 64-bit carry lookahead adders using conventional and hierarchical structure styles. In *Pacific Rim Conference on Communications, Computers and Signal Processing*. IEEE, 2015.

[6] Massimo Alioto, Gaetano Palumbo, and Massimo Poli. Optimized design of parallel carry-select adders. *Integration, the VLSI Journal*, 44(1):62–74, 2011.

[7] Nitin Anand, Greeshma Joseph, Johny S Raj, and P Jayakrishnan. Implementation of adder structure with fast carry network for high speed processor. In *Green Computing, Communication and Conservation of Energy (ICGCE)*, pages 188–190. IEEE, 2013.

[8] S Baba Fariddin and E Vargil Vijay. Design of efficient 16-bit parallel prefix ladner-fischer adder. *International Journal of Computer Applications*, 79(16):10–14, 2013.

[9] Atef Ibrahim and Fayez Gebali. Optimized structures of hybrid ripple carry and hierarchical carry lookahead adders. *Microelectronics Journal*, 46(9):783–794, 2015.

[10] R.A. Javali, R.J. Nayak, A.M. Mhetar, and M.C. Lakkannavar. Design of high speed carry save adder using carry lookahead adder. In *Circuits, Communication, Control and Computing (I4C)*, pages 33–36, Nov 2014.

[11] V Kamalakannnan, Dr PV Rao, and Veeresh Patil. Low power and reduced area carry select adder. *International Journal of Advances in Electrical and Electronics Engineering (ISSN: 2319-1112)*, 1(2):128–133, 2012.

[12] Simon Knowles. A family of adders. *Computer Arithmetic, 1999. Proceedings. 14th IEEE Symposium on*, pages 30–34, 1999.

[13] Peter M Kogge and Harold S Stone. A parallel algorithm for the efficient solution of a general class of recurrence equations. *Computers, IEEE Transactions on*, 100(8):786–793, 1973.

[14] Laureando. Study and Design of a 32-bit High-Speed Adder. Master's thesis, Universit degli Studi di Padova, 2012.

[15] Yusuf Leblebici. *CMOS Digital Integrated Circuits: Analysis and Design*. McGraw-Hill College, 1996.

[16] Mano M. Morris and Michael D. Ciletti. *Digital Design*. Pearson Prentice Hall, 2007.

[17] Santanu Maity, Bishnu Prasad De, and Aditya Kr Singh. Design and implementation of low-power high-performance carry skip adder. *International Journal of Engineering and Advanced Technology,(1)*, 4:212–218, 2012.

[18] M Moghaddam and MB Ghaznavi-Ghoushchi. A new low-power, low-area, parallel prefix sklansky adder with reduced inter-stage connections complexity. In *EUROCON-International Conference on Computer as a Tool (EUROCON)*, pages 1–4. IEEE, 2011.

[19] Basant Kumar Mohanty and Shital K Patel. Area–delay–power efficient carry-select adder. *Circuits and Systems II: Express Briefs, IEEE Transactions on*, 61(6):418–422, 2014.

[20] Zahi Moudallal, Ibrahim Issa, Moussa Mansour, Ali Chehab, and Ayman Kayssi. A low-power methodology for configurable wide kogge-stone adders. In *Energy Aware Computing (ICEAC)*, pages 1–5. IEEE, 2011.

[21] Veena V Nair. Modified low-power and area-efficient carry select adder using d-latch. *International Journal of Engineering Science and Innovative Technology (IJESIT) Volume*, 2, 2013.

[22] Shahrzad Naraghi. Reduced Swing Domino Techniques for Low Power and High Performance Arithmetic Circuits. Master's thesis, University of Waterloo, 2004.

[23] Manuel Ortiz, Francisco Quiles, Javier Hormigo, Francisco J Jaime, Julio Villalba, and Emilio L Zapata. Efficient implementation of carry-save adders in fpgas. In *Application-specific Systems, Architectures and Processors*, pages 207–210. IEEE, 2009.

[24] Vikramkumar Pudi and K Sridharan. Efficient design of a hybrid adder in quantum-dot cellular automata. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 19(9):1535–1548, 2011.

[25] A. Albert Raj and T. Latha. *VLSI Design*. PHI Learning, New Delhi-110001, 2008.

[26] B Ramkumar and Harish M Kittur. Low-power and area-efficient carry select adder. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 20(2):371–375, 2012.

[27] B Ramkumar, Harish M Kittur, and P Mahesh Kannan. Asic implementation of modified faster carry save adder. *European Journal of Scientific Research*, 42(1):53–58, 2010.

[28] Geeta Rani and Sachin Kumar. Delay analysis of parallel-prefix adders. *International Journal of Science and Research (IJSR)*, 3(6):2339–2342, 2014.

[29] Pandu Ranga Rao and Priyanka Halle. An efficient carry select adder with less delay and reduced area application. *International Journal of Engineering Trends and Technology (IJETT) Volume*, 4.

[30] Gustavo A Ruiz and Mercedes Granda. An area-efficient static cmos carry-select adder based on a compact carry look-ahead unit. *Microelectronics journal*, 35(12):939–944, 2004.

[31] Jagannath Samanta, Mousam Halder, and Bishnu Prasad De. Performance analysis of high speed low power carry look-ahead adder using different logic styles. *International Journal of Soft Computing and Engineering*, 2(2):330–336, 2013.

[32] Michael J Schulte, Kai Chirca, John Glossner, Haoran Wang, Suman Mamidi, Pablo Balzola, and Stamatis Vassi. A low-power carry skip adder with fast saturation. In *Application-Specific Systems, Architectures and Processors*, pages 269–279. IEEE, 2004.

[33] Adilakshmi Siliveru and M Bharathi. Design of kogge stone and brent kung adders using degenerate pass transistor logic. *International Journal of Emerging Science and Engineering(IJESE)*, 1:38–41, 2012.

[34] Sreenivaas Muthyala Sudhakar, Kumar P Chidambaram, and Earl E Swartzlander Jr. Hybrid han-carlson adder. *Circuits and Systems (MWSCAS)*, pages 818–821, 2012.

[35] Tania Sultana, Rajon Bardhan, Tangina Firoz Bithee, Zinia Tabassum, and Nusrat Jahan Lisa. A compact design of n-bit ripple carry adder circuit using qca architecture. *Computer and Information Science (ICIS)*, pages 155–160, 2015.

[36] Yuke Wang, C Pai, and Xiaoyu Song. The design of hybrid carry-lookahead/carry-select adders. *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, 49(1):16–24, 2002.

[37] I-Chyn Wey, Cheng-Chen Ho, Yi-Sheng Lin, and Chien-Chang Peng. An area efficient carry select adder design by sharing the common boolean logic term. In *Proceedings on the International Multiconference of Engineering and computer scientist, IMECS*, 2012.