Spring 2015

CorMem digital reasoning architecture using CMOS Technology

Indira Priyadarshini Dugganapally

Department: Electrical and Computer Engineering

CORMEM DIGITAL REASONING ARCHITECTURE USING CMOS TECHNOLOGY

by

INDIRA PRIYADARSHINI DUGGANAPALLY

A DISSERTATION

Presented to the Graduate Faculty of the

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

In Partial Fulfillment of the Requirements for the Degree

DOCTOR OF PHILOSOPHY

in

COMPUTER ENGINEERING

2015

Approved by:

Steve E. Watkins, Advisor
Daryl Beetner
Minsu Choi
Sahra Sedigh Sarvestani
Yiyu Shi

ABSTRACT

This dissertation describes the application of a multi-level, memory-based approach for building digital circuits. To reflect the alternative approach, the basic science is termed digital reasoning and the specific CorMem technology is based on recent patents. CMOS transistors are used in a non-traditional way for multi-level operations and memory manipulation. The combination of multi-level architectures and matrix algebra principles can create flexible, modular systems using standard fabrication methods and can avoid many of the limitations of other multi-valued logic approaches.

Quaternary, memory-based systems are developed to implement logic-gate-type functions, digital adder circuits, a complete arithmetic and logic unit (ALU), quaternary-to-binary and binary-to-quaternary circuits, and ALU control circuits. All of these implementations use driver and array architectures using CMOS TSMC 130 micron technology. The circuit layouts and functional simulations are given and are compared to binary circuits with comparable functionality. Experimental performance of a hardware AND chip is also demonstrated including excellent signal discrimination. The digital reasoning approach requires more chip area for basic logic gate functionality, but it grows increasingly efficient for more complex systems through hardware reuse. Potential quaternary advantages include fewer interconnections, lesser area requirements, and lower power consumption.

ACKNOWLEDGEMENTS

Foremost, I would like to express my sincere gratitude to my advisor, Dr. Steve Watkins, for the continuous support of my PhD study and research, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my PhD study.

Besides my advisor, I would like to thank the rest of my thesis committee: Dr. Daryl Beetner, Dr. Minsu Choi, Dr. Sahra Sedigh and Dr. Yiyu Shi, for their encouragement and insightful comments.

My sincere thanks to Benjamin Cooper and Gregg Rawson, without whom, this would not even have started in the first place. Ben has been the driving force behind this entire technology and Gregg has been second to none in developing it. They along with the Core Memory Circuits LLC team have been of great help to me being able to pursue my PhD by providing the required funds and resources.

I cannot express enough gratitude to my cousin Abhilash and my loving friends Siddhardha, Pavan, Satish, Vamsi, and Mani for supporting me during the hardest of times. The entire lot has been very helpful in taking care of my son, Shritan when I was busy working.

Apart from everyone, there are three people in my life who have been holding my hand through all my ups and downs, pushed me to pursue the career I was always interested in, never let me give up when I was feeling discouraged and loved me unconditionally: My dad Venkatarami Reddy, my mom Vijaya Lalitha, and my loving husband Uday Kiran – thank you.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

Table Page

1. INTRODUCTION

Multi-level logic approaches to digital circuits have been explored for many years in pursuit of potential benefits in speed, power, and reliability. Proposed approaches usually exploit fewer data interconnects and processing components than binary systems with comparable functionality. However, multi-level implementation has proved difficult. Binary systems have been successful due to simple signal levels and versatile, basic logic gates. Fabrication technology for binary systems has matured and predictable improvements in device density have given enormous economic advantages until recently [1, 2]. Physical limits and associated costs for photolithography techniques and problems with power per computation have largely eliminated the latter advantages. Consequently, interest in alternative approaches for future electronic systems is great [3]. Multi-level approaches, either as complete multi-level systems or as hybrid systems with binary elements, are likely to have a role [4-6].

Research in multi-valued logic, or "fuzzy logic," has been ongoing since 1920. There have been many advances since then and various approaches have been developed. Issues with implementation, signal discrimination, scalability, etc. have limited the application of many of these approaches. This dissertation investigates a multi-level, memory-based architecture that uses mature CMOS fabrication technology and facilitates a modular circuit design methodology. The processing approach is based on matrix algebra and memory manipulation [21, 22]. The basic science behind this approach is termed digital reasoning and the technology originated with Core Memory Circuits LLC. Hence, the name CorMem digital reasoning has originated.

This multi-level architecture is based on a set of modular CMOS structures that are used in a nontraditional way. These modular elements are more complex than the basic logic gates in binary circuitry, but the overall circuitry of complex systems are less complex than binary circuits with comparable functionality. This dissertation deals with quaternary implementations. Four-level development is pursued here since it provides substantial reduction in interconnect lines

without excessive signal discrimination constraints. Quaternary logic circuits are developed with driver and array architectures in CMOS TSMC 130 micron technology. Circuit layouts and functional simulations are given and are compared to binary circuits with comparable functionality. Specifically, quaternary, memory-based systems are developed to implement logic-gate-type functions, digital adder circuits, a complete arithmetic and logic unit (ALU), quaternary-to-binary and binary-to-quaternary circuits, and ALU control circuits. The advantages of CorMem digital reasoning as compared to binary circuits include:

- Fewer interconnections with speed and noise benefits,
- Smaller chip area requirements, and
- Lower power consumption.

Also, this architecture is scalable to higher-level logic.

This dissertation demonstrates a versatile approach to digital logic circuits using quaternary, memory-based technology. Several logic circuits, including a complete ALU, are shown to be feasible through simulation tests. Hardware reuse is extensively applied to optimize the electronic implementations. The third chapter contains an overview of binary versus CorMem digital reasoning approaches in the third chapter. The fourth chapter gives a brief summary of how the logic gates are implemented using this technology and gives details on the array architectures. The following chapter shows the implementation of a full adder circuit and gives details on the driver architectures. The sixth chapter discusses the hardware reuse aspect of the technology in the context of an ALU. The next chapter discusses needed ALU control circuitry. The circuits are laid out and simulations are done using HSPICE. Some of the circuits have been implemented in hardware on a chip and the experimental results confirm device functionality.

2. LITERATURE REVIEW

2.1. MOORE'S LAW

According to Gordon Moore, manufacturers had been doubling the density of components per integrated circuit at regular intervals. He also said that this trend would continue [1]. This observation was made in 1965 and has been well known since then as "Moore's Law". The prediction has been accurate regarding the growth in technology and IC complexity and has become a baseline assumption in the industry's strategic roadmap. It has the power of a self-fulfilling prophecy, if repeated often and sincerely enough [2, 3].

The transistor or the transfer resistor was invented in 1947, and the miniaturization of this device was the basis of Moore's law and that would in return be a hallmark for the semiconductor industry [1]. Fairchild semiconductors developed the first planar IC in the late 1950s. This invention revealed the potential for operation benefits of transistor to electronics.

To go from science to technology is another important factor to be able to implement the theories of solid state electronics. The process to diffuse impurities into the semiconductor surface, the practice of adding conducting and insulating material layers on the top of the substrate, the photographic techniques that were introduced later on to lay intricate mask patterns, and the lithographic techniques have all contributed to the success of Moore's law [43].

The article by Moore in 1965 speculated that, by 1975 there would be around 65,000 components integrated in a silicon chip of an area 6 millimeters square, which after a decade, turned out to be true. Moore based his forecast on a log-linear plot of device complexity over time. Moore redrew his plot of component densities from 1975 forward, stating that the density doubled every 18 months. At present, researchers are still learning to exploit the properties of semiconductors and production processes. However, physical limits, associated cost, etc are changing the engineering and business considerations [24, 42].

2.2. HISTORY OF BOOLEAN LOGIC

The ideological foundations of digital science have been established over a period of time. In 1856, Boole introduced and established algebraic logic. He is the inventor of the prototype of what is now called Boolean logic, which became the basis of the modern digital computer. It is a little known fact that Nikola Tesla is the acknowledged inventor of the electronic AND logic gate circuit, a critical element of every digital computer. In New York City in the mid 1890s Tesla's work was focused on the development of an independent remotely-controlled device—the "telautomaton." In 1932, Claude Shannon entered the University of Michigan, where he took a course that introduced him to the work of George Boole. While studying the complicated ad hoc circuits of the differential analyzer, Claude Shannon saw that George Boole's concepts could be used to great utility. Claude Shannon proved that boolean algebra and binary arithmetic could be used to simplify the arrangement of the electromechanical relays that were used then in telephone call routing switches. He next expanded this concept, and he also proved that it would be possible to use arrangements of relays to solve problems in Boolean algebra. Using this property of electrical switches to do logic is the basic concept that underlies all electronic digital computers. Claude Shannon's work became the foundation of practical digital circuit design. The industry has been developed using Claude Shannon’s concept for more than 60 years [49].

The first electronic computer - ENIAC which stood for Electronic Numerical Integrator and Calculator- was built in 1946 at the University of Pennsylvania, but the invention of the binary system dates almost three centuries back [46]. Gottfried Wilhelm Leibniz (1646-1716), the co-inventor of Calculus, published his invention in 1701 in the paper “Essay d'une nouvelle science des nombres” that was submitted to the Paris Academy to mark his election to the Academy. However the actual discovery occurred more than 20 years earlier.

F.G. Heath described the development of the binary code from Francis Bacon’s “two-letter alphabet” which was conceived at the beginning of the seventeenth century. Subsequently,

Jacquardt's punch-card operated loom (1805) and Boole's logical algebra (1854) led to the introduction of a binary telegraphic alphabet by Baudot (1875). According to the Oxford Encyclopedic Dictionary, an entry "Binary Arithmetic" first appeared in English in 1796. Gottfried Leibniz derived a system of logic for verbal statements that would be completely represented in a mathematical code. He was theorizing that life could be reduced to simple codes of rows of combinations of zeros and ones. These developments led to the use of on/off system of zeros and ones for basic algebraic operations. The on or off codes can rapidly be implemented by computers for doing a great variety of applications [49].

Binary numbers are written with only two symbols - 0 and 1. For example, a = 1101. Since symbols 0 and 1 are also a part of the decimal system and in fact of a positional system with any base, there's an ambiguity as to what 1101 actually stands for. To avoid confusion, the base is often written explicitly, e.g. a= $(1101)_2$ or b = $(1101)_{10}$.

2.3. MULTI-VALUED LOGIC

Multi-level digital approaches to computational electronics, either as complete multi-level systems or as hybrid systems with binary elements, are likely to have a role [4, 6]. Multi-level signals used can potentially reduce interconnect line requirement and processing elements, but their implementation is often hindered by problems with signal-level discrimination and overall system complexity. The principle areas of research included multiple-valued algebra, multiple-valued semi-conductor circuits, and multiple-valued network synthesis [5]. Research in multi-valued logic, or "fuzzy logic," has been ongoing since 1920. Jan Lukasiewicz created a system of many-valued logic [7] that he further developed with Alfred Tarski [8]. Lotfi A. Zadeh added his theory of fuzzy logic in 1973 [9].

Various multiple-valued systems have been investigated. Ternary circuits have been designed and were proved to be smaller, but nevertheless failed to impress the industry due to the lack in ease of use of the ternary system [32, 34]. The multiple-values systems have also been

implemented in various architectures like hybrid SETMOS [10], novel voltage-mode CMOS [11], and also using dynamic differential logic [12].

Several multi-valued FPGA-based systems have been demonstrated [16, 17] in the process of developing the idea. The power comparisons have been done between binary and quaternary, proving the latter to be cutting down the power consumption by at least one-third compared to the binary versions [13, 25, 35, 36]. There have been array-based implementations of the quaternary look up tables and were almost successful in building the quaternary adder but failed in making further developments [18, 19]. There has been another multiple-valued implementation by changing the threshold voltages [33], but this requires a change in the fabrication process.

This work proposes a multi-level architecture that uses mature CMOS fabrication technology and facilitates a modular circuit design methodology. The approach is based on matrix algebra and memory manipulation [21, 22]. Benjamin Cooper, in his patents gives detailed description of how the quaternary values can be decoded without having to change the fabrication process, and also enhances on the memory aspect by using the array-based implementations.

2.4. CMOS TECHNOLOGY AND ADVANTAGES

"CMOS" (complementary metal-oxide semiconductor) refers to both a particular style of digital circuitry design and the family of processes used to implement that circuitry on integrated circuits (chips). CMOS circuitry dissipates less power than logic families with resistive loads. Since this advantage has increased and grown more important, CMOS processes and variants have come to dominate, thus the vast majority of modern integrated circuit manufacturing uses CMOS processes. CMOS circuits use a combination of p-channel and n-channel metal–oxide–semiconductor field-effect transistors (MOSFETs) to implement logic gates. The major advantages offered by silicon-gate CMOS technology have been applied to the fabrication of monolithic integrated circuits for micro power applications [26, 28].

Complementary n-channel and p-channel enhancement-mode devices fabricated with silicon gate in conjunction with a unique substrate preparation technique have exhibited threshold voltages of 0.5 and 0.2 V respectively. Drain-to-source leakage currents, as well as drain to substrate leakage current, are typically less than 1 nA at 3 V. Several types of silicon-gate CMOS integrated circuits (ICs) have been fabricated. An IC that is particularly indicative of the potential utility of silicon-gate CMOS IC's is a toggle flip-flop (binary counter) that is operable from supply voltages down to about 0.9 volt. At 1.5-volt supply with a fan-out of unity, the maximum toggle frequency is 400 kHz, the dynamic power consumption is about 20 nW/kHz and the static power consumption is only about 1.0 nW. Comparisons have been made between silicon gate guard banded CMOS technology, oxide isolated CMOS and CMOS/SOS. The points of comparison include performance, density and flexibility in layout [29]. Also, the reduction in parasitic capacitance and the elimination of parasitic devices in the last two technologies compared with the guard banded technology have significant advantages. However, only marginal performance improvement of the SOS/CMOS over the oxide isolated CMOS technology is predicted due to the effects associated with the SOS material and the floating substrate.

MOS is further classified under PMOS (P-type MOS), NMOS (N-type MOS) and CMOS (Complementary MOS). MOS derives its name from the basic physical structure of these devices; Originally, MOS devices were comprised of a metal, oxide, and semiconductor gate. In current technology, poly-silicon is more widely used as gate. Voltage applied to the gate controls the current between source and drain. Since they consume very low power, MOS allows very high integration. Multiple-valued logics have also been implemented using the CMOS technologies [38, 39].

2.5. COMPUTATIONAL MEMORY

In this dissertation, the expansion of the ideological foundations is done based on bio-mimicry. Bio-mimicry is the examination of nature, its models, systems, processes, and elements to emulate or take inspiration from in order to solve human problems. The same is applied to the digital technology where one of the steps is a comparative study of increasingly complex neural systems. The primary lesson learnt is that in simple systems such as those found in flatworms, they use what could be called 'pure logic system' [47, 48]. The more the neural system becomes advanced, the more nature relies on the concept of "computational memory". An operation is performed by recognizing patterns in data, using values stored in memory, and applying a formal computational procedure.

For example, to multiply 99 and 33, first multiply the 9 and 3 in the units place and ten's place of 99 and the unit's place of 33. Then in the next row the same process is repeated just by shifting the product by one position to the left. So, in the entire process, there is an initial computation of 9 times 3 which is a 27 that we get from the memory and the rest of it follows a computational procedure which is standard. Humans have taken the patterns they recognize in mathematics and embedded the process into the digital systems. The primary subjects that provide pattern recognition are algebra and matrix algebra.

In this work, computational micro-memory is implemented in electronic hardware. Basic structures of such circuitry are memory decoders and memory encoders. The memory decoders are not necessarily binary, and this work uses a quaternary system. The memory encoders provide functions, in part, of an enhanced look-up table. The electronic implementation and performance of a multi-valued, memory-based system [21,22] is the subject of this dissertation. The flexibility and utility of the approach are demonstrated by extensive hardware optimization that is based on comprehensive pattern recognition.

3. CORMEM DIGITAL REASONING OVERVIEW

This chapter describes the digital reasoning approach in developing quaternary digital circuits. The comparison between the binary and the digital reasoning is given. The basic components that are used to build the circuits are introduced and explained briefly.

3.1. BINARY VS CORMEM DIGITAL REASONING

Traditional electronics technology is based upon binary high-low levels representing 1 and 0 and basic Boolean operations of NOT, AND, OR, etc. Semiconductor transistor approaches, e.g. CMOS structures, are used to implement progressively more complex operations of Boolean logic gates, flip-flops, registers, adders, arithmetic logic units (ALUs), etc. Numbers represented in binary require many data interconnect lines. Our CorMem digital reasoning architecture is compared to traditional binary architecture for the quaternary case in Table 3.1. The signal levels are 0, 1, 2, and 3. Semiconductor transistor implementations, also in CMOS, form basic quaternary components. These components are used to assemble systems such as adders and ALUs. Numbers represented in quaternary require fewer data interconnects due to greater information density, e.g. decimal 255 = quaternary 3333 = binary 11111111.

Table 3.2. Comparison of Binary and CorMem Digital Reasoning Technology

	Binary	Digital Reasoning
Signal Levels	0,1	0,1,2,3
Implementation	CMOS	CMOS
Information Density	Low	High
Complexity of Basic Structures	Low	Moderate
Modularity for Systems	No	Yes
Complexity of Systems	High	Low

Figure 3.1 shows a quaternary signaling scheme in CorMem digital reasoning using 130-nm fabrication technology. The logic voltage levels are 0.0 V, 0.4 V, 0.8 V, and 1.2 V. These four levels between 0.0 V and 1.2 V have level tolerances of 0.0 to 0.1 V for a quaternary 0 value, 0.3 to 0.5 V for a 1 value, 0.7 to 0.9 V for a 2 value, and 1.1 to 1.2 V for a 3 value. Other voltages in the CorMem digital reasoning circuitry include the transistor operational voltage.

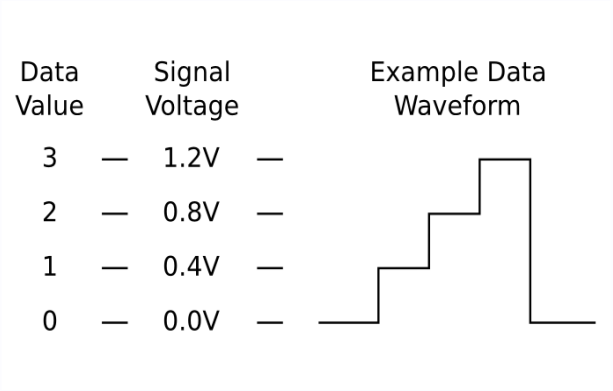

Figure 3.1. Quaternary Signaling Scheme of CorMem Digital Reasoning

The basic structures are sense amplifiers, pass amplifiers, and inverter amplifiers. A sense amplifier compares two quaternary voltage lines and outputs one value for a logic match and another value for a mismatch, typically the output is either the supply voltage or ground voltage. A pass amplifier is a transistor switch that can pass or block a prescribed quaternary logic level. An inverter has the traditional function of high-to-low or low-to-high voltage conversions, although its main function can be considered current amplification. The CMOS implementation of these structures will be described in a later section.

Consider the following example binary and CorMem digital reasoning structures and systems. In CMOS, a basic NOT logic gate is a simple circuit with typically two transistors. A

binary system of an 8-bit ripple carry adder has about 288 transistors and a binary 8-bit carry select adder has about 600 transistors. For the CorMem digital reasoning sense amplifier, the CMOS implementation consists of two to six (typically four) transistors. For the CorMem digital reasoning full adder, an un-optimized system can be formed with 768 transistors. But hardware reuse methodologies and the transformation techniques can be employed to optimize the circuits and the optimized version of the CorMem digital reasoning full adder can be formed with an estimated 256 transistors. Furthermore, the complexity of systems are greatly influenced by interconnect needs. The area of an 8-bit binary carry save adder (CMOS 130-nm technology) is about 6210 μm^2 whereas the area for the 8-bit binary ripple carry adder is 2214 μm^2 [23]. The un-optimized version, which is being discussed in this paper, is much bigger in size when compared to the optimized, the details of which are discussed in the later chapters. Note that the CorMem digital reasoning adder is more flexible, in that it can be used as an 8-bit adder, two 4-bit adders, four 2-bit adders, etc.

3.2. CORMEM DIGITAL REASONING APPROACH AND COMPONENTS

The CorMem digital reasoning approach is based upon performing operations using matrix algebra. Matrix array values are determined using connections to reference logic levels. A CorMem digital reasoning scheme uses drivers and arrays as the primary build-out components. These systems are formed in a modular way using the basic CorMem digital reasoning structures. For instance, a driver can be formed using sense amplifiers. The driver acts as a decoder or register for a quaternary voltage input. The driver produces decode or control voltages for the array. The array is formed using pass amplifiers. Two decode voltages select a single cell in the array and pass the associated quaternary memory value as an output. The cells are arranged in rows and columns and the memory values are set by reference connections, i.e. values structures. The double inverters are used as amplifiers of the signals.

Figure 3.2 shows an example of a digital reasoning circuit with two drivers and an array. The driver takes the quaternary inputs and produces desired control voltages which are binary signals. The array consists of four rows of pass amplifiers with each row having four cells, i.e. four pairs of pass amplifiers. Each pair will store a single quaternary memory value for a total of sixteen values in the array, i.e. a four-by-four array. The control voltages from one driver will select a row and the control voltages from the other driver will select a column. The single cell of the sixteen cells is enabled such that its reference connection or value structure is the quaternary output.

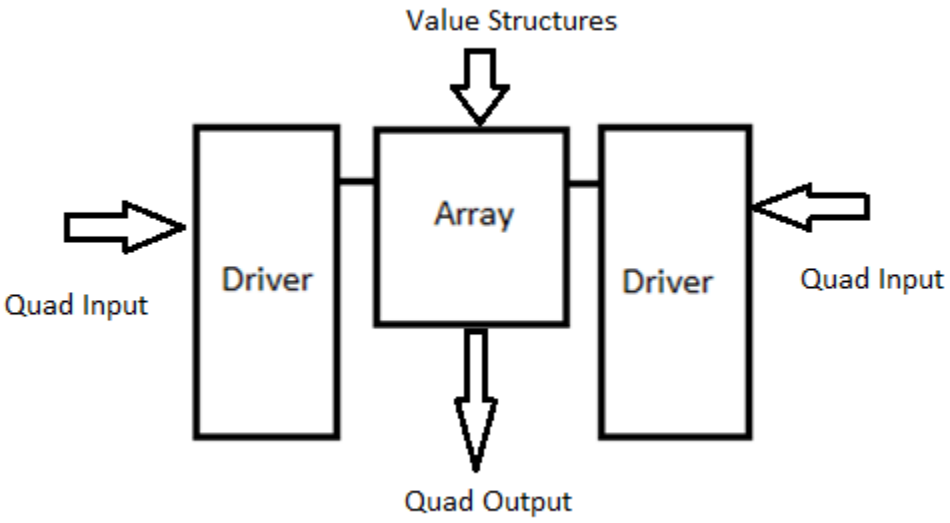

Figure 3.2 CorMem Digital Reasoning Basic Components

Digital reasoning circuits can be used to implement a fully multi-level architecture or can be used in conjunction with binary circuits. For the latter case, binary-to-quaternary and

quaternary-to-binary circuits are needed. A binary-to-quaternary circuit is formed using pass amplifiers similar to those in the array. The binary input selects the appropriate pair of pass amplifiers to output the needed quaternary voltage. A quaternary-to-binary circuit is formed using sense amplifiers similar to those in the driver. The quaternary input results in a binary output. The detailed architecture of these circuits is shown in the next chapter.

4. LOGIC GATES

This chapter gives a detailed description of quaternary-based logic gates as an example of the approach. The basic architecture that is used in the implementation is explained and the method of selecting the final output from the memory-based array cells is formulated. The schematic implementations of the array and the buffer are explained along with the simulation of one of the logic gates as an example.

4.1. INTRODUCTION

The truth tables of quaternary logic for the quaternary AND, NAND, OR and NOR operations are shown in Table 4.1. The inputs can be 0, 1, 2, or 3. Considering two input gates, there can be 16 possible input combinations for each gate. Consider the example of inputs being 2 and 3. The representation in binary is 10 and 11. When the AND operation is performed on the binary numbers, it is a bit-by-bit operation, hence giving an output of 10, which is a 2 in quaternary logic. The truth table is thus obtained by the above method.

Table 4.1 Truth Tables for Quaternary Logic Operations

Input 1	Input 2	AND	NAND	OR	NOR
0	0	0	3	0	3
0	1	0	3	1	2
0	2	0	3	2	1
0	3	0	3	3	0
1	0	0	3	1	2
1	1	1	2	1	2
1	2	0	3	3	0
1	3	1	2	3	0
2	0	0	3	2	1
2	1	0	3	3	0

Table 4.1 Truth Tables for Quaternary Logic Operations (cont.)

2	2	2	1	2	1
2	3	2	1	3	0
3	0	0	3	3	0
3	1	1	2	3	0
3	2	2	1	3	0
3	3	3	0	3	0

The digital reasoning based implementations of these operations are shown in the block diagram of Figure 4.1. Each array is populated with the truth table values for a given logic operation. The array values are set by internal voltage connections (not shown). The cells in this array are populated with the bottom-most row being row zero and the left-most column being column zero. The quaternary inputs to the drivers, shown to either side of the arrays, produce control voltages that select an entire row or entire column. The quaternary output of the array is the value in the selected cell.

Note that the drivers have the same function for each array; hence, a common row driver and a common column driver can control multiple logic arrays to reduce transistor needs in complex circuits. For instance, the circuitry for two drivers and four arrays could produce the outputs for all four AND, NAND, OR, and NOR operations.

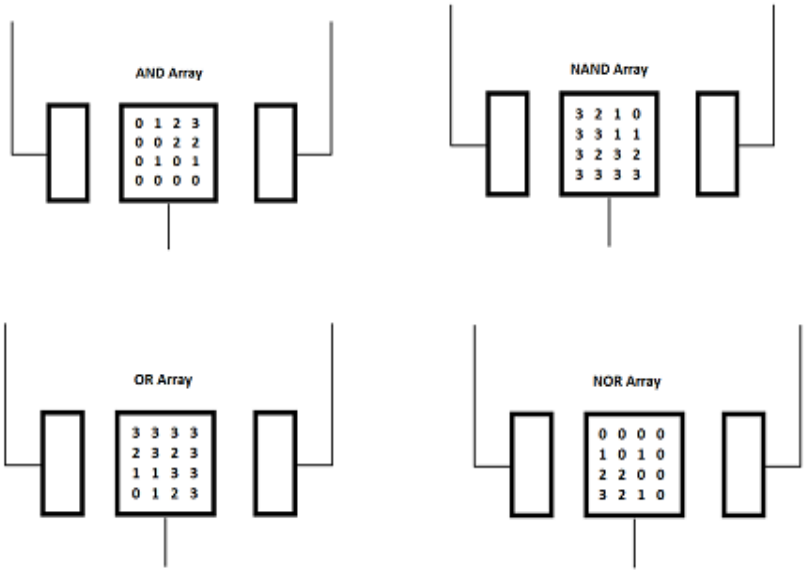

Figure 4.1 Array Blocks for Quaternary Logic Operations

Figure 4.2 shows an example of cell selection in the AND array component. The inputs to the drivers activate the row corresponding to logic 2 and the column corresponding to logic 3. The array output is a quaternary logic 2.

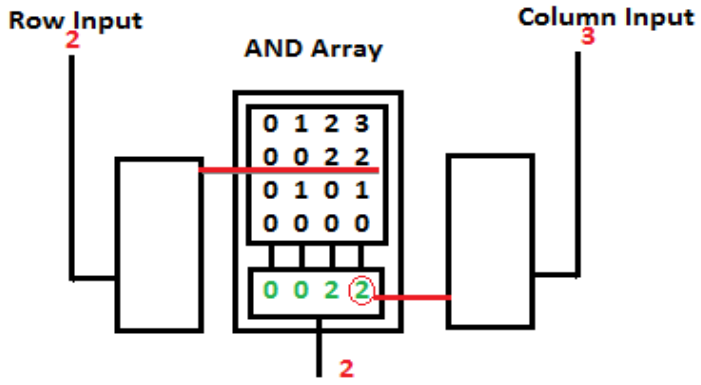

Figure 4.2 Example Selection in the AND Array Component

4.2. COMPONENT STRUCTURES IN THE DIGITAL REASONING APPROACH

The basic structures of the digital reasoning approach are the inverters, sense amplifiers, and pass amplifiers. These structures will be more complex, e.g. higher transistor count, than the basic components in a traditional binary system of logic gates. However, due to the modularity of the digital reasoning approach, system circuitry for higher-order applications such as adders [17] will be less complex for the digital reasoning approach as compared to a traditional binary system. The CMOS implementation of these structures will be shown in the next section.

A chain of two inverters acts as a buffer. Rather than inverting a logic signal as in a binary circuit, the function of the inverters is amplification between drivers and arrays. The sense amplifiers are used to implement the drivers that act as decoders for the multi-level input signals. These structures sense a differential voltage and amplify the difference to the appropriate value for the needed control line for the array row or column. The differential voltage is between the input quaternary signal and reference voltage lines. The pass amplifiers are used to implement the arrays. The array cells use pass amplifiers as switches to pass voltage levels between nodes in the array circuit. The cell values are set by reference connections, i.e. selected voltage lines or value structures.

The circuit schematics for a buffer (double-inverter) and an array are shown in Figure 4.3 and Figure 4.4 respectively. The driver (explained in future chapters) produces six output lines which act as control voltages. The six output lines from the driver are sent to a six-to-eight decoder (explained in future chapters) which produce the four pairs of control lines. These inverted pairs from the decoder are labeled L0, L0 _N, L1, L1_ N, L2, L2_ N, L3, and L3_N. Table II shows the relationship among the quaternary input cases and the array control pairs. Each inverted pair has high-low voltages that select desired PMOS and NMOS transistors in the array.

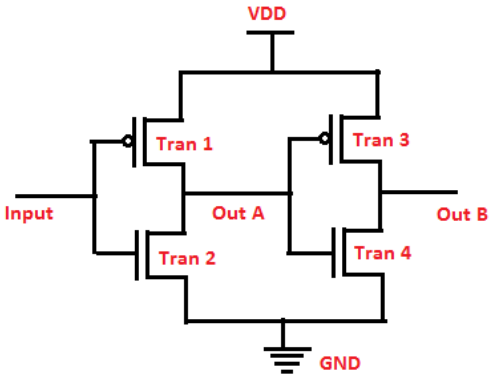

Figure 4.3 Double Inverter Implementation

For the AND array in Figure 4.4, the row control lines are L0 and L0_N for row zero, L1 and L1_N for row one, L2 and L2_N for row two, and L3 and L3_N for row three. The value structures are shown at the top and the outputs to a column selector circuit are shown at the bottom. This column selector circuit has a similar architecture of pass amplifiers with the control lines from the column driver as inputs and the selected row values as reference voltages. The layout of the array and the selector are shown in appendix A.

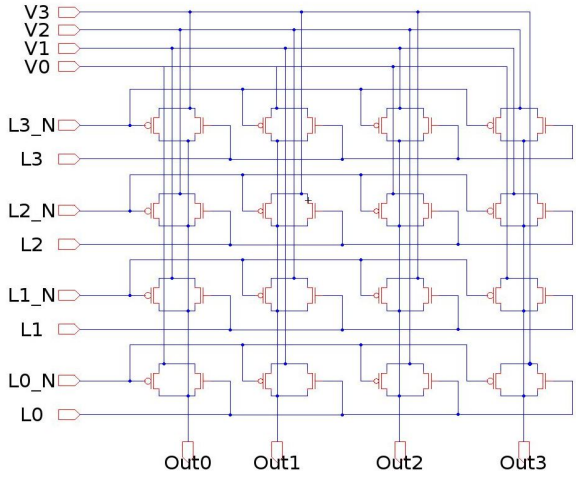

Figure 4.4 AND Array Implementation

Table 4.2 Truth Tables for Quaternary Logic Operations

	Quad Input 0	***Quad Input 1***	***Quad Input 2***	***Quad Input 3***
L3	0	0	0	1
L3 N	1	1	1	0
L2	0	0	1	0
L2 N	1	1	0	1
L1	0	1	0	0
L1 N	1	0	1	1
L0	1	0	0	0
L0 N	0	1	1	1

4.3. RESULTS AND SIMULATIONS

The quaternary AND, NAND, OR and NOR circuits were simulated in HSPICE based on the layout for CMOS TSMC technology. Each truth table case was reproduced. Typical plots of the internal outputs from the array are given in Figure 4.5 for all the possible 16 cases in the AND circuit. The inputs are datainrow and dataincol and the outputs of the array show addout0, addout1, addout2, and addout3 from top to bottom, respectively. The datainrow signal decrements from value 3 to value 2, to value 1, and then to value 0 with a time period of 30ns for each value. The dataincol goes from value 2 to value 1, to value 0, and then to value 3 with a time period of 120ns for each value. The cursor is placed at datainrow value 0 (voltage 0V) and dataincol value 2 (voltage 0.8V). The array should output the values 0, 1, 2, and 3 for addout0, addout1, addout2, and addout3 respectively for which the voltages are 0 V, 0.4 V, 0.8 V, and 1.2 V. The addoutput is the final output of the selector, which at the cursor adds the values 0 and 2 and outputs a 2 (voltage 0.8 V). Note that the simulations are from the layouts and considering the parasitic. Also, the valid data window is when the powerinlow signal is a 0 V. The powerinlow signal being at 1.2 V is the time required for the driver to sense the input voltages.

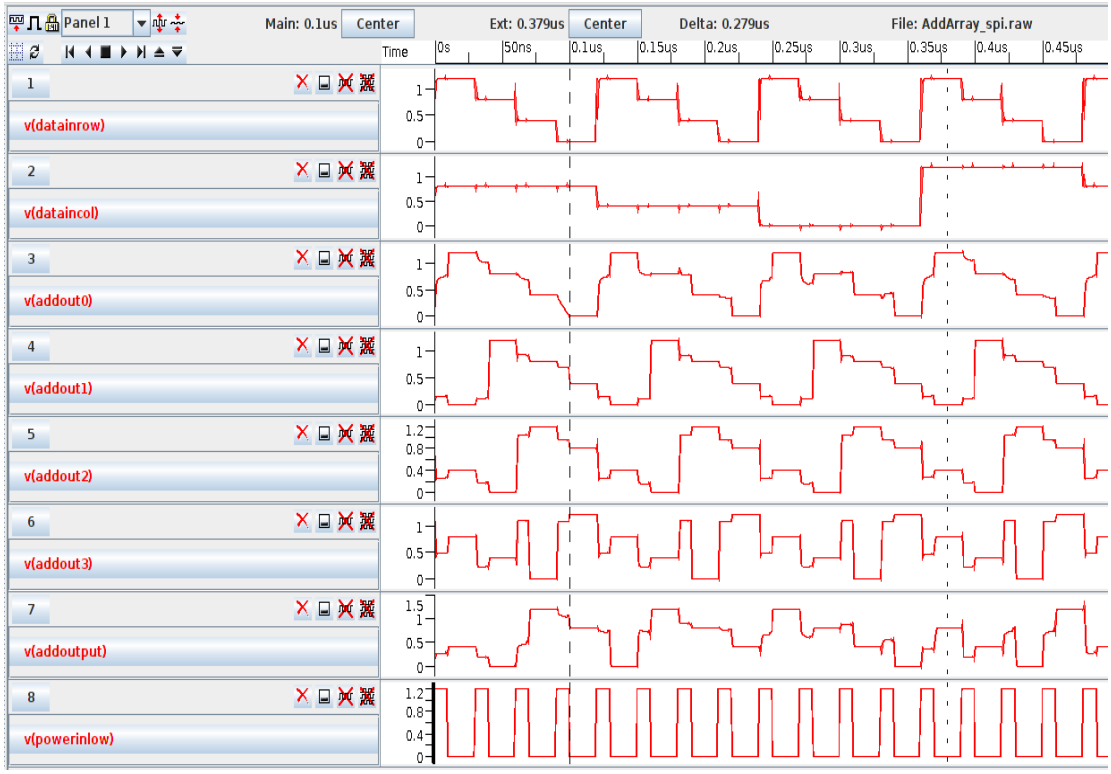

Figure 4.5 AND Array Simulation

5. ADDER IMPLEMENTATION

This chapter explains the implementation of a full adder using the digital reasoning approach. It shows the truth table and architecture of an adder in the quaternary logic with its sum and carry output. The implementation for the driver circuit is shown, which converts a quaternary input to three pairs of control signals. Also, binary-to-quaternary and quaternary-to-binary schemes are explained along with a six-to-eight decoder circuit. Simulations are shown for the adder circuit.

5.1. INTRODUCTION

Consider the truth table and the half adder sum scheme shown in Figure 5.1. The quaternary inputs, designated C and R for column and row, respectively, produce a quaternary output with signal levels 0, 1, 2, and 3. A row driver and a column driver take these inputs and produce two control voltages. The control voltages select either an entire row or column in the digit array. The cells in this array are populated according to the truth table with the bottom most row being row zero and the left most column being column zero. The array output is the selected cell. The highlighted case is for a row input of logic 2 and column input of logic 3 such that the array output is logic 1. The adder is not completely done, since the carry part of the result is neglected. Figure 5.2 shows an additional truth table and the associated carry out scheme. This array, which is populated with new values, produces an output that is the high or low of the carry. For the example of the sum of logic 2 and 3, the digit output is 1 with a carry out of 1.

Out	C	R
0	0	0
1	0	1
2	0	2
3	0	3
1	1	0
2	1	1
3	1	2
0	1	3
2	2	0
3	2	1
0	2	2
1	2	3
3	3	0
0	3	1
1	3	2
2	3	3

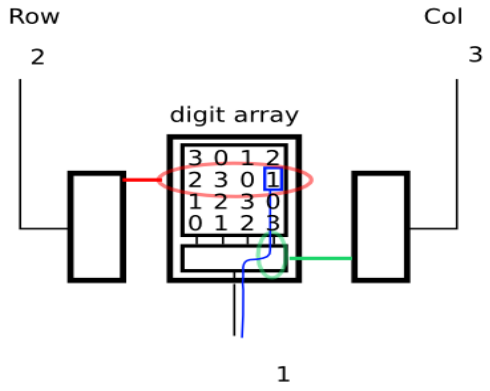

Figure 5.1. Truth Table and Quaternary Scheme of Half Adder Sum

Carry Out	C	R
0	0	0
0	0	1
0	0	2
0	0	3
0	1	0
0	1	1
0	1	2
1	1	3
0	2	0
0	2	1
1	2	2
1	2	3
0	3	0
1	3	1
1	3	2
1	3	3

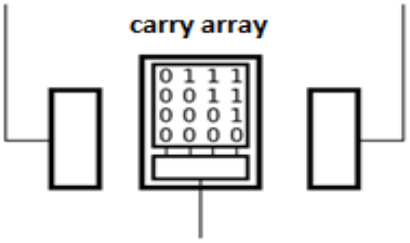

Figure 5.2. Truth Table and Quaternary Scheme of Half Adder Carry Out

The complete half adder is shown in Figure 5.3(a). However, the two row drivers and the two column drivers have the same function. In Figure 5.3(b), the redundant drivers are eliminated to reduce the device area and transistor count.

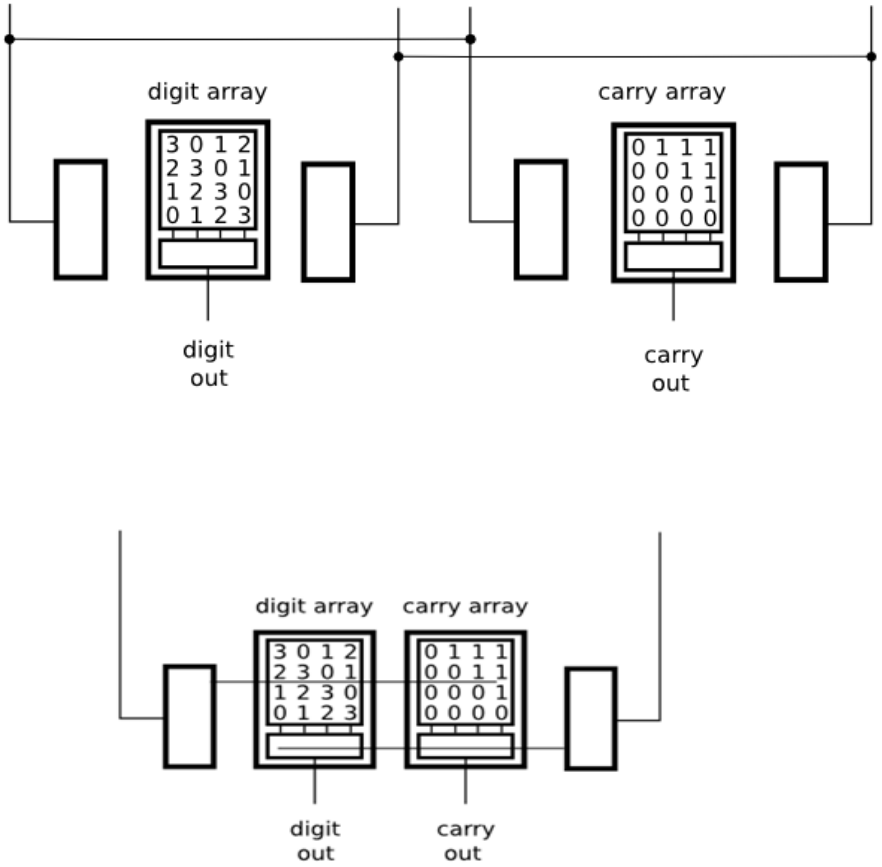

Figure 5.3(a) Half Adder 5.3(b) Half Adder with Hardware Re-use

A full adder must include a third input which is the carry input. If the carry input is 0, the truth table and array in Figure 5.1 apply. If the carry input is 1, the applicable truth table for the row, column, and carry inputs is shown in Figure 5.4. Hence, another array is needed that is populated by this truth table. The driver for the carry input controls another component, effectively a 1x2 array, that selects which output to pass as shown in the scheme of Figure 5.4. Similarly, the carry input will also modify the carry output and a dual carry array scheme is needed. The final full adder scheme with carry in and carry out is shown in Figure 5.5.

Digit w/Carry			Carry
1	0	0	1
2	0	1	1
3	0	2	1
0	0	3	1
2	1	0	1
3	1	1	1
0	1	2	1
1	1	3	1
3	2	0	1
0	2	1	1
1	2	2	1
2	2	3	1
0	3	0	1
1	3	1	1
2	3	2	1
3	3	3	1

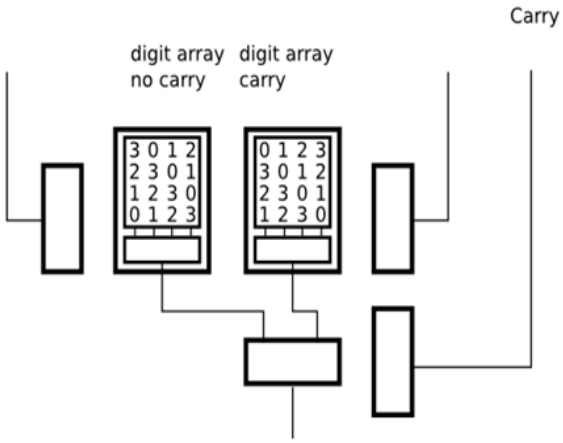

Figure 5.4. Truth Table for Carry In and Quaternary Scheme of Half and Full Adder

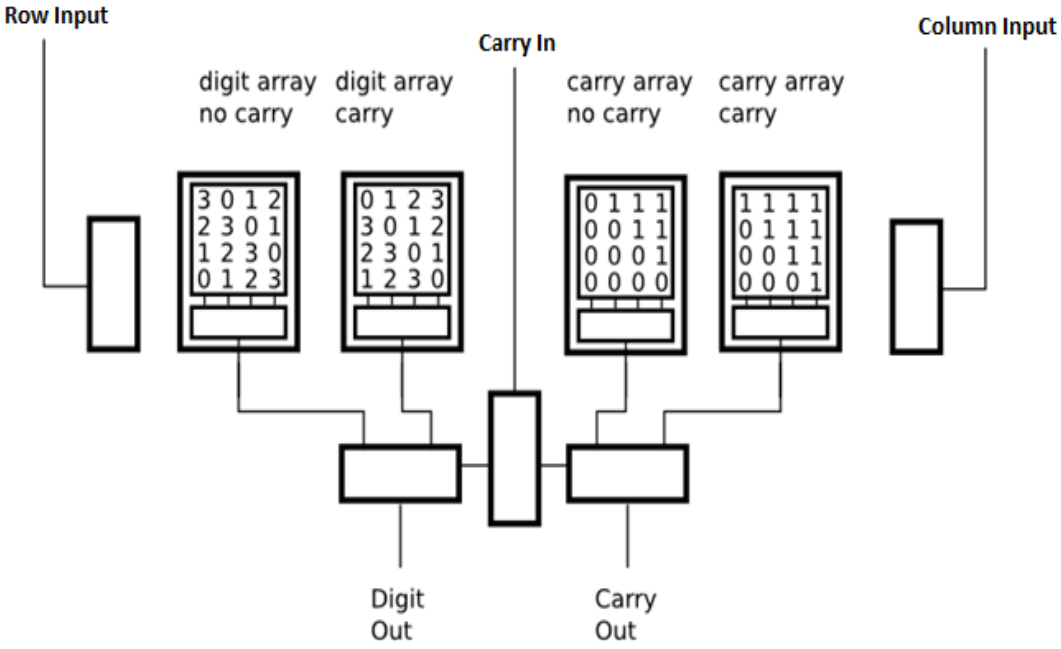

Figure 5.5. Quaternary Scheme of Half and Full Adder with Sum and Carry Out

5.2. BINARY AND QUATERNARY CONVERSION SCHEME

In order to create hybrid circuits for binary and quaternary, conversion circuits are considered. Figure 5.6 shows the scheme for a binary-to-quaternary converter circuit. This circuit uses pass gates, similar to those in an array. The only difference here is that it uses double pass gates, which is a pair of pass gates. These allow the input to go the output only when both the pairs are turned on. The inputs to the pass gates are the binary values which allows to select one of the four pass gates and outputs the voltage from the value structures which is either a 0V, 1V, 2V, or a 3V, hence giving the quaternary output.

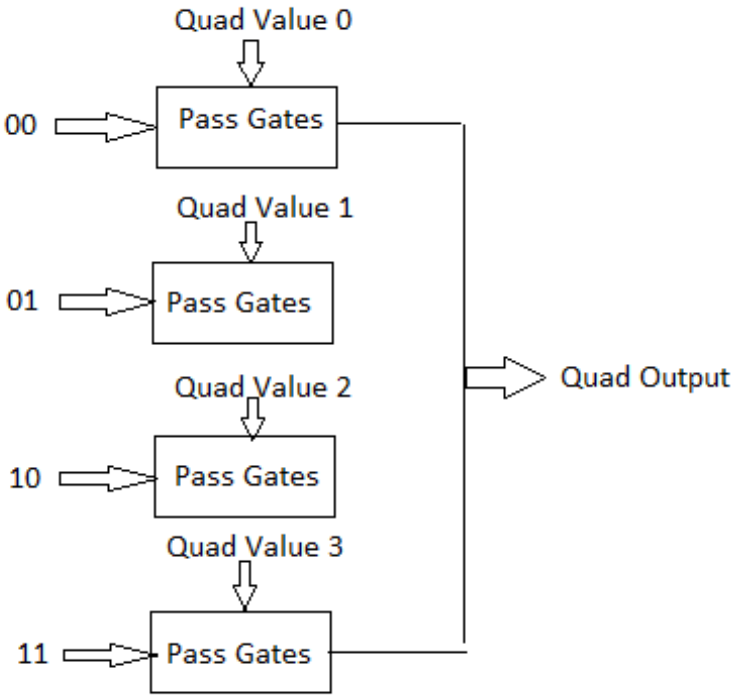

Figure 5.6. Binary-to-Quaternary Converter Scheme

The quaternary-to-binary converter circuit has a quaternary input and outputs a two-bit binary number. The conversion needs a driver that decodes the quaternary input to three pairs of output lines. (The driver circuitry will be described in the next section.) The driver produces these output lines which act as the inputs for the logic to get the binary output. The output pairs are referred to as data and reference lines and are abbreviated as Dat and Ref lines, respectively. Table 5.1 below shows the corresponding binary output along with the six output lines of a driver for the given quaternary input. From the table, it can be seen that the most significant bit (MSB) is the same as the Dat1 output line and the least significant bit (LSB) can be obtained by obtaining a sum of products equation by using two AND gates and one OR gate. Figure 5.7 depicts the quaternary-to-binary scheme.

Table 5.1. Binary values corresponding to Quaternary Inputs

Quaternary Input	Dat2	Ref2	Dat1	Ref1	Dat0	Ref0	Binary Out MSB	Binary Out LSB
0	0	1	0	1	0	1	0	0
1	0	1	0	1	1	0	0	1
2	0	1	1	0	1	0	1	0
3	1	0	1	0	1	0	1	1

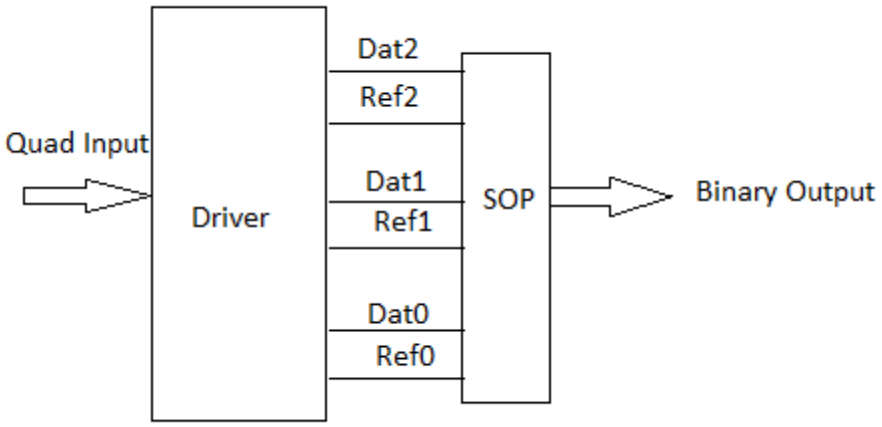

Figure 5.7. Quaternary-to-Binary Converter Scheme

5.3. QUATERNARY ADDER CIRCUIT COMPONENTS

The sense amplifiers are the basic building blocks of the driver. The schematic in Figure 5.8 is the driver which has the sense amplifiers along with isolation circuit and inverters. There are three steps in firing off the sense amplifier: in the first step, the output is isolated. Once the isolation is done, the power and the ground to the core circuit is cut-off. Then, the data and reference values are brought in. Once this is done, the reverse process occurs. The data and reference values are shut off, then core circuit is tied to power and ground which starts the amplification process and then the isolation is shut off. Now, the sense amplifier outputs its value. Three such sense amplifier circuits are stacked on each other to complete the driver. So, the entire driver has three sets of data and reference values. The Table 5.2 below gives the functioning outputs of the driver depending on the input data.

Table 5.2. Driver Outputs for Corresponding Input Data

	Input Data 0	Input Data 1	Input Data 2	Input Data 3
Data 2	0	0	0	1
Reference 2	1	1	1	0
Data 1	0	0	1	1
Reference 1	1	1	0	0
Data 0	0	1	1	1
Reference 0	1	0	0	0

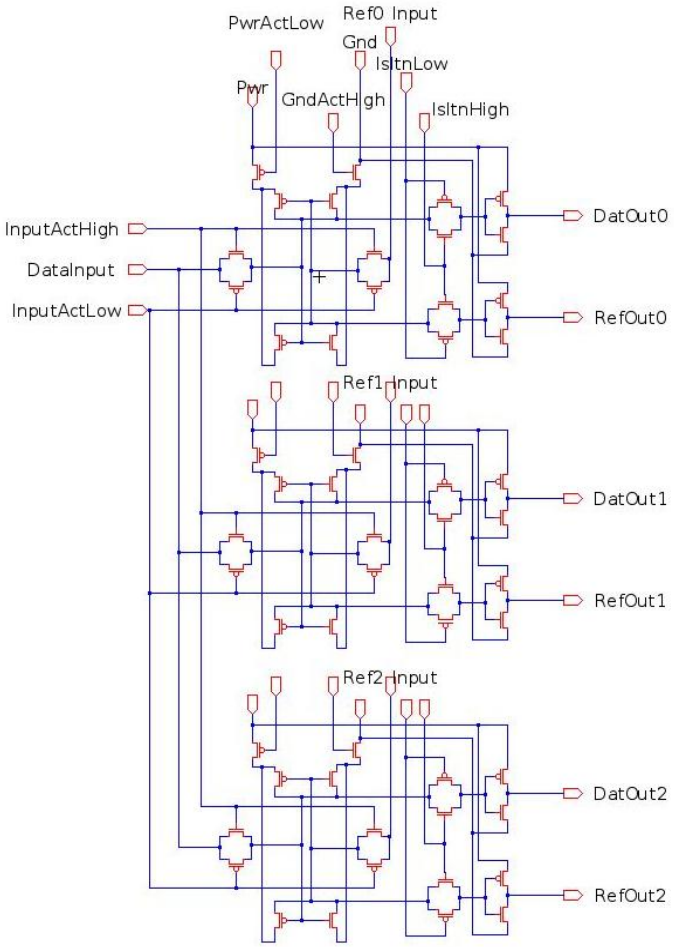

Figure 5.8. Circuit Schematic for the Driver

The six output lines from the driver are sent to a decoder, shown in Figure 5.9 which produces four pairs of control lines. These inverted pairs from the decoder are labeled L0, L0_N, L1, L1_N, L2, L2_N, L3, and L3_N. Table 5.3 shows the relationship among the quaternary input cases and the array control pairs. Each inverted pair has high-low voltages that select desired PMOS and NMOS transistors in the array. Appendix B shows the layouts of the driver and the six-to-eight decoder.

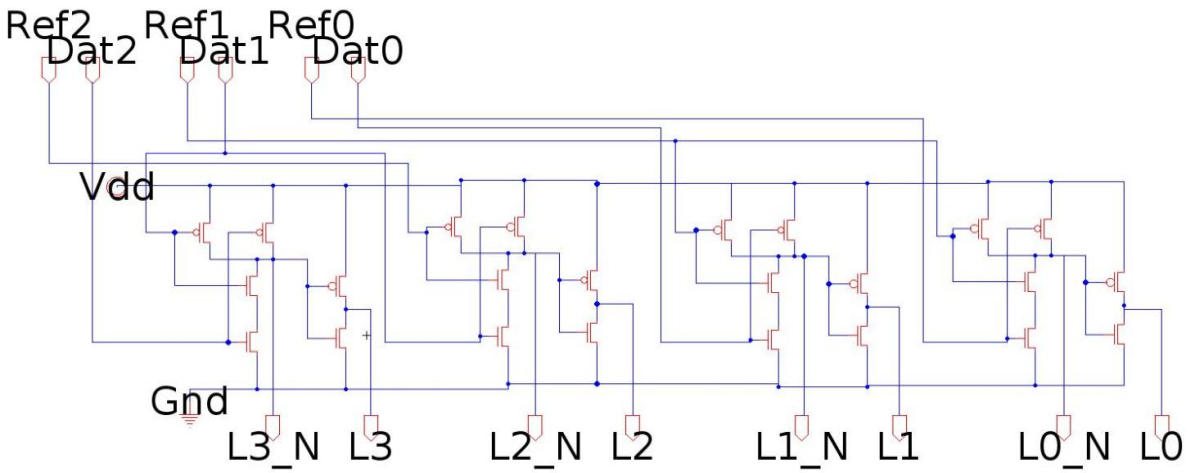

Figure 5.9. Decoder that connects the Driver to the Array

Table 5.3. Array Row Control Line Values

	Input Data 0	Input Data 1	Input Data 2	Input Data 3
L3_N	High	High	High	Low
L3	Low	Low	Low	High
L2_N	High	High	Low	High
L2	Low	Low	High	Low
L1_N	High	Low	High	High
L1	Low	High	Low	Low
L0_N	Low	High	High	High
L0	High	Low	Low	Low

To implement any function, two inputs are sent through two drivers, row driver and column driver. The six output lines of the row driver are sent into the decoder which outputs four pairs of lines. Each pair is used to select one out of the four rows of pass amplifiers in an array. The example shown in the Figure 5.10 below is of a digit array with no carry input. The inputs of the pass amplifiers are tied to the external value structures with inputs 0V, 0.4V, 0.8V, and 1.2V. So, the row 3 which is the top most row of the array is tied to values 3, 0, 1, and 2 respectively starting from left to right, row 2 which is the second row from top is connected to values 2, 3, 0, and 1 respectively and so on. The select lines of the pass amplifiers activate one of the four rows and outputs the value structure inputs, thus giving four outputs. Another selector component, shown in Figure 5.11, is also made of pass gates, and is connected to the array. The select lines of the selector come from the second driver, which is the column driver, and the inputs are connected to the four outputs from the array. The array has four rows and four columns, which is

made of 16 pass amplifiers. The selector has two rows and two columns, which is made of four pass amplifiers. The select lines of the second driver activate one of the pass amplifier pairs, hence outputting one final value.

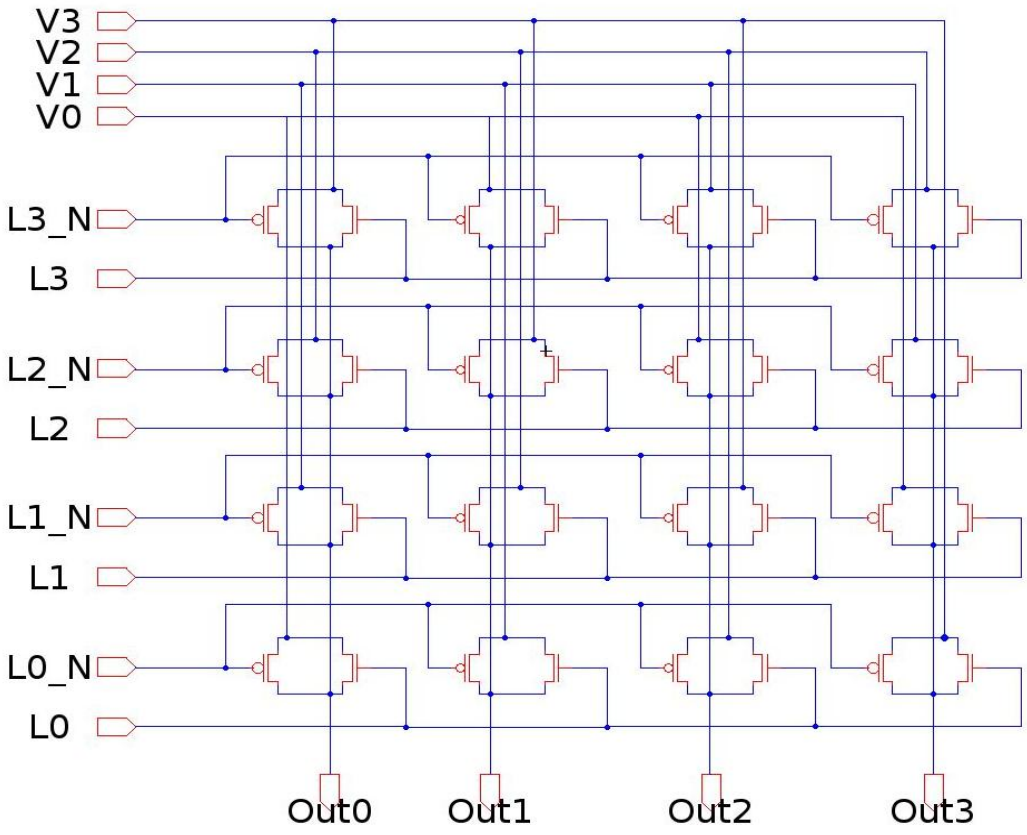

Figure 5.10. Array Component with Digit Out Programming

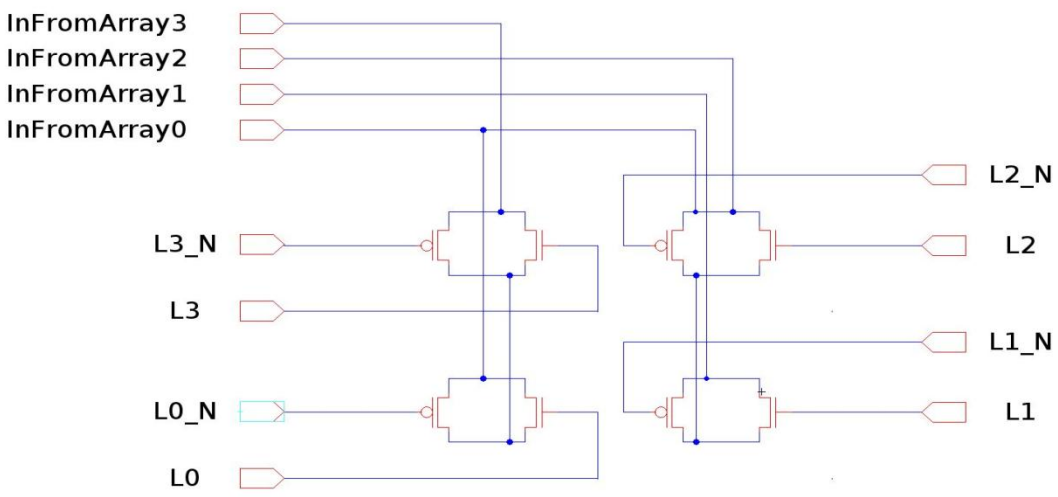

Figure 5.11. Selector Schematic

5.4. BINARY AND QUATERNARY CONVERTER CIRCUITS

The schematic of the binary-to-quaternary circuit is shown in the Figure 5.12 below. There are four sets of double pass gates used and only one set is turned on at a given point of time with the other three being in the off state. The double pass gates are on only if both the PMOS are on or if both the NMOS are turned on at the same time. The inputs to the PMOS and NMOS are always complementary. The quaternary values are passed to the output depending on which set of double pass gates are on. For example, if we want value 2 to pass to the output, the binary input should be 10. So, the MSB is a 1 and the LSB is a 0, and they are arranged such that one of the PMOS in the third set of pass gates gets the LSB and the other PMOS gets the inverted MSB. This allows both the PMOS to be on and hence passed the value 2 to the output.

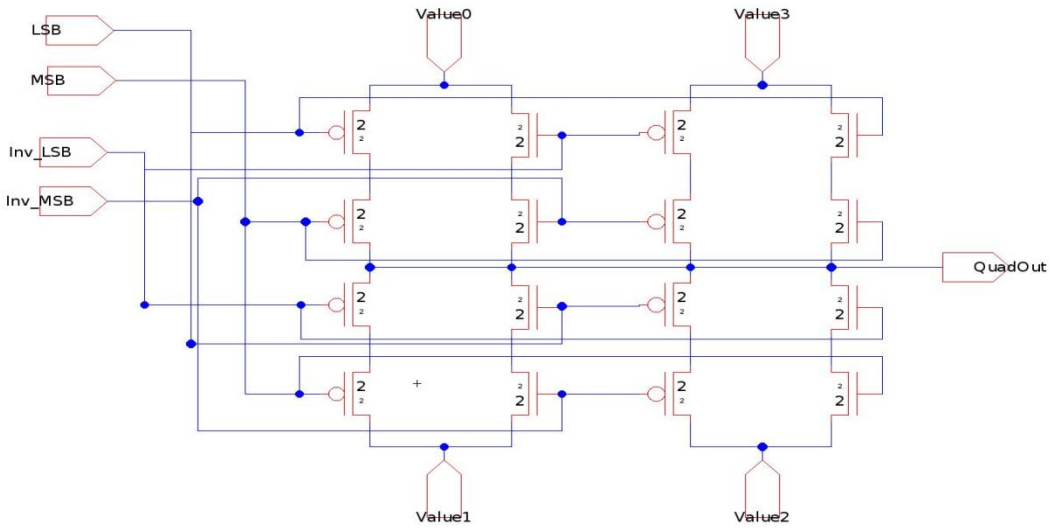

Figure 5.12. Binary-To-Quaternary Schematic

The schematic of the quaternary-to-binary circuit is partially shown in the Figure 5.13 below. The six input lines that are going into the logic gates to obtain the binary output are the outputs of the driver circuit. Appendix C shows the layouts of both the converter circuits.

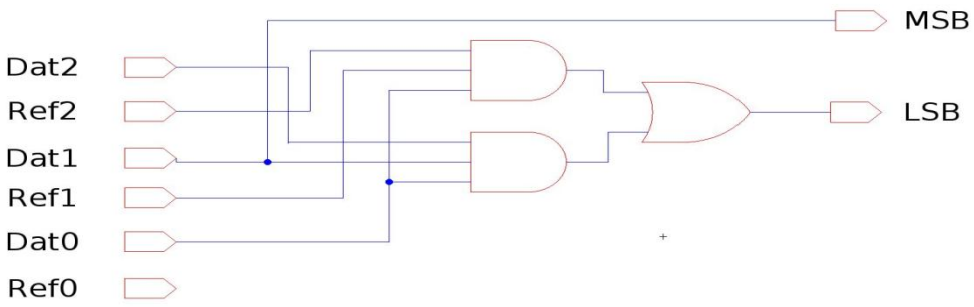

Figure 5.13. Quaternary-To- Binary Schematic

5.5. RESULTS AND SIMULATIONS

The driver circuit along with the binary-to-quaternary and the quaternary-to-binary circuits have been simulated in HSPICE. The simulation below in Figure 5.14 shows the driver's simulation results. The data outputs for the input values of the driver going from a digit 3 to 2 to 1 to a 0 are shown. Note that the digit 3 is 1.2V, 2 is 0.8V, 1 is 0.4V, and 0 is 0V. The Figure 5.15 shows the simulation of the binary-to-quaternary circuit. All the simulations are obtained from the layouts considering the parasitic extractions.

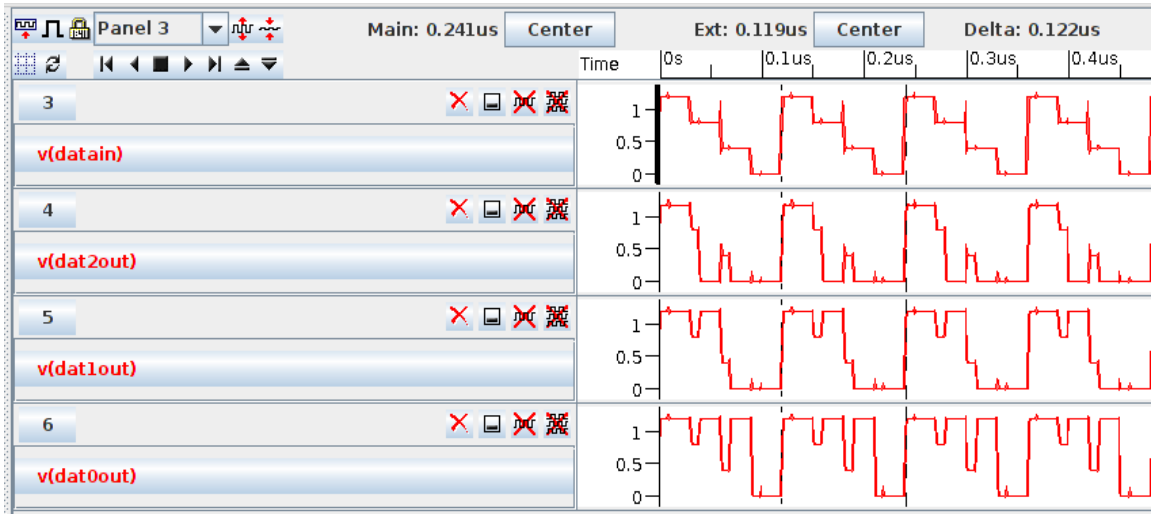

Figure 5.14. Driver Simulation Results

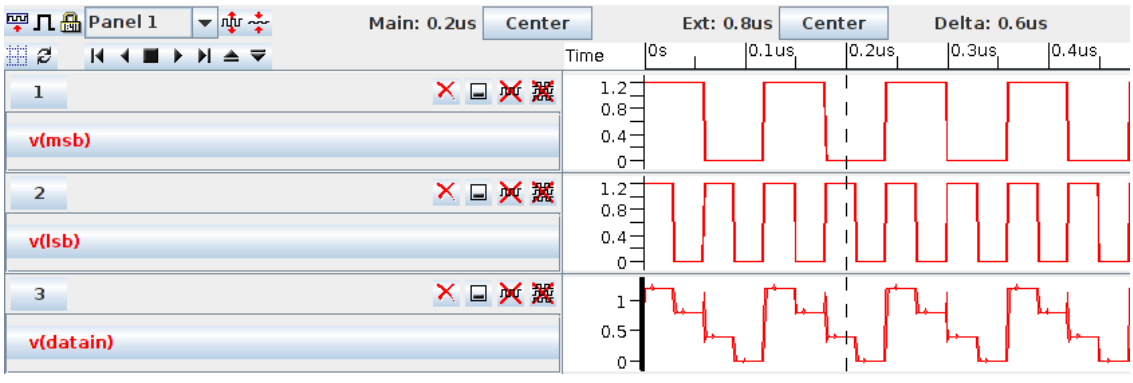

Figure 5.15. Simulation of Binary-to-Quaternary Circuit

The quaternary ADD circuit was simulated in HSPICE based on the layout for CMOS TSMC technology. Each truth table case was reproduced. Typical plots of the internal outputs from the array are given in Figure 5.16 for all the possible 16 cases in the ADD circuit. The inputs are datainrow and dataincol and the outputs of the array show addout0, addout1, addout2, and addout3 from top to bottom, respectively. The datainrow decrements from value 3 to value 2, to value 1, and then to value 0 with a time period of 30 ns for each value. The dataincol goes from value 2 to value 1, to value 0, and then to value 3 with a time period of 120ns for each value. The cursor is placed at datainrow value 0 (voltage 0 V) and dataincol value 2 (voltage 0.8 V). The array should output the values 0, 1, 2, and 3 for addout0, addout1, addout2, and addout3 respectively for which the voltages are 0 V, 0.4 V, 0.8 V, and 1.2 V. The addoutput is the final output of the selector, which at the cursor adds the values 0 and 2 and outputs a 2 (voltage 0.8 V). Note that the simulations are from the layouts and considering the parasitic. Also, the valid data window is when the powerinlow signal is a 0 V. The powerinlow being at 1.2 V is the time required for the driver to sense the input voltages.

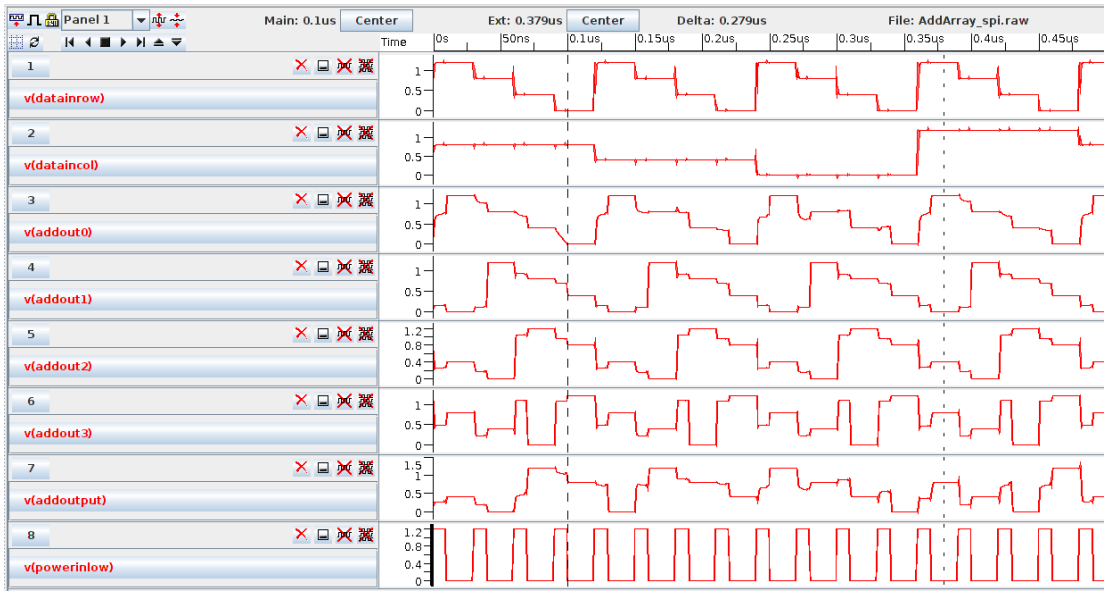

Figure 5.16. Simulation of Digit Output of the ADD Array

6. ALU IMPLEMENTATION

This chapter explains the implementation of arithmetic and logic unit (ALU) using the digital reasoning approach. The optimization of the ALU using hardware reuse is the major concern of this chapter. Several pattern recognition techniques are shown by which the hardware reuse can be attained. The logic functions along with adder and subtractor are implemented in the ALU. The schematic of the hardware required to attain the reuse is shown.

6.1. INTRODUCTION

Arithmetic and Logic Unit has an input and an output bus and it performs arithmetic and bitwise logical operations on digital numbers. The ALU consists of control signals which are usually called the opcode. Based on the opcode the ALU configures to perform either add, subtract, or any of the logical operations. The ALU can be designed based on the application or the user requirements. The Figure 6.1 below shows a general representation of an ALU.

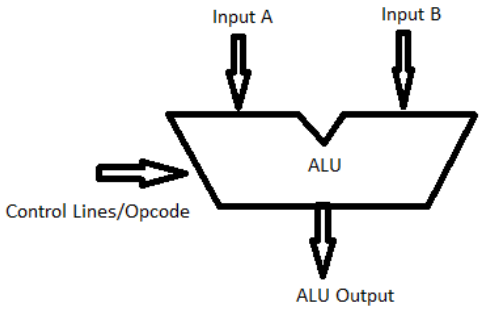

Figure 6.1. Representation of Arithmetic and Logic Unit

However, for the memory based architecture, the quaternary logic has been considered instead of binary but is not limited to four levels. The same concepts can be applied for higher enumerations as well. The logical operations are performed on the binary equivalents of the

inputs and are converted to quaternary from the binary. The truth table of quaternary logic for the quaternary operations in the ALU is shown in Table 6.1. The inputs can be 0, 1, 2, or 3. Considering two input gates, there can be 16 possible input combinations for each gate. Consider the example of inputs being 2 and 3. The representation in binary is 10 and 11. When the AND operation is performed on the binary numbers, it is a bit-by- bit operation, hence giving an output of 10, which is a 2 in quaternary logic. The truth table is thus obtained by the above method.

Table 6.1. Truth Table of Quaternary ALU Operations

Input 1 (Col)	***Input 2 (Row)***	***Sum w/o CarryIn***	***Sum with CarryIn***	***Carry Out w/o CarryIn***	***Carry Out w CarryIn***	***Bin Equivalent of Sub***	***Bin Equivalent of AND***	***Bin Equivalent of NAND***	***Bin Equivalent of OR***	***Bin Equivalent of NOR***	***Bin Equivalent of XOR***	***Bin Equivalent of INV***
0	0	0	1	0	0	0	0	3	0	3	0	3
0	1	1	2	0	0	3	0	3	1	2	1	3
0	2	2	3	0	0	2	0	3	2	1	2	3
0	3	3	0	0	1	1	0	3	3	0	3	3
1	0	1	2	0	0	1	0	3	1	2	1	2
1	1	2	3	0	0	0	1	2	1	2	0	2
1	2	3	0	0	1	3	0	3	3	0	3	2
1	3	0	1	1	1	2	1	2	3	0	2	2
2	0	2	3	0	0	2	0	3	2	1	2	1
2	1	3	0	0	1	1	0	3	3	0	3	1
2	2	0	1	1	1	0	2	1	2	1	0	1
2	3	1	2	1	1	3	2	1	3	0	1	1
3	0	3	0	0	1	3	0	3	3	0	3	0
3	1	0	1	1	1	2	1	2	3	0	2	0
3	2	1	2	1	1	1	2	1	3	0	1	0
3	3	2	3	1	1	0	3	0	3	0	0	0

The memory-based implementation uses the arrays where each array is populated with the truth table values for a given logic operation. The array values are set by internal voltage connection. The cells in the arrays are populated with the bottom-most row being row zero and the left-most column being column zero. The quaternary inputs to the two drivers, which are on either side of the array, produce control voltages that select an entire row or entire column. The quaternary output of the array is the value in the selected cell. Figure 6.2 shows an example of cell selection in the AND array component. The inputs to the drivers activate the row corresponding to logic 2 and the entire row is now sent to what is called a selector and then the column corresponding to logic 3 is output from the selector, which in this case is a quaternary logic 2.

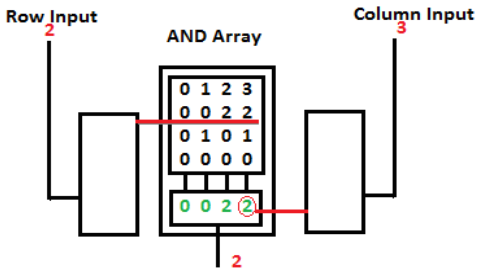

Figure 6.2. Cell Selection in AND Array

All the arrays can have a common row driver and column driver. Hence, the ALU contains the row and column driver and one array for each of the function, that is sum without carry input, sum with carry input, carry out without the carry input, carry out with the carry input, AND, NAND, OR, NOR, XOR, and INV. So, that will be a total of 10 arrays and two drivers. Figure 6.3 shows the block diagram of the ALU. Note that it doesn't show the Invert function. Also, there can be many more functions added to it like compare, subtract with and without borrow, etc.

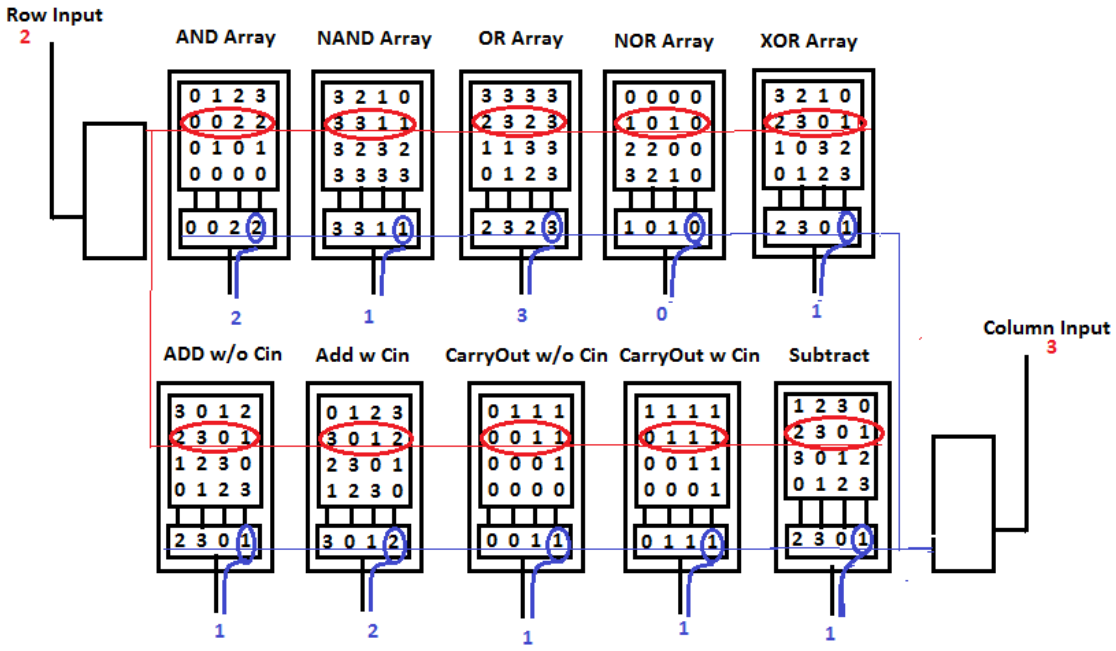

Figure 6.3. Block Diagram of ALU with the Operations And, Nand, Or, Nor, XOR, Add Without Carry Input, Add With Carry Input, Carry Out Without Carry Input, Carry Out With Carry Input, and Subtract

The area of the ALU keeps increasing drastically as we add more functions to it. This is not an optimal solution to reducing the area using the memory-based technology. Hence, the concept of matrix algebra is introduced by recognizing the patterns in the arrays.

6.2. PATTERN RECOGNITION

Pattern recognition is a branch of learning that focuses on the recognition of patterns and regularities in data. We use this knowledge and apply the rotation, transformation or substitution mechanisms to the arrays, along with introducing variables to benefit in the area aspect of the design.

6.2.1 Rotation Of Columns. First, let us consider the example of the Add array with and without carry input. The Figure 6.4 below shows the two arrays.

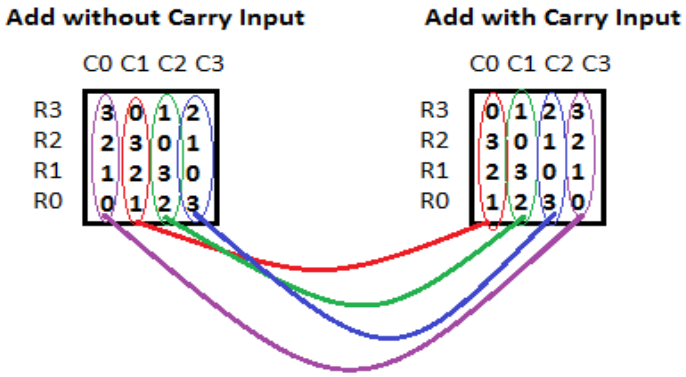

Figure 6.4. Add Arrays With and Without Carry Input and the Similarities in Pattern

The two arrays have the same elements column wise, except that the columns have been shifted to the left by one. This recognition of the pattern helps us in reusing the same array for both with or without the carry input conditions. There will be an addition on a hardware piece called the data mangler, which will be discussed later in the paper. The data mangler helps in shuffling the columns when there is a carry input. When there is no carry input, it passes the values directly based on the row and column inputs. When there is a carry input, the data mangler is activated. In this case, we use the data mangler 1230. This implies that Column 1 in the original array (considering the array without the carry input to be the original one) moves to Column 0 in the mangled array, Column 2 in the original array moves to Column 1, Column 3 in the original array moves to Column 2, and finally Column 0 of the original array moves to Column 3. So, if Row 2 and Column 3 are selected, the result without the carry input must be a 1, but the result with a carry input should give an output of 2.

As shown in Figure 6.5, the two arrays are now merged to just one array. Row 2 is selected, hence the values 2, 3, 0, 1 are sent as inputs to the data mangler. If the carry input is '0', the data mangler is deactivated and the entire row is passed on to the selector without any manipulation. If the carry input is high, the data mangler 1230 is activated and the elements of the row 2, 3, 1, 0 are rotated to the left, hence giving the output value 2 for the column input being 3.

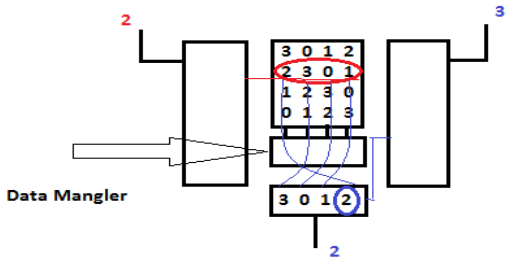

Figure 6.5. Reuse of Hardware by Introducing Data Mangler

6.2.2 Transformation of Matrix. For the next step, let's consider add with the carry input array and the subtract array. Add with the carry input is already using add without the carry input as the base array and the data mangler. Figure 6.6 shows the pattern recognition between add with the carry input array and the subtract array.

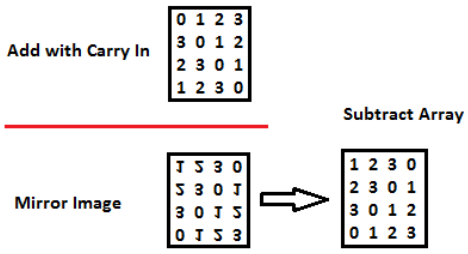

Figure 6.6. Pattern Recognition Between Add and Subtract Arrays

When the add array with the carry input is mirrored along the column, the resultant array is the same as subtract array. So, here a row mangler which is very similar to the data mangler can be used. The row mangler 3210 is used here which implies that when the subtract operation is performed, the row mangler is activated. So, the row 3 in the original array (considering the array with the carry input to be the original one) moves to row 0 in the mangled array, the row 2 in the original array moves to row 1, row 1 in the original array moves to row 2, and finally the row 0 of the original array moves to row 3.

6.2.3 Introduction of Variables. The XOR array is the fourth array which is very similar to the add array. Considering the add without the carry input and the XOR array, there are only four positions that are different from each other, those are the values corresponding to row 1 column 1, row 1 column 3, row 3 column 1, and row 3 column 3. So, here the variables are introduced. When add is activated, first set of values are assigned in the array and when the XOR is activated the second set of values are assigned. Figure 6.7 below shows the introduction of variables through an external line.

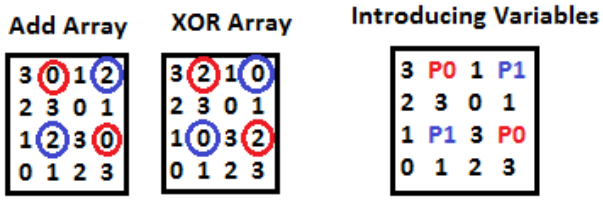

Figure 6.7. Hardware Reuse by Introducing Variables

So, the add array with carry input, add array without the carry input, subtract array, and the XOR array have all been merged as one single array by adding two small pieces of hardware, the data mangler 1230 and the row mangler 3210.

6.2.4 Substitution. The carry out array with and without carry input is considered next and the diagonal from the top left to the bottom right is the only difference between the two arrays, as seen in Figure 6.8. So, if there is no carry input, the elements in the diagonal are tied to low voltage, and when there is a carry input, they are tied to high voltage. Or, it can be simplified as the values in the diagonal being the same as carry input and hence tied to it.

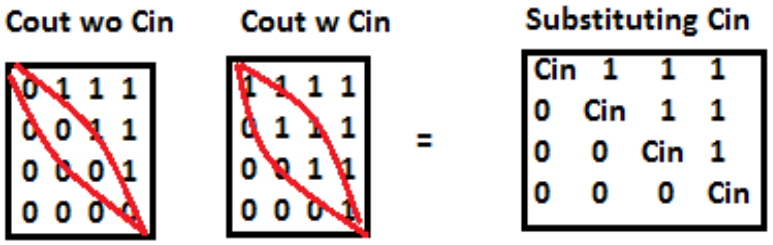

Figure 6.8. Reuse of Carry Arrays by Substituting Values in the Diagonal

Next comes the logical arrays AND, NAND, OR, and NOR arrays. Firstly, consider the AND and NAND arrays. The patterning is very similar in both the arrays except that the values are mangled. All the 0s in the AND array are replaced with 3s, the 1s with 2s, 2s with 1s, and 3s with 0s. So, a value structure mangler is used to take care of these replacements. Figure 6.9 below shows the use of value structure mangle.

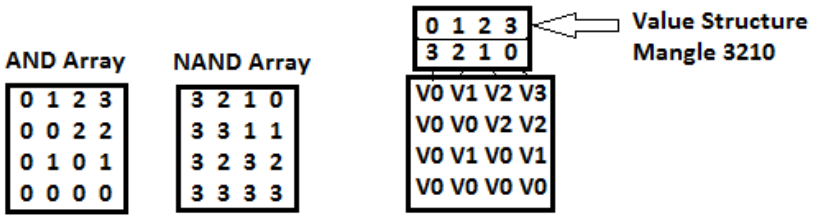

Figure 6.9. Usage of Value Structure Mangle 3210

6.2.5 Multiple Transformations. The AND array and the NOR array need more than one transformation to be able to reuse the hardware. First the rows are mangled using the row mangler 3210 and then a data mangler 3210 is used to achieve the transformation from AND array to the NOR array.

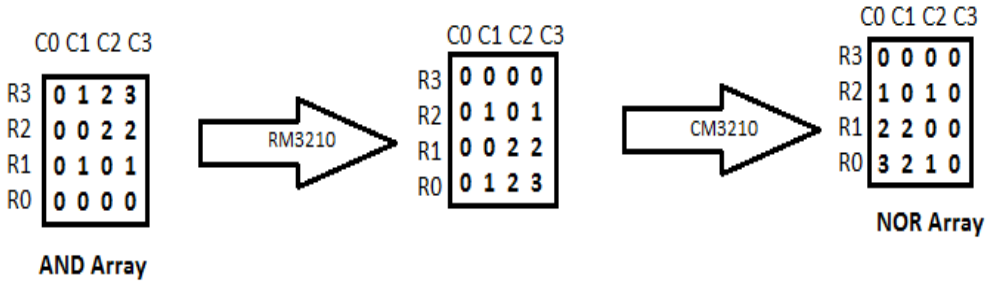

Figure 6.10. Transformation from AND Array to NOR Array

By applying the same value structure mangle 3210 to a NOR Array, the OR Array can be achieved. So, finally the AND array will have a row mangle 3210, a column mangle 3210, and a value structure 3210 to obtain the NAND, NOR, and OR arrays. Also, by forcing a 0 input on the row driver, the NOR array can also act as the inverter.

6.3. CORMEM DIGITAL REASONING ALU

The general arithmetic and logic unit, as described earlier has an input bus, an output bus and control signals. The CorMem digital reasoning ALU also has an input bus and an output bus. But in the Figure 6.11 below, a single digit ALU is considered, which is the equivalent of a two bit binary ALU. The binary-to-quaternary and quaternary-to-binary circuits are used to make the conversions. These converting circuits are shown in chapter 5.

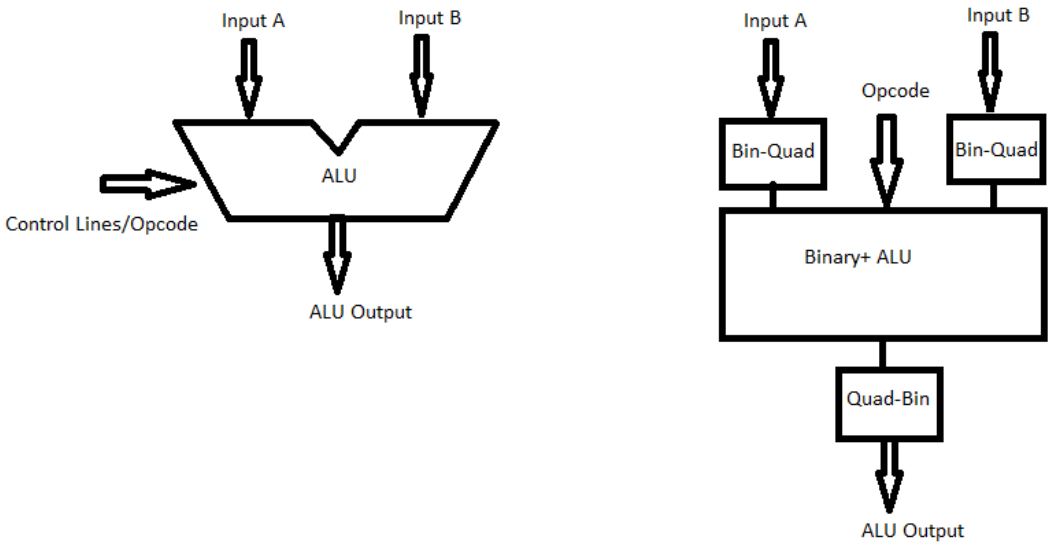

Figure 6.11. General ALU Block Diagram for Binary and CorMem Digital Reasoning

As described in the above section, with the application of matrix algebra and by adding some hardware, the 11 arrays can be minimized to just 3 arrays, thus reducing the area by almost 70%. The block diagram of the final digital reasoning ALU after all the applied transformation is shown in the Figure 6.12 below.

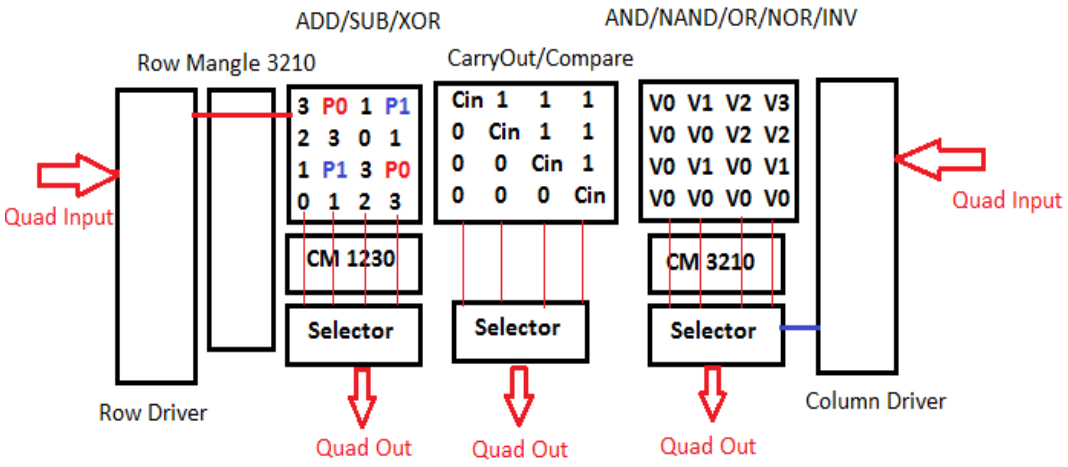

Figure 6.12. CorMem Digital Reasoning ALU After the Reuse of Hardware and Transformations

6.4. IMPLEMENTATION OF HARDWARE

The CorMem digital reasoning technology can be implemented using CMOS transistors and the already available fabrication techniques. There is no special requirement for the design or development purposes. The driver uses three sense amplifiers. Each sense amplifier outputs one pair of signals, data and reference, which are inverted to each other. So, the driver outputs three pairs of data and reference signals. The detailed architecture of the driver is described in chapter 4. Based on the reference voltages and the input quaternary data, the six output lines take either a '0' value or a '1' value. The Table 6.2 below shows the corresponding output values depending on the quaternary input.

Table 6.2 Data and Reference Values Corresponding to Quad Input

	Quad Input 0	***Quad Input 1***	***Quad Input 2***	***Quad Input 3***
Dat2	0	0	0	1
Ref2	1	1	1	0
Dat1	0	0	1	1
Ref1	1	1	0	0
Dat0	0	1	1	1
Ref0	1	0	0	0

6.4.1. Six-To-Eight Decoder. The Array has four rows of pass gates, with each row having four pass gates. There are four pairs of select lines; each pair being inverted lines that control one row of pass gates. So, there are a total of four pairs of lines. The six output lines from the driver are converted to the eight lines by using a six-to-eight decoder. The six-to-eight decoder consists of four NAND gates. The output of each NAND gate is one of the select lines for the rows of the array. It is passed through an inverter to produce the inverted pair. The decoder is also used on the column driver and it is used to select one of the four pass gates on the selector. The schematic of the decoder is shown in the Figure 6.13 below.

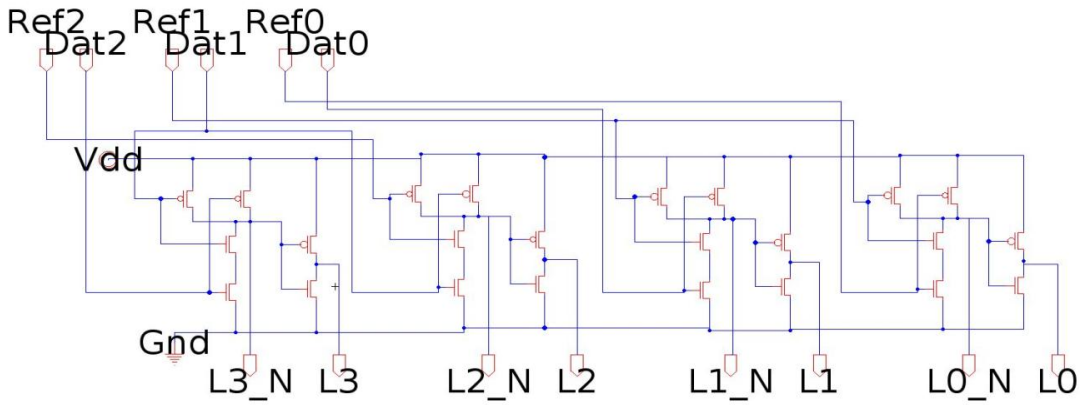

Figure 6.13. Six-To-Eight Decoder

The six-to-eight decoder outputs the values such that no more than one row of the array is selected at any given point of time. Table 6.3 showing the outputs based on the quaternary input is shown below:

Table 6.3. Decoder Output Values Corresponding to Quad Input

	Quad Input 0	***Quad Input 1***	***Quad Input 2***	***Quad Input 3***
L3	0	0	0	1
L3_N	1	1	1	0
L2	0	0	1	0
L2_N	1	1	0	1
L1	0	1	0	0
L1_N	1	0	1	1
L0	1	0	0	0
L0_N	0	1	1	1

The waveform below in Figure 6.14 shows the quaternary input and the corresponding output of the six-to-eight decoder. All the possible four inputs are given to see the transitions in the output. The top three waveform windows show the data and reference input pairs to the decoder. The bottom four waveform windows are the outputs from the decoder that serve as the inputs to the arrays.

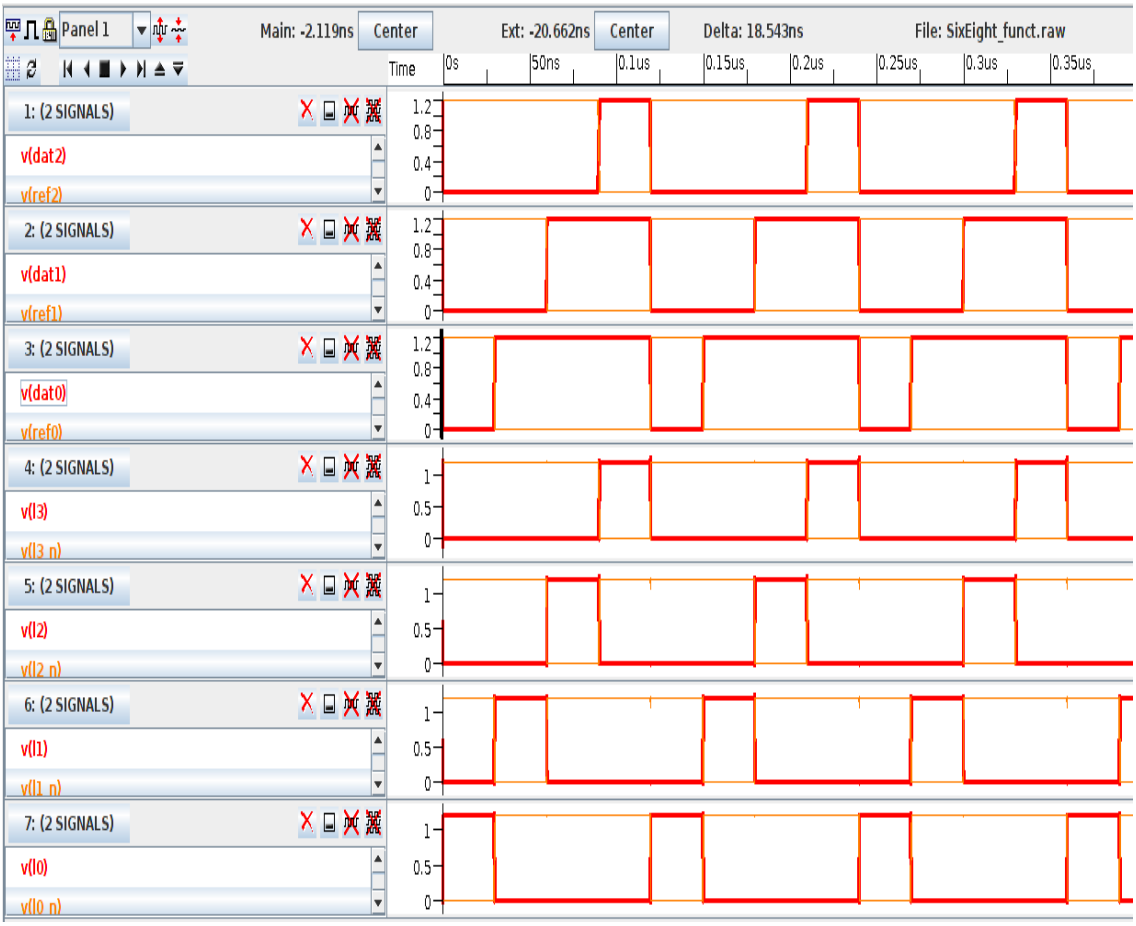

Figure 6.14. Simulation Results of Six-To-Eight Decoder

6.4.2. Mangler. The mangler circuit is basically used to shuffle the reference and data values to select the required row or column. As mentioned earlier, the add array and add with carry array can be implemented as one array by just adding a small piece of hardware which is used to shuffle the columns.

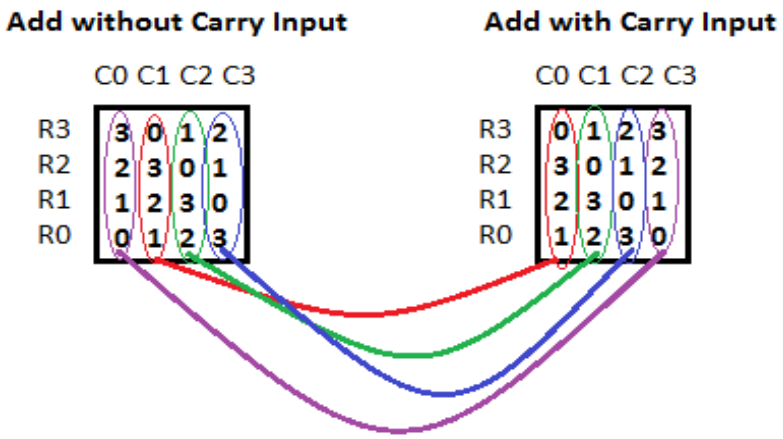

Figure 6.15. Pattern Recognition Between the Arrays

For this purpose, a data mangler 1230 is used. That implies that, if the driver input on the column side is a 0, column 1is selected. If the column input is a 1, column 2 is selected; if the column input is a 2, column 3 is selected, and if column input is a 3, column 0 is selected.

Similarly, for the subtract array, as mentioned earlier, a 3210 mangler is required on the row driver end. Hence, it is called row mangler 3210. A mangler is made of six double pass gates. If the select line for the mangler is low, the original data and reference values are passed to the output. If the select line is high, the shuffled data and reference values are passed to the output. Figure 6.16 below shows the circuit of a mangler 3210. Appendix D shows the layout of the mangler circuit along with the layout of the complete ALU block.

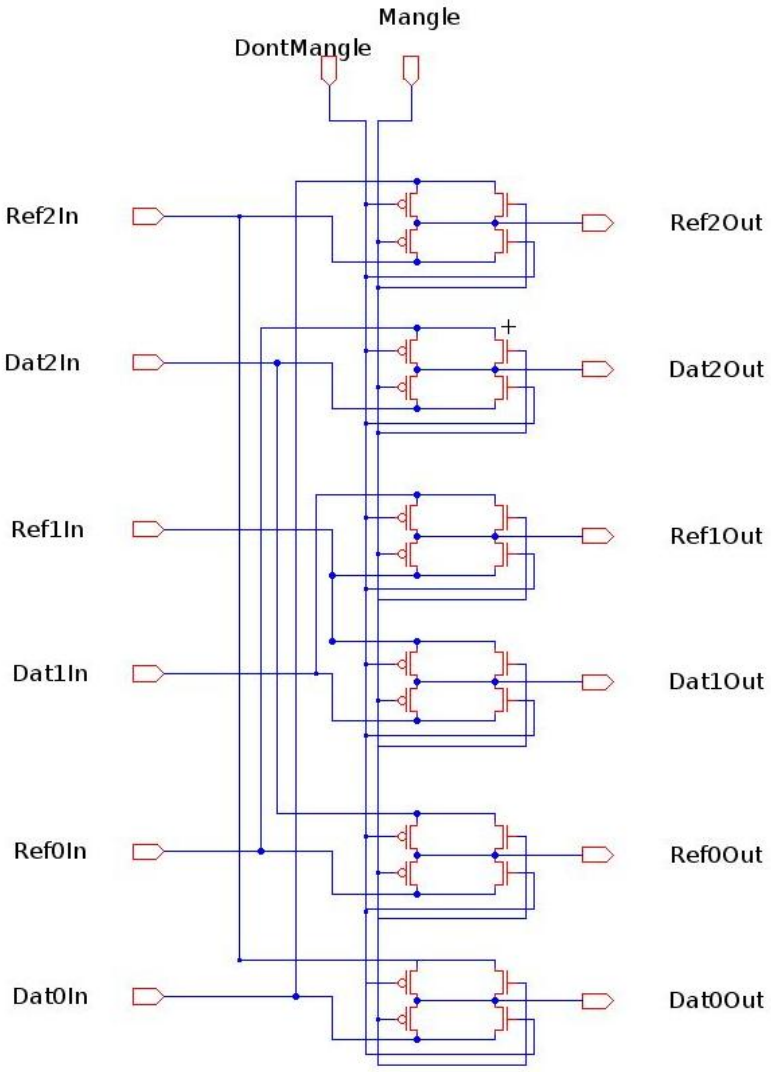

Figure 6.16. Schematic of Mangler 3210

The simulation results of the mangler are shown below. The data and reference value pairs are always complementary. Hence, the waveform below in Figure 6.17 shows only the data lines. The top three lines are the output lines. It can be seen that when the mangle input is a ‘0’, the data input values pass to the output and when the mangle input is high, the shuffled values go to the output.

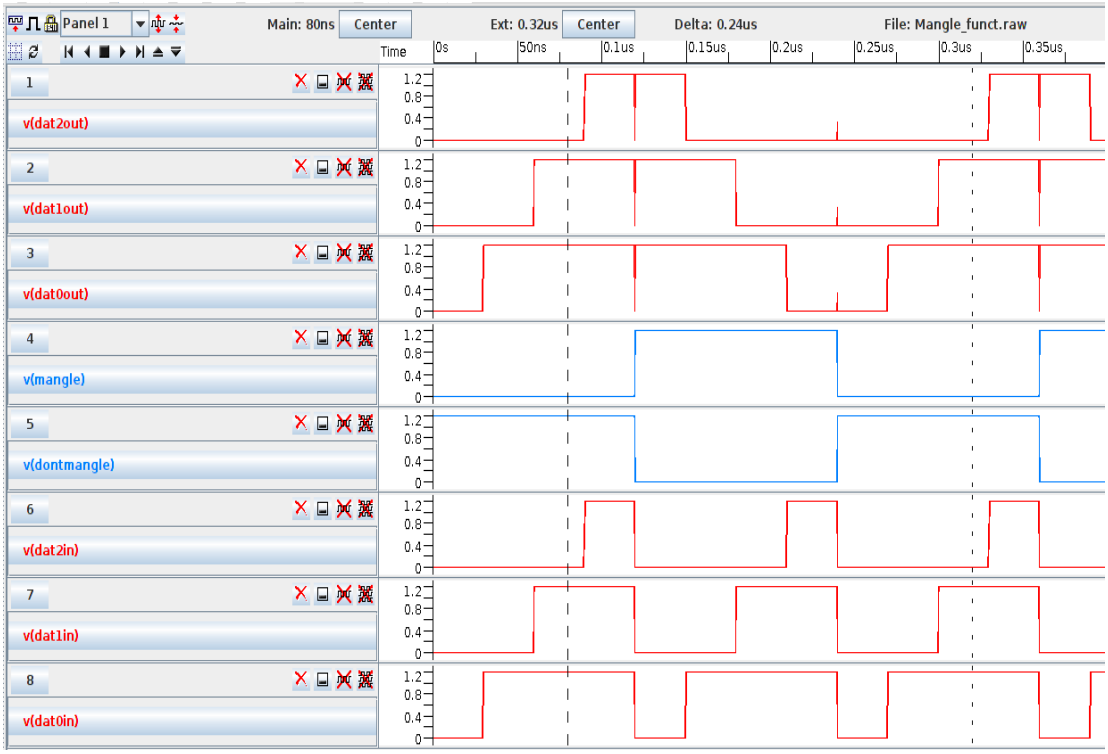

Figure 6.17. Simulation Results of Mangler 3210

6.5. RESULTS AND DISCUSSION

The CorMem digital reasoning technology, as shown in the above sections, effectively reuses the hardware to perform all the logical operations as described and also is capable of more operations. It can be used to perform XNOR, compare functions (less than, greater than, or equal to). It also performs add with and without the carry input, subtract function, and the drivers act not only as the decoders but also as the registers that are used to store the data values.

The CorMem digital reasoning uses 1200 transistors to implement all the mentioned functions and the advantage of using this technology takes a leap when interconnections are considered. The interconnections are arranged in a very orderly manner, and the area of the ALU is based on the transistor density. The numbers of corners are also constrained to a minimum number. The layout is developed in IBM 130nm technology and the number of metal layers used

is limited to up to 3. The area of the CorMem digital reasoning ALU is 1076 μm^2. The binary version of the ALU that performs the same functions has an area of 7800 μm^2.

Figure 6.18 below shows the simulation of the complete ALU, with two quaternary inputs coming in from two drivers. The decoded values go to the arrays and the manglers are either activated or not depending on which function is to be performed. The simulation is shown for the row inputs changing from value 1 to value 3, the column input going from a 0 to 2. The rowmangle and datamangle signals are either both ON or both OFF. When they are both OFF, the logic AND is being performed and when they are both ON, the logic NOR is being performed. The arrayout0, arrayout1, arrayout2, and arrayout3 show the outputs of the array. The logicout0, logicout1, logicout2, and logicout3 show the outputs of the data mangler. The logicout is the final output of the selector.

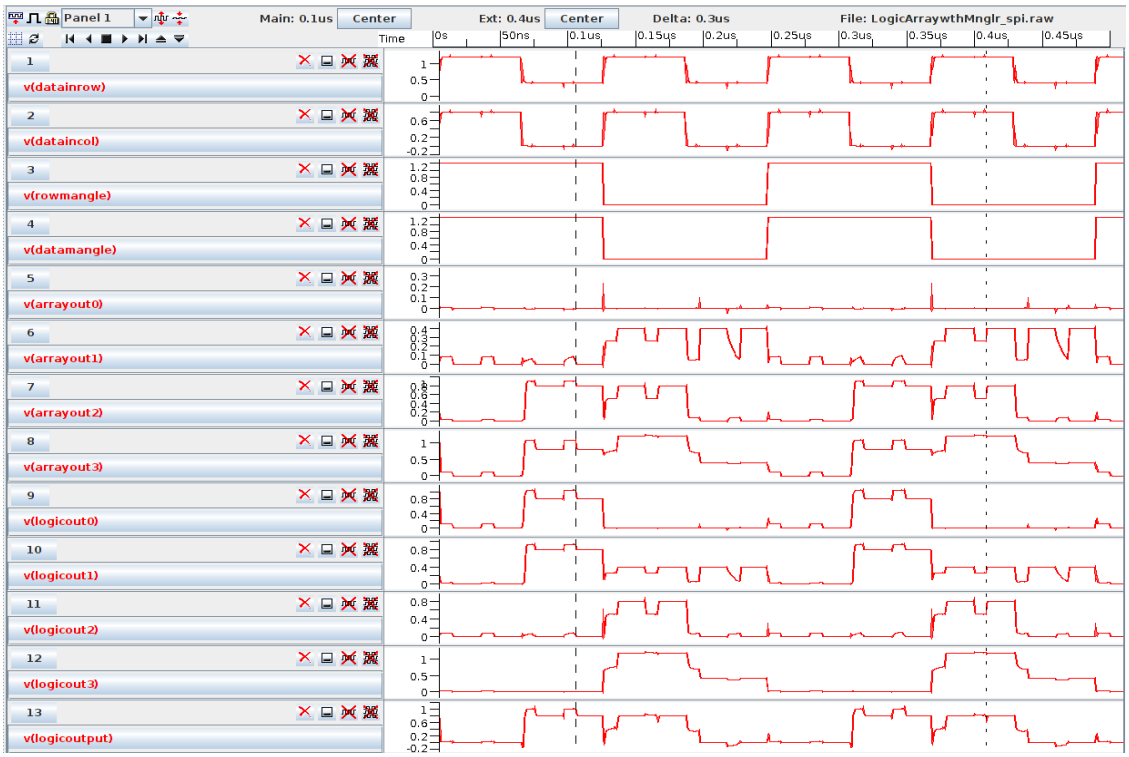

Figure 6.18. Simulation Results of the ALU Programmed as AND and NOR

The Figure 6.19 shows just the inputs and the final output with the mangle signals. At the first cursor, the row input is a 1 and the column input is a 0. The mangle signals are both high. So, the NOR logic should be performed. From the Table 6-1, the NOR output of 1 and 0 is a value 2. The value 2 is at 0.8V and that is proved to be true from the simulation below. At the second cursor, the row input is a 3 and the column input is a 2. The mangle signals are both low. So, the AND logic should be performed. From the Table 6.1, the AND output of 3 and 2 is a value 2. The value 2 is at 0.8V and that is also proved to be true from the simulation below. At time 0.2µs, the row input is a 1 and the column input is a 0 with row and data mangler signals being low. So, the AND operation is performed between 0 and a 1 which should give a 0 output.

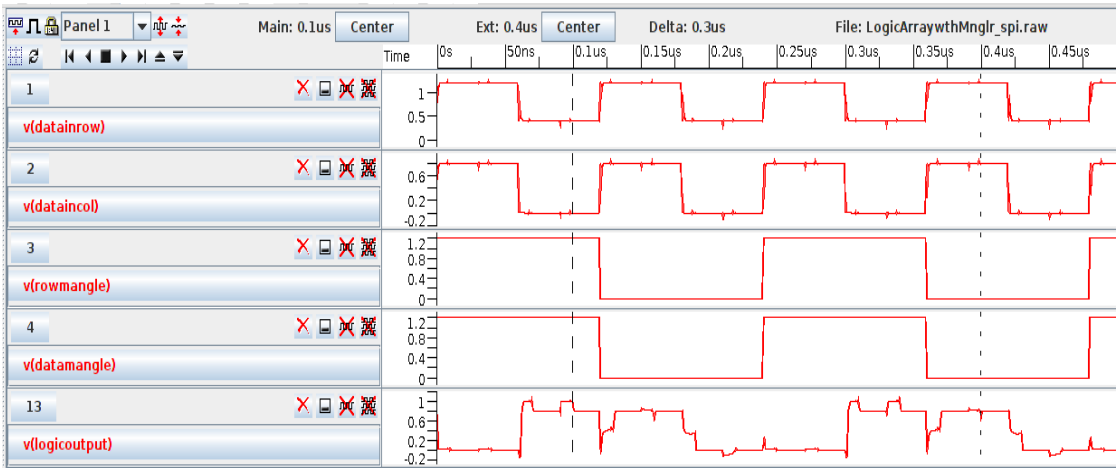

Figure 6.19. Output of the ALU Programmed as AND and NOR

7. ALU CONTROLLER

This chapter describes the control structure of the arithmetic and logic unit from the processor point-of-view. The ARM Cortex M0 has been taken as a model to duplicate the functionalities in the digital reasoning method. The M0 is a 32-bit processor and the Arithmetic and Logic Unit shown in the previous chapter is a two-bit or a one-digit ALU. There are sixteen of them stacked one over the other to replicate the 32-bit processor. The ALU programs the control lines depending on the opcode that has been executed. So, all these control lines are programmed using this control unit. The inputs to this control unit will be the local opcode, which is different from the 16-bit ARM M0 opcode. The outputs will be going to the control signals of the different components of the ALU. There is just one control unit sitting on the top of the stacked 16-digit ALU.

7.1. ALU CONTROLS

The Arithmetic and Logic Unit (ALU) described in the previous chapter is capable of performing the logic functions AND, NAND, OR, NOR, and XOR. The ALU also performs addition and subtraction with the option of either having a carry input or not. The compare functions like equal to, greater than, less than can also be determined. The ALU with the hardware reuse and with all the above functionalities being performed has three arrays with the selectors, two drivers, a row mangler 3210, a data mangler 1230 corresponding to the digit array and a data mangler 3210 corresponding to the logic array. The logic array also has a value structure mangle 3210 sitting on the top of the logic array. The digital reasoning one digit or two-bit ALU is shown below (value structure mangle not shown) in Figure 7.1.

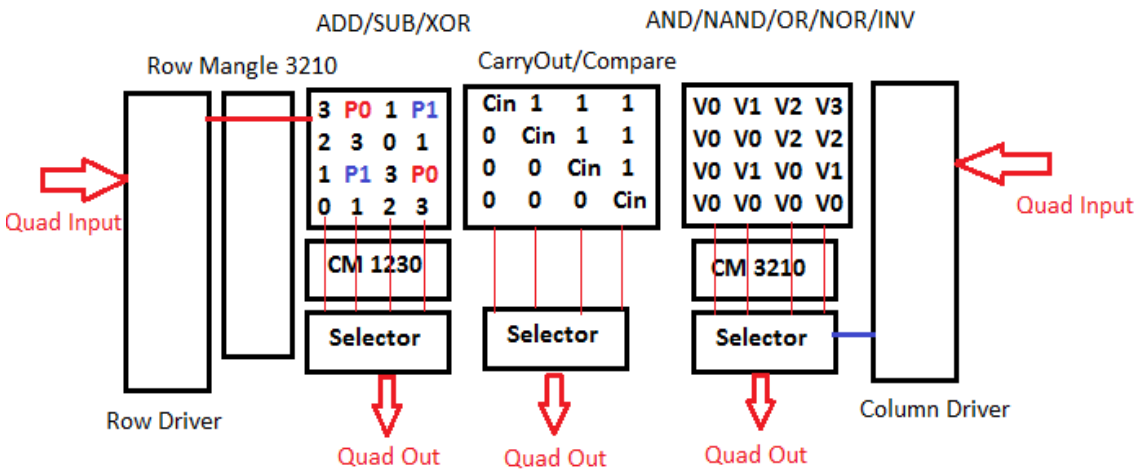

Figure 7.1. CorMem Digital Reasoning ALU after the Reuse of Hardware and Transformations

Note that, sixteen of the blocks shown above will be stacked on each other to create one complete 32-bit ALU. The ALU output will be transferred onto a data bus and hence all the outputs must be connected. For this reason, there is an isolation circuit placed for the digit array and the carry array and another isolation circuit for the logic array. The isolation circuit, if in the ON state, will isolate the arrays from the drivers. Hence, the output lines from the drivers cannot reach the array and the selector and the final output is in high impedance mode. So, if the ADD operation is being performed, the isolation circuit which is attached to the logic array is turned ON, isolating the logic array. But, the isolation circuit corresponding to the digit array is OFF, hence the output of the digit array is sent onto the selector and the output of the selector goes on to the data bus. Also, the digit array and the carry array are both ON at the same time or OFF at the same time. So, just one isolation circuit will be sufficient for both these arrays together. The output of the carry array from the least significant digit goes as input to the next digit's data mangler 1230 enable signal and to the Cin of the carry array as it represents whether there is a

carry input or not. In addition, there is a forced zero circuit added which is used for some of the functionalities which require only a single input. So, the next version of ALU looks like the Figure 7.2 below.

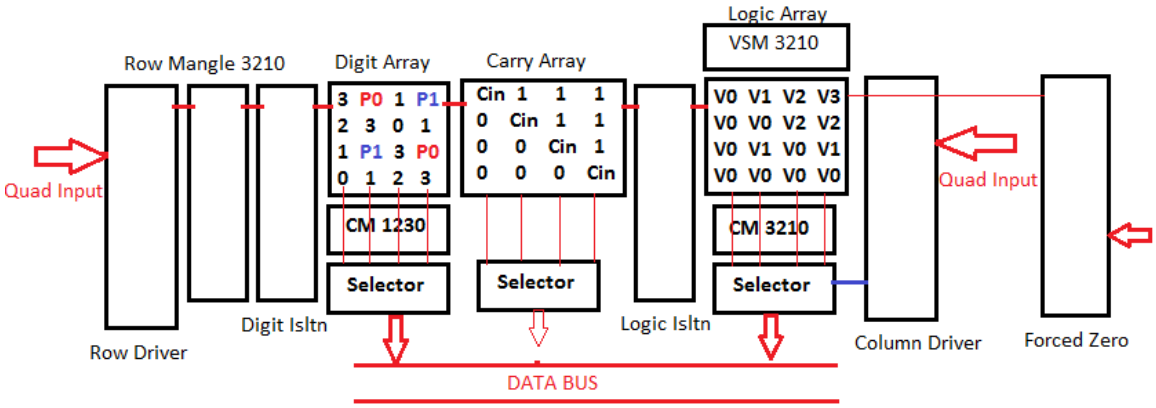

Figure 7.2. CorMem Digital Reasoning ALU with Isolations

The above ALU is set up to perform all the arithmetic and logic operations that are performed by an ARM Cortex M0 processor. Table 7.1 below shows the list of ARM operations that are carried out in the ALU.

Table 7.1. ARM Operations in the ALU

	ARM Cortex M0 Instruction	**Basic Action**
1	ADDS Rd, Rs, Rn	Addition
2	ADDS Rd, Rs, #offset3	Addition
3	SUBS Rd, Rs, Rn	Subtraction

Table 7.1. ARM Operations in the ALU (cont.)

4	SUBS Rd, Rs, #offset3	Subtraction
5	CMP Rd, #offset8	Compare
6	ADDS Rd, Rd, #offset8	Addition
7	SUBS Rd, Rd, #offset8	Subtraction
8	ANDS Rd, Rd, Rs	Logic AND
9	EOR Rd, Rd, Rs	Logic Ex-OR
10	ADCS Rd, Rd, Rs	Addition with
11	SBCS Rd, Rd, Rs	Subtraction
12	TST Rd, Rs	Set condition
13	RSBS Rd, Rs, #0	Negation
14	CMP Rd, Rs	Set condition
15	CMN Rd, Rs	Set condition
16	ORRS Rd, Rd, Rs	Logic OR
17	BICS Rd, Rd, Rs	Rd = Rd AND
18	MVNS Rd, Rs	Rd = NOT Rs
19	ADD Rd, Rd, Hs	Addition
20	ADD Hd, Hd, Rs	Addition
21	ADD Hd, Hd, Rs	Addition
22	CMP Rd, Hs	Compare
23	CMP Hd, Rs	Compare
24	CMP Hd, Hs	Compare
25	ADD Rd, R15, #Imm	Addition to
26	ADD Rd, R13, #Imm	Addition to
27	ADD R13, R13, #Imm	Addition to
28	SUB R13, R13, #Imm	Subtraction

The next step is to figure out all the control/enable lines of the ALU that are to be programmed using the ALU controller. The row mangler 3210 has one enable line which will go ON for the operations like subtract, the digit isolation has one control line which will turn the isolation OFF for all the functions that take care of any type of addition, subtraction, Ex-Or, compare etc. The P0 and P1 lines of the digit array need a control line to be able to switch the values from a 0 and 2. If the control line is OFF P0 takes a 2 value (0.8V) and P2 takes a 0 value (0V) or vice versa if the control line is ON. This control line will be an ON for Ex-Or functionality. The Cin is one of the control lines as well which is useful while performing the operations like add with carry or a negation function. Then, there is a control line for logic isolation which will turn the isolation OFF for all the functions that take care of any type logical operations except Ex-Or. The value structure mangle has one enable signal which when OFF gives the value 0, 1, 2, and 3 to V0, V1, V2, and V3 cells respectively. When it is ON, the values are mangled and 0, 1, 2, and 3 are given to V3, V2, V1, and V0 respectively. The data mangler 3210 for the logic array has one enable line. The forced zero circuit has a control line which is enabled for operations like invert and negation. Thus, there are a total of eight control lines to be programmed by the controller.

7.2. LOCAL OPCODE BY GROUPING

The 28 instructions shown in Table 7.1 will be given a local opcode and some of the instructions can be grouped to a single opcode if all the instructions in this particular group enable the same set of signals. For example, all the ADDS along with the CMN can be grouped into one opcode as the basic operation performed is addition. Table 7.2 gives the re-ordered instructions and the signals that are enabled for those particular instructions.

Table 7.2. Control Signals

	ARM Cortex M0 Instruction	**RM 3210**	**Isltn Digt Low**	**P0P1 For Digit**	**Cin**	**Isltn Logic Low**	**VSM 3210**	**DM 3210**	**FZ**
1	ADDS Rd, Rs, Rn	0	1	0	0	0	0	0	0
2	ADDS Rd, Rs, #offset3	0	1	0	0	0	0	0	0
3	ADDS Rd, Rd, #offset8	0	1	0	0	0	0	0	0
4	CMN Rd, Rs	0	1	0	0	0	0	0	0
5	ADD Rd, Rd, Hs	0	1	0	0	0	0	0	0
6	ADD Hd, Hd, Rs	0	1	0	0	0	0	0	0
7	ADD Hd, Hd, Rs	0	1	0	0	0	0	0	0
8	ADD Rd, R15, #Imm	0	1	0	0	0	0	0	0
9	ADD Rd, R13, #Imm	0	1	0	0	0	0	0	0
10	ADD R13, R13, #Imm	0	1	0	0	0	0	0	0
11	SUBS Rd, Rs, Rn	1	1	0	1	0	0	0	0
12	SUBS Rd, Rs, #offset3	1	1	0	1	0	0	0	0
13	CMP Rd, #offset8	1	1	0	1	0	0	0	0
14	SUBS Rd, Rd, #offset8	1	1	0	1	0	0	0	0
15	CMP Rd, Rs	1	1	0	1	0	0	0	0
16	CMP Rd, Hs	1	1	0	1	0	0	0	0
17	CMP Hd, Rs	1	1	0	1	0	0	0	0
18	CMP Hd, Hs	1	1	0	1	0	0	0	0
19	SUB R13, R13, #Imm	1	1	0	1	0	0	0	0
20	ANDS Rd, Rd, Rs	0	0	0	0	1	0	0	0
21	TST Rd, Rs	0	0	0	0	1	0	0	0
22	EOR Rd, Rd, Rs	0	1	1	0	0	0	0	0

Table 7.2. Control Signals (cont.)

	ARM Cortex M0 **Instruction**	**RM 3210**	**Isltn Digt Low**	**P0P1 For Digit**	**Cin**	**Isltn Logic Low**	**VSM 3210**	**DM 3210**	**FZ**
23	ADCS Rd, Rd, Rs	0	1	0	1	0	0	0	0
24	SBCS Rd, Rd, Rs	1	1	0	0	0	0	0	0
25	RSBS Rd, Rs, #0	1	1	0	0	0	0	0	1
26	ORRS Rd, Rd, Rs	1	0	0	0	1	1	1	0
27	BICS Rd, Rd, Rs	1	0	0	0	1	0	0	0
28	MVNS Rd, Rs	1	0	0	0	1	0	1	1

The operations in Table 7.2 have been grouped and it can be observed that the instructions 1 through 10 have the same set of signals being fired, and hence can be given a single opcode. Instructions 11 through 19 have the same set of signals being activated, hence giving it the second opcode. Instructions 20 and 21 can be grouped and then 22 through 28 are all different taking their own opcode. So, there will be a total of 10 opcodes. Two digits are required to give unique codes to ten different instructions. Table 7.3 below shows the temporary opcode assignment to the instructions. The instructions 1 through 10 will be just referred to as ADD, 11 through 19 will be referred to as SUB, 20 and 21 will be referred to as AND, and the instructions from 22 will be referred to as EOR, ADC, SBC, NEG (as in negation), OR, BIC, and NOT respectively.

Table 7.3. Temporary Opcode for the Instructions – Iteration 1

	Instruction	Temp Opcode	**RM 3210**	**Isltn Digt Low**	**P0P1 For Digit**	**Cin**	**Isltn Logic Low**	**VSM 3210**	**DM 3210**	**FZ**
1	ADD	00	0	1	0	0	0	0	0	0
2	SUB	01	1	1	0	1	0	0	0	0
3	AND	02	0	0	0	0	1	0	0	0
4	EOR	03	0	1	1	0	0	0	0	0
5	ADC	10	0	1	0	1	0	0	0	0
6	SBC	11	1	1	0	0	0	0	0	0
7	NEG	12	1	1	0	0	0	0	0	1
8	OR	13	1	0	0	0	1	1	1	0
9	BIC	20	1	0	0	0	1	0	0	0
10	NOT	21	1	0	0	0	1	0	1	1

Later, the instructions in the above table are arranged in the form of the arrays and will be given the opcode depending on which configuration would result in the minimal hardware area. The arrays, rather than having a 4x4 configuration can be reduced by some minimization techniques which will be explained in the next section. Once the minimization techniques are applied, the reduced arrays will be designed, called truncated arrays.

7.3. REALIZATION OF FINAL VERSION OF OPCODE

The control signals data is arranged in the form of an array, thus having to implement eight arrays, each for one control line from Table 7.3. The opcode is fed to the drivers. Since, there are two digit opcodes, two drivers are required. The drivers select the appropriate values from the eight arrays, hence programming the ALU to perform the required operation. The array

is configured such that the first digit selects the row and the second digit selects the column. The bottom most row of the array is referred to as row 0 and the top most being referred to as row 3. The left most column of the array is referred to as column 0 and the right most being column 3. So, the first iteration of opcodes from Table 7.3 can be arranged in the form of arrays as shown in Figure 7.3.

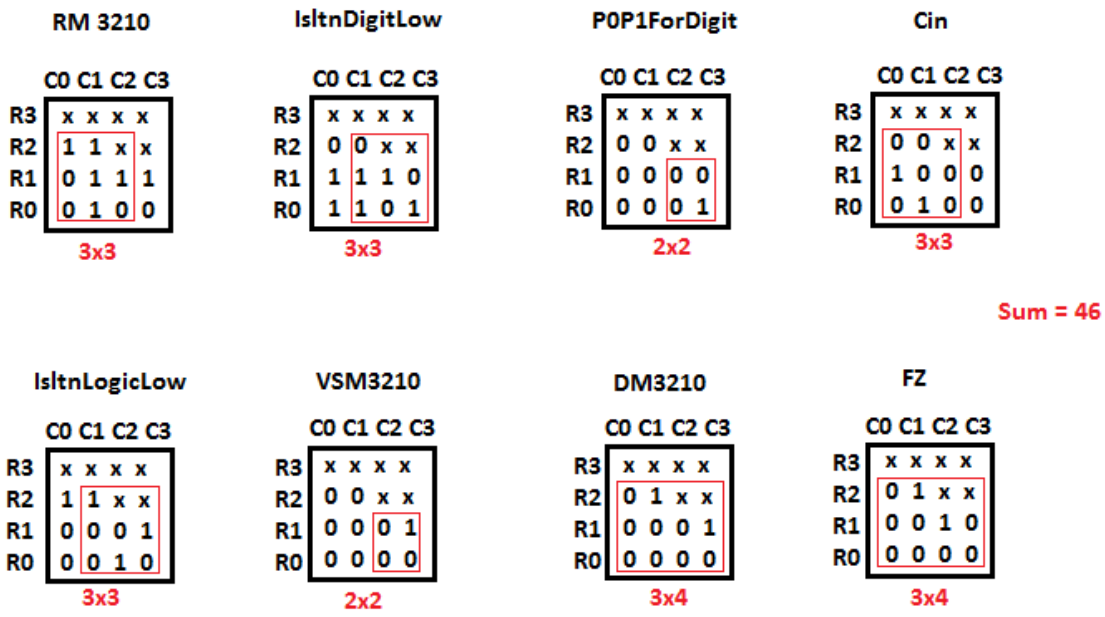

Figure 7.3. Control Signals Data in Array Format – Iteration 1

From the 4x4 arrays above, a pattern can be recognized to minimize the arrays. There are only ten operations required but sixteen different possibilities available. So, the additional six available positions take don't care values represented by 'x'. That implies that the array can already be minimized to just three rows. For further minimization technique, as an example, consider the row mangle 3210 control array. It can be seen that the column 2 and column 3 have

the same values, also row 2 and row 3 can be reduced to just row 2 because of the don't cares. Hence, the RM3210 array can be reduced to a 3x3 array. Also, consider the P0P1ForDigit array as the next example. Row 1, row 2, and row 3 have the same set of values; hence they can all be reduced to just row 1. The column 0, column 1, and column 2 have the same set of values; hence they are reduced to just column 2. The final array is reduced to just row 0 and row 1, and column 2 and column 3, ultimately giving a 2x2 array. Similarly, all the other arrays can be reduced and the highlighted area in the red box shows the minimized version of the arrays. Later, the temporary opcodes are shuffled to see which configuration would give the best case minimal arrays. As a reference, the total number of rows and columns are added to see which version gives the minimum sum and that would be the final version of opcode. In the first iteration the sum of the rows and columns of all the arrays add up to 46.

For the second iteration, the opcode 02 and 03 are interchanged. That implies that AND takes the opcode 03 and EOR takes 02. With this change, the arrays look like in Figure 7.4.

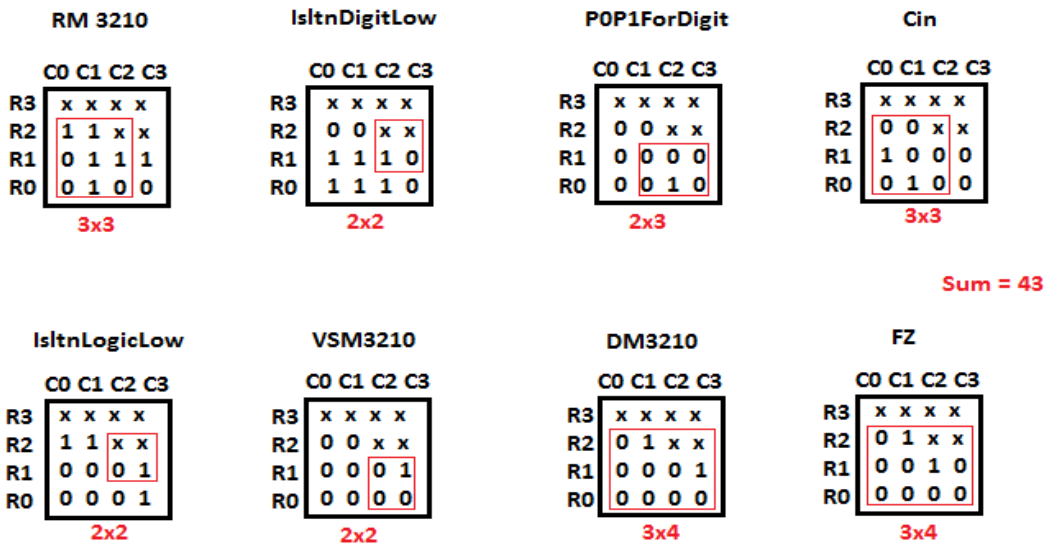

Figure 7.4. Control Signals Data in Array Format – Iteration 2

After a series of iterations, the best configuration of the arrays has been deducted. This final configuration has the sum of the rows and columns as 31. The final arrays are a 2x1 array for row mangle 3210 enable signal, a 2x1 array for isolation of digit array control line, a 2x3 array for P0/P1 lines for digit array, a 2x2 array for Cin, a 2x1 array for isolation of logic array control line, a 2x2 array for value structure mangle 3210 enable signal, a 2x2 array for data mangle 3210 enable signal, and a 2x3 for a forced zero enable signal. The arrays are shown in Figure 7.5 below.

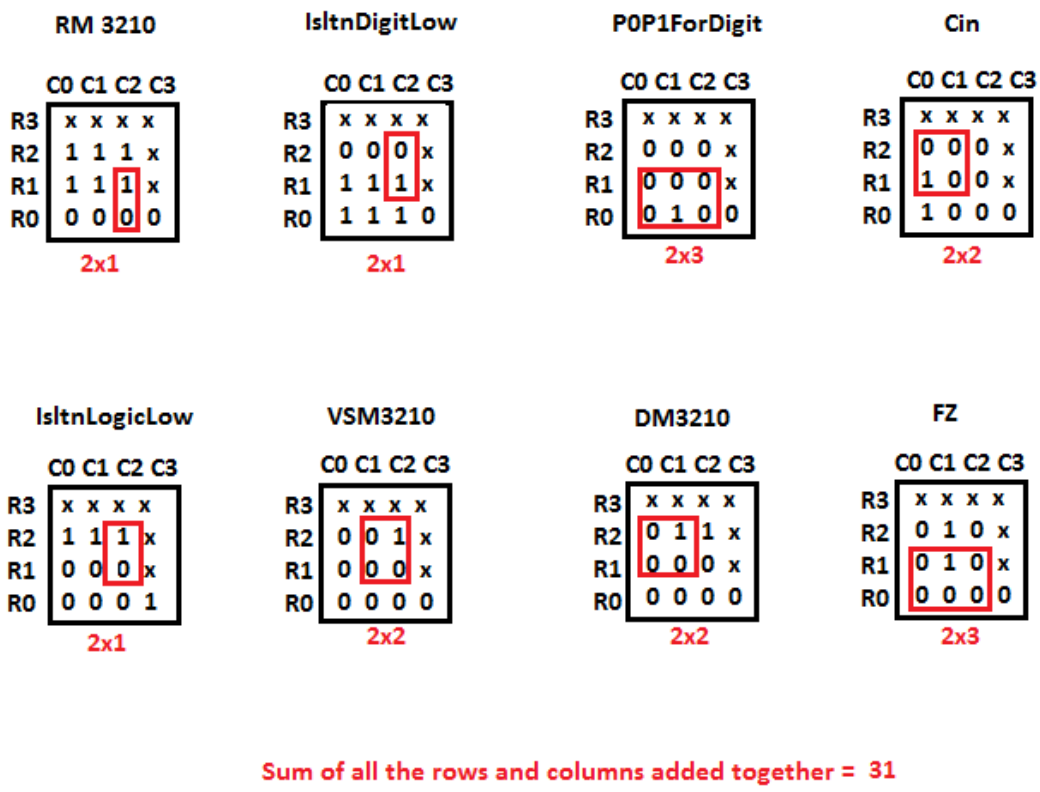

Figure 7.5. Control Signals Data in Array Format – Final Iteration

The final version of the opcode corresponding to this configuration is shown in Table 7.4 below. The opcodes 13, 23, 30, 31, 32, and 33 are not used. The most significant digit selects the row of the array and the least significant digit selects the column of the array.

Table 7.4. Final Version Opcode for the Instructions

	Instruction	Opcode	**RM 3210**	**Isltn Digt Low**	**P0P1 For Digit**	**Cin**	**Isltn Logic Low**	**VSM 3210**	**DM 3210**	**FZ**
1	ADC	00	0	1	0	1	0	0	0	0
2	EOR	01	0	1	1	0	0	0	0	0
3	ADD	02	0	1	0	0	0	0	0	0
4	AND	03	0	0	0	0	1	0	0	0
5	SUB	10	1	1	0	1	0	0	0	0
6	NEG	11	1	1	0	0	0	0	0	1
7	SBC	12	1	1	0	0	0	0	0	0
8	BIC	20	1	0	0	0	1	0	0	0
9	NOT	21	1	0	0	0	1	0	1	1
10	OR	22	1	0	0	0	1	1	1	0

7.4. TRUNCATED ARRAYS

The arrays which are minimized based on the similarity in the patterns are called truncated arrays. The truncated arrays are built using the pass gates, similar to the arrays. Here, instead of the arrays holding 16 values in 16 cells, they are reduced depending on the size of the truncated array. The configuration of the truncated arrays can be varied widely based on the size and the location of the minimization. In the Figure 7.6 two completely different arrays are taken as an example. In the array ‘a’, row 0 and row 1 are the same and hence the final array has been

truncated to three rows. Also the column 0 and column 1 are the same and the array is truncated to three columns, finally resulting in a 3x3 array. Similarly, the array ‘b’ has row 0 and row 1 with the same values, and column 2 and column 3 with the same values, resulting in a 3x3 truncated array. The truncated arrays of both the arrays are shown in the figure below.

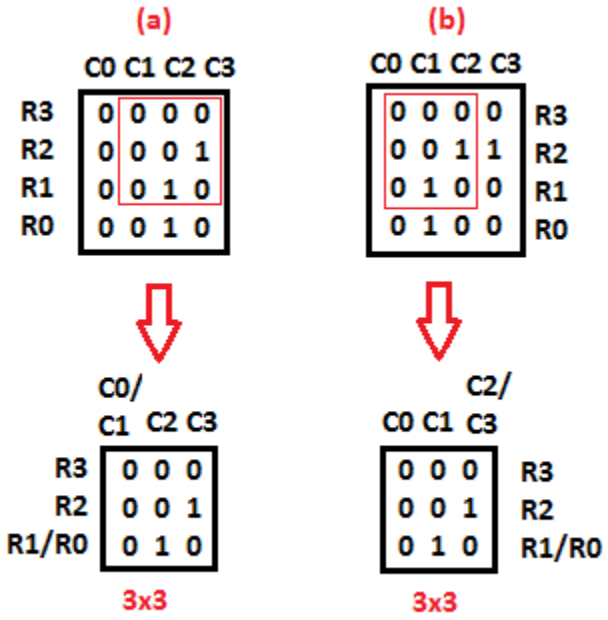

Figure 7.6. Example Arrays that are Truncated to 3x3 Arrays

The truncated arrays that have been derived from the array ‘a’ and array ‘b’ seem to be similar though the original arrays are completely different. The difference in the truncated arrays would be how they are programmed. For the truncated array ‘a’, column 0 and column 1 are combined, whereas for truncated array ‘b’ column 2 and column 3 are combined. To achieve this, first consider the reference and data values from the driver for a given quaternary input, which is shown in Table 7.5.

Table 7.5. Driver Outputs for Corresponding Input Data

	Input Data 0	Input Data 1	Input Data 2	Input Data 3
Data 2	0	0	0	1
Reference 2	1	1	1	0
Data 1	0	0	1	1
Reference 1	1	1	0	0
Data 0	0	1	1	1
Reference 0	1	0	0	0

Considering the truncated arrays, row 0 and row 1 are the same, so the first step is to deduct the common data and reference values for the digit 0 and digit 1. From the table above, Data 1 and Reference 1 are the lines that are same for input data 0 and 1. Hence, these lines should be the control lines for the pass gates to pass the values in row0/row1. For the values from row 2 to pass to the output of the array, reference 2 and data 1 should both be high or data 2 and reference 1 should both be low. So we can use double pass gates. For row 3 to pass, data 2 will be a high. The output values from these three sets of pass gates will be passed on to another three sets of pass gates representing the column data. This is where the arrays ‘a’ and ‘b’ would differ. For columns 0 or 1 to pass through and get to the output from the array ‘a’, the first set of pass gates are controlled by the lines data 1 and reference 1 coming from the column driver. For column 2 to pass to the output, reference 2 and data 1 from the column driver should both be high or data 2 and reference 1 should both be low. For row 3 to pass, data 2 will be a high. Whereas for array ‘b’, for column 0 to pass to the output, the control lines of the pass gates are controlled by signals data 0 and reference 0. For column 1 to get through, data 0 and reference 1 should both be high or reference 0 and data 1 should both be low. For columns 2 and 3 to pass to the output,

the control lines of the pass gates should be tied to data 1 and reference 1 with data 1 being high and reference 1 being low. The block diagram of the array ‘a’ is shown in the Figure 7.7 below.

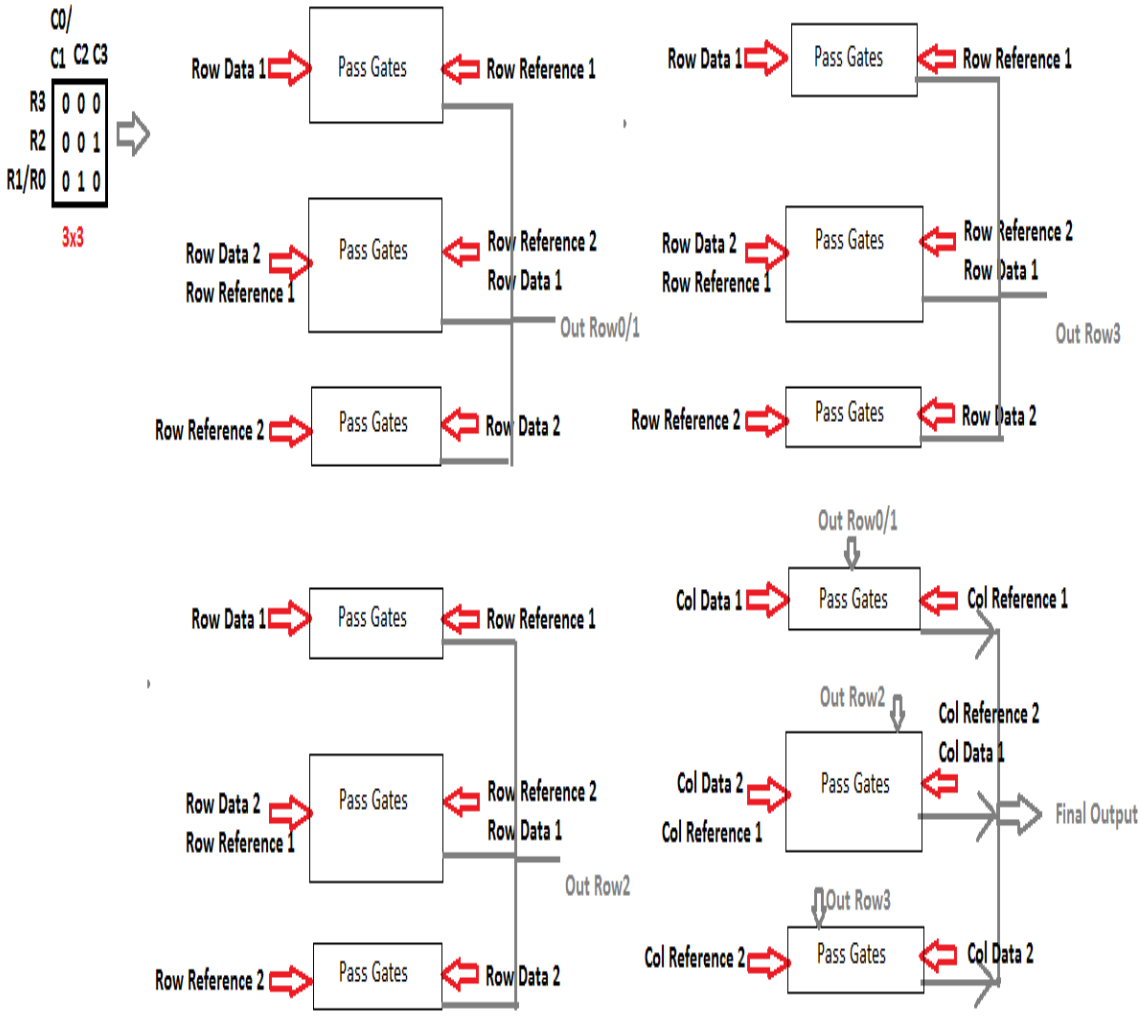

Figure 7.7. Block Diagram of 3x3 Truncated Array

As the second set of examples of how the array can be truncated, Figure 7.8 is considered. The array has the data similar for all the columns, row 0 and row 1 have the same data, and row 2 and row 3 have the same data. Hence, the final truncated array is a 2x1 array with two rows and one column.

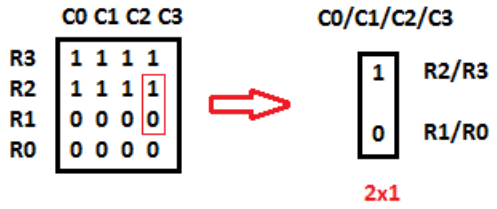

Figure 7.8. Example Array that is truncated to 2x1 Array

For the array to pass row 0 or row 1, the pass gates will have data 1 and reference 1 as the control lines, with data 1 being low and reference 1 being high. For the array to pass through row 2 or row 3, the pass gates will have data 1 and reference 1 as the control lines, with data 1 being high and reference 1 being low. The block diagram of the 2x1 array is shown in Figure 7.9 below.

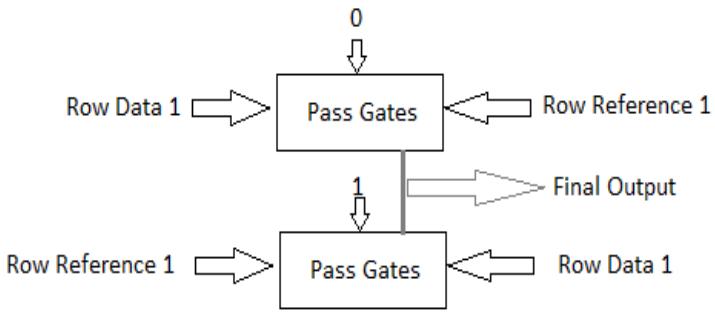

Figure 7.9. Block Diagram of 2x1 Truncated Array

7.5 SCHEMATICS AND SIMULATIONS

The truncated arrays have been designed using the CMOS technology and are simulated using HSPICE. Figure 7.10 shows the schematic of a 3x3 truncated array corresponding to the block in Figure 7.7. The driver outputs the reference and data signals and these in turn are used to control the pass gates. The row driver has three pairs of data and reference lines and the column driver has another three pairs of data and reference lines. The input values to the cells in the array are programmed using external value structures.

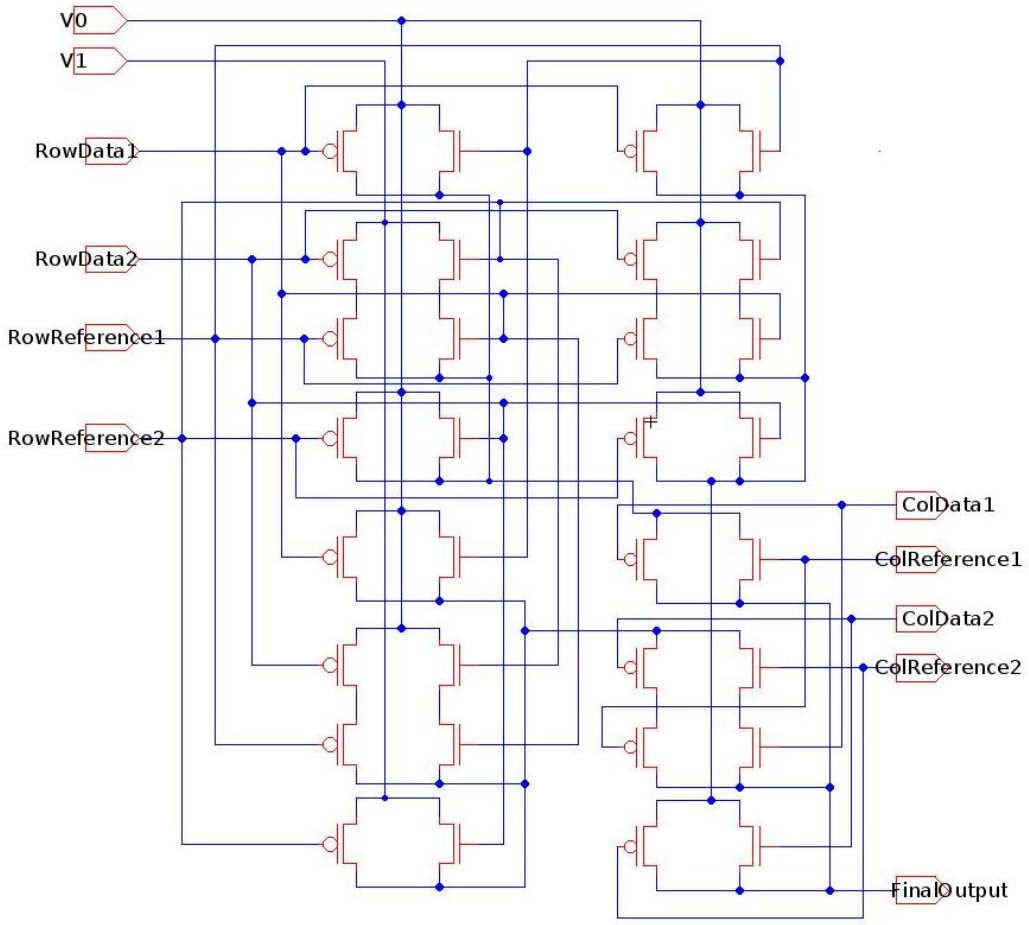

Figure 7.10. Schematic of 3x3 Array

The simulation for the above schematic is shown in Figure 7.11. The drivers are given the inputs so that all the sixteen combinations are observed in the simulation. It can be seen that the results are the same for the column data being a 0 or a 1. Also the results for row 0 and row 1 are the same.

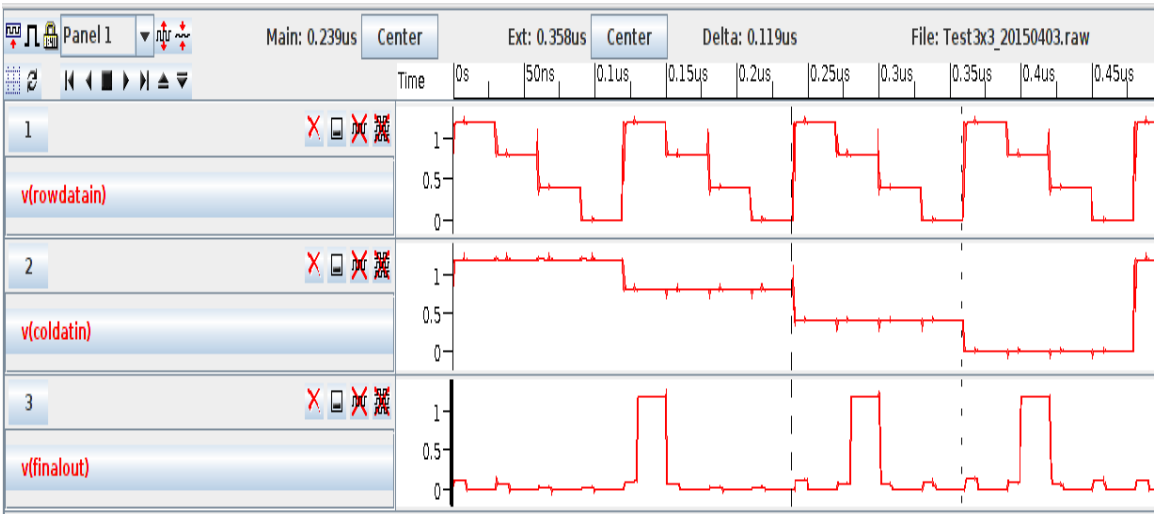

Figure 7.11. Simulation of 3x3 array

Figure 7.12 shows the schematic of a 2x1 truncated array corresponding to the block in Figure 7.9. Since all the columns are exactly the same, the data and reference signals from the column driver do not control the pass gates. The row driver is the only component that affects the selection of the pass gates. Figure 7.13 shows the simulation results for the 2x1 array. Appendix E shows the layouts of the truncated arrays along with complete ALU control circuitry.

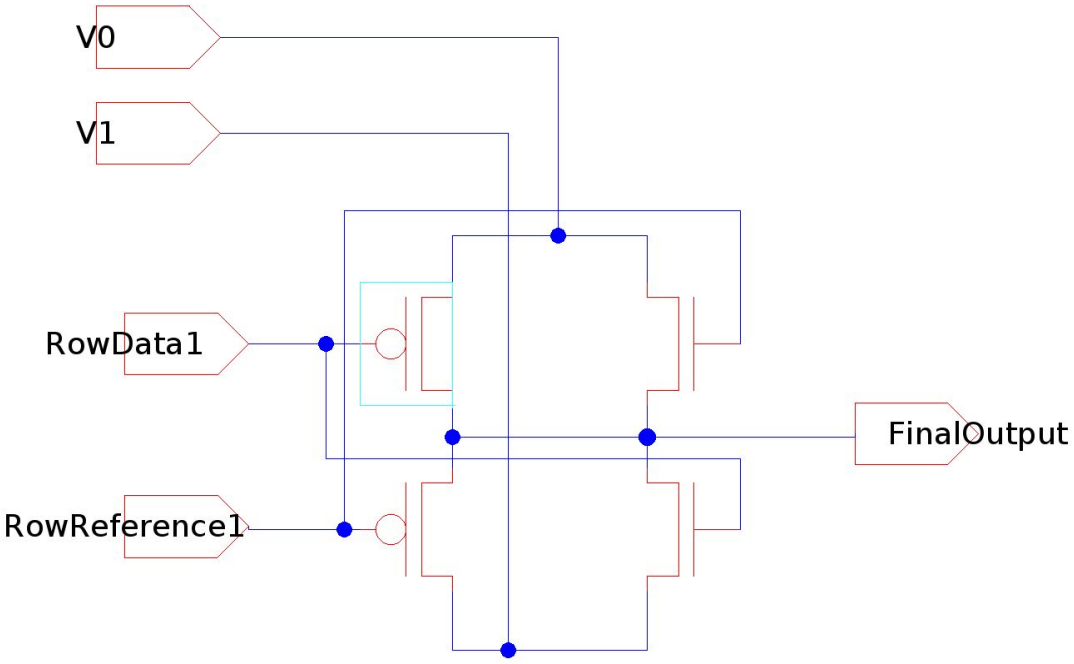

Figure 7.12. Schematic of 2x1 Array

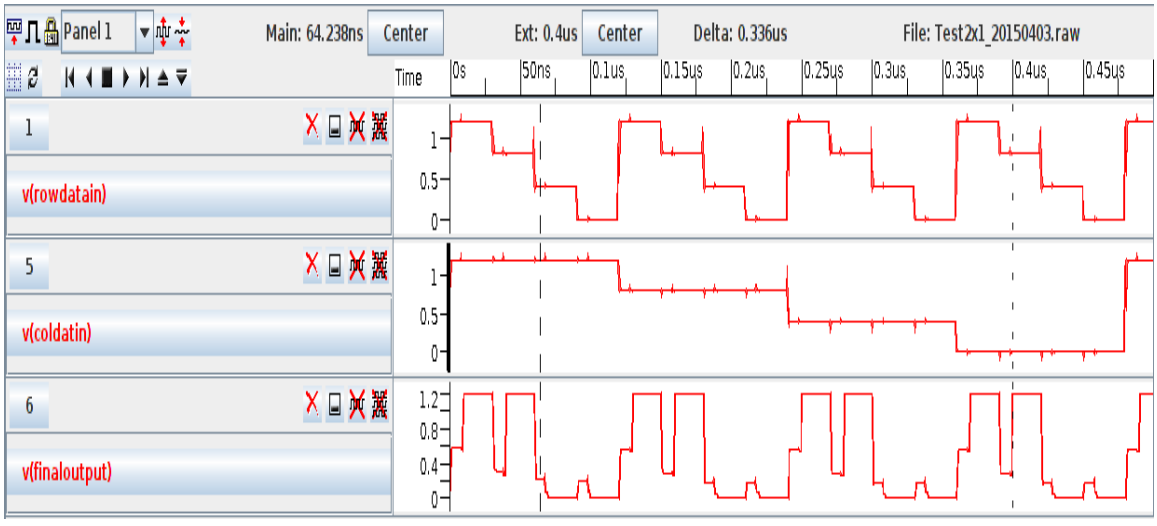

Figure 7.13. Simulation of 2x1 Array

7.6 RESULTS AND DISCUSSION

The ALU control unit is designed using the truncated arrays, and the outputs from these arrays are used to control the components in the ALU. The local opcode is determined, and the operations are performed based on the local opcode. Since the controller is designed based on the ARM Cortex M0 processor, the high level control unit that takes in the 16-bit THUMB instruction is referred to as legacy controller and the ALU control unit is referred to as native controller. The block diagram of the ALU control circuitry and how it controls the ALU is shown in the Figure 7.14.

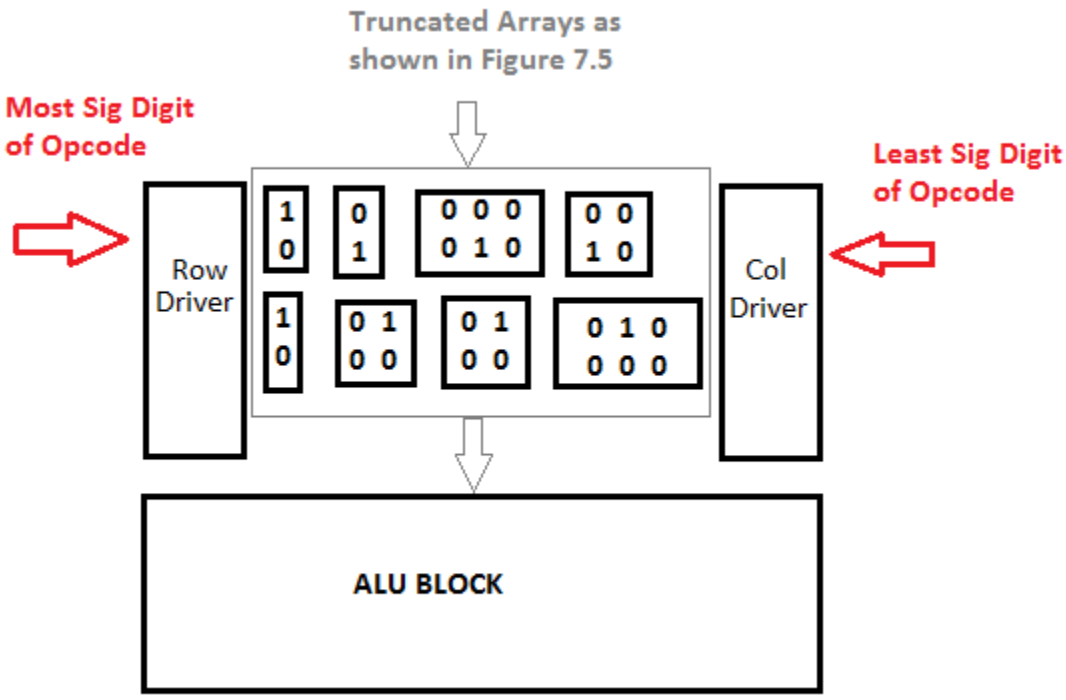

Figure 7.14. ALU Control Circuitry with the ALU

8. SUMMARY

Logic circuits including an adder and an arithmetic and logic unit have been implemented and simulated using CorMem digital reasoning technology. The signals are quaternary and the processing is done through matrix algebra operations and memory manipulation. This combination has important advantages in interconnection complexity, chip area requirements, and power consumption as compared to binary circuits with comparable functionality. This dissertation work has demonstrated the functionality of the approach and the significant capability for hardware reuse. In particular, it provides important insight into the degree that chip area requirements can be minimized with the approach.

The CorMem multi-level, memory-based approach is based on a set of modular CMOS structures that are used in a nontraditional way. Flexible, modular systems are hence possible using standard fabrication methods. While basic modular structures are more complex than the basic logic gates in binary circuitry, the overall system circuitry becomes relatively less complex for higher order circuits. Basic architectures are drivers and arrays. The drivers provide control voltages from multi-level input signals and the arrays store multi-level output voltages through internal reference voltage connections. The output voltage for a simple circuit is the value stored in the selected array cell. The basic CorMem architectures are used for circuitry to perform the various processing functions as well as binary-to-quaternary and quaternary-to-binary conversion. Hence, hybrid circuits, i.e. mutli-level and binary, are easily implemented.

Quaternary or four-level architectures were investigated using the digital reasoning approach. The quaternary case was chosen as a proof of concept and for its simplicity when compared to higher-order bases. The quaternary development facilitated the pattern recognition that was needed to optimize the arrays for efficient hardware reuse. The optimization was accomplished through inspection and iteration. However, the approach is not limited to quaternary architectures; octal, decimal, base 12, or hexadecimal architectures can be implemented. For example, the array structure for quaternary has sixteen cells, i.e. four by four.

The array structure for higher-order bases will have more cells and will require additional reference voltages and finer signal discrimination. There will be a need of an automated, software process for the pattern recognition to get optimized arrays and optimized area for these higher order bases. The extension of the technology to these other bases is an area for future research.

In the fourth chapter, the equivalent of logic gates has been implemented as an example of the approach. The digital reasoning circuit for higher-order functions would not necessarily be limited to a logic-gate implementation. Later, a full adder circuit has been implemented using the CMOS 130-nm technology and simulated using HSPICE. The area of an 8-bit binary carry save adder (CMOS 130-nm technology) is about 6210 μm^2 whereas the area for the 8-bit binary ripple carry adder is 2214 μm^2, whereas using the digital reasoning the four-digit or eight-bit adder is 3450.2 μm^2. The dissertation also demonstrates the arithmetic and logic unit with hardware reuse being effectively introduced through matrix algebra concepts. The layout is developed in IBM 130-nm technology and the number of metal layers used is limited to up to three. The area of the quaternary ALU is 1076 μm^2. The binary version of the ALU that performs the same functions has an area of 7800 μm^2. The seventh chapter demonstrates the controller block which takes in the local opcode and sends the control signals to the ALU to allow it to perform the functions that it ought to. Truncated arrays are used to design the controller block and they illustrate how the hardware can be reused and how the area can be minimized through pattern recognition.

The CorMem digital reasoning approach has advantages over the traditional binary logic. Applications can include multi-level-only circuits or hybrid circuits with both multi-level and binary components. The future work includes creating an entire 16-digit processor (comparable to 32-bit binary devices) based on the digital reasoning approach. This processor includes the ALU along with the register banks, the control units, interface circuitry, memory banks and beyond. Also, ongoing work includes formalizing the pattern recognition methodology and seeking added hardware re-use which may reduce further the area of the circuits.

APPENDIX A

LAYOUT OF ARRAY AND SELECTOR

The layout of the array and selector is built using IBM 130 nm library. The metal layers from 1 through 3 have been used in building these components. The array has four pairs of input lines, and only one row is activated at a given point of time. The cells are stored with values which are programmed using value structures. There are four outputs coming out of the array. The selector has four inputs coming from the outputs of the array and four pairs of input lines activating only one cell and hence giving one final output. Figure A.1 shows the layout of the array and A.2 shows the layout of the selector.

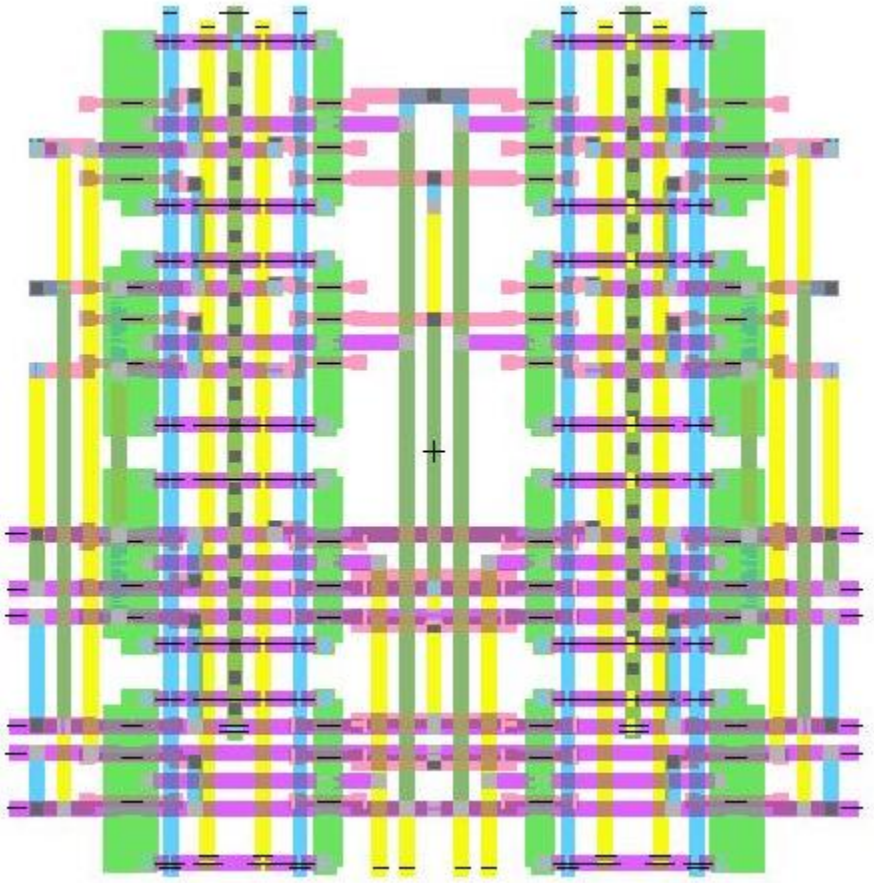

Figure A.1. Layout of the Array

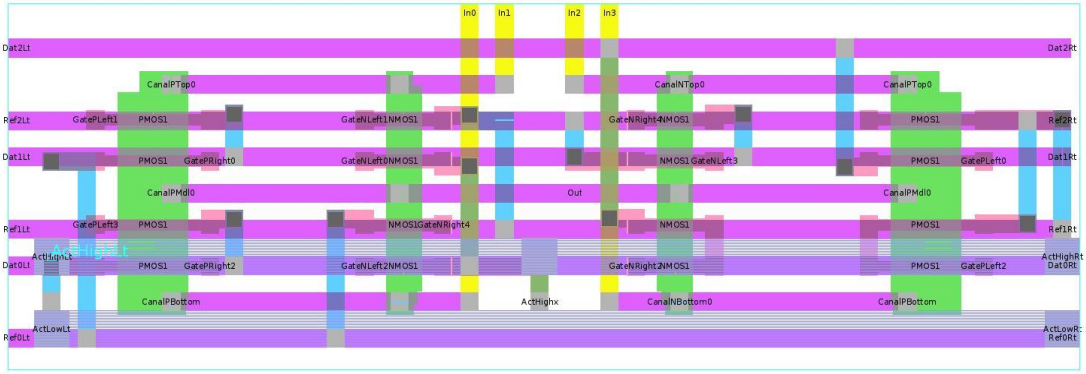

Figure A.2. Layout of the Selector

APPENDIX B

LAYOUT OF DRIVER AND SIX-TO-EIGHT DECODER

The driver takes the quaternary input and outputs the control voltages to the array. The driver is built using three sense amplifiers and the layout is done using the IBM 130 nm library. The driver only uses two metal layers. The six output lines from the driver act as inputs to the six-to-eight decoder. The eight output lines from the decoder are used in activating one of the rows of the array. Figure B.1 shows the layout of the driver and Figure B.2 shows the layout of the six-to-eight decoder.

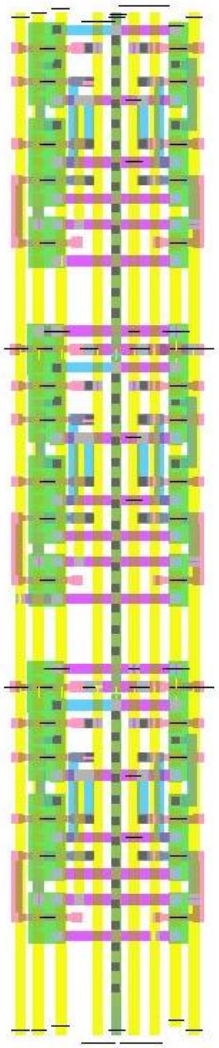

Figure B.1. Layout of Driver

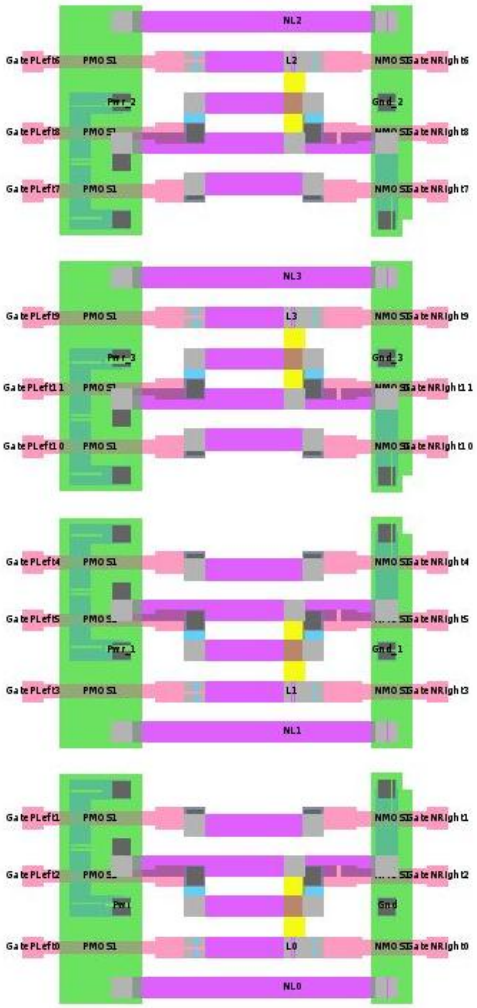

Figure B.2. Layout of six-to-eight decoder

APPENDIX C

LAYOUT OF THE CONVERSION CIRCUITS

Binary-to-quaternary and quaternary-to-binary circuits are used as interface units to do the conversion from base 2 to base 4 and vice versa. The circuits are laid out using CMOS technology and the IBM 130 nm library. The quaternary-to-binary circuit is shown along with the driver connected. Figure C.1 shows the binary-to-quaternary layout and Figure C.2 shows the quaternary-to-binary layout.

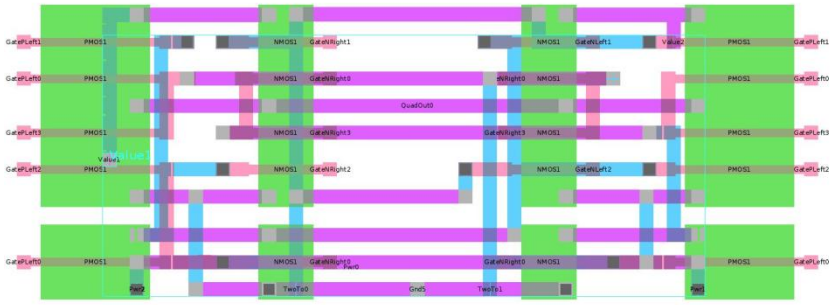

Figure C.1. Layout of binary-to-quaternary

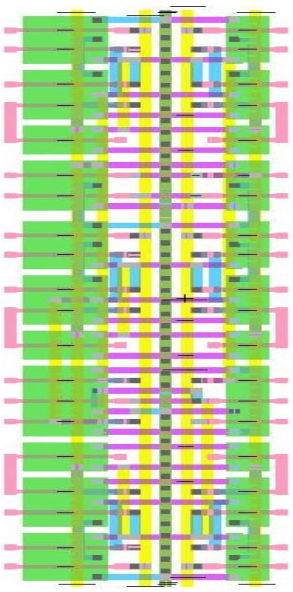

Figure C.2. Layout of quaternary-to-binary

APPENDIX D

LAYOUT OF THE MANGLER AND ALU BLOCK

Mangler is a circuit that is used to shuffle the elements in the matrix. By introducing a mangler, the same array can be used for more than one operation. A row mangler 3210 is shown below and the layout is done using the IBM 130 nm technology. The entire ALU block which contains the two drivers, the three arrays with the manglers is also laid out and even at this higher level, only three metal layers have been used which shows the effectiveness of the approach with the interconnects. Layout of the row mangler 3210 is shown in Figure D.1 and the entire ALU Block is shown in Figure D.3.

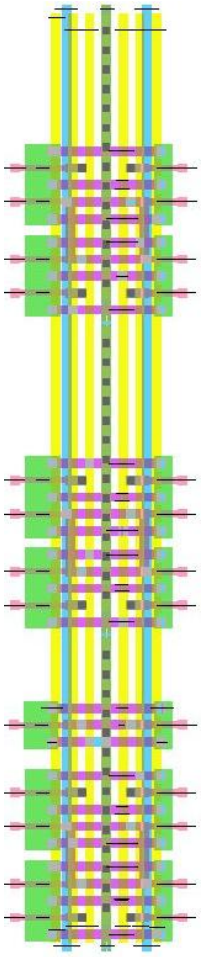

Figure D.1. Layout of Row Mangle 3210

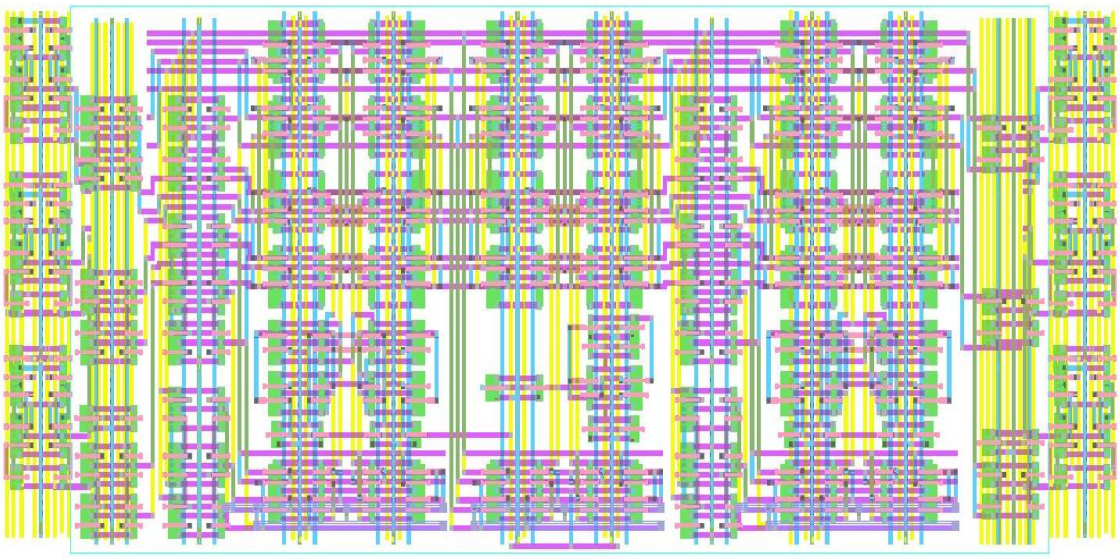

Figure D.2. Layout of ALU Block

APPENDIX E

LAYOUT OF TRUNCATED ARRAYS AND ALU CONTROLLER BLOCK

Figure E.1 shows the layout of the truncated array 3x3 and Figure E.2 shows the truncated array 2x1. The entire ALU controller block is laid out using IBM 130 nm technology and is shown in Figure E.3.

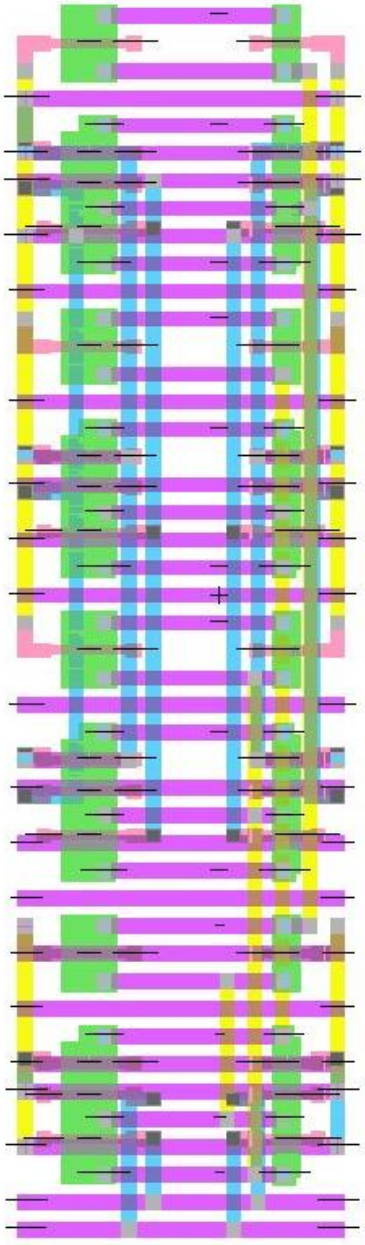

Figure E.1. Layout of 3x3 Array

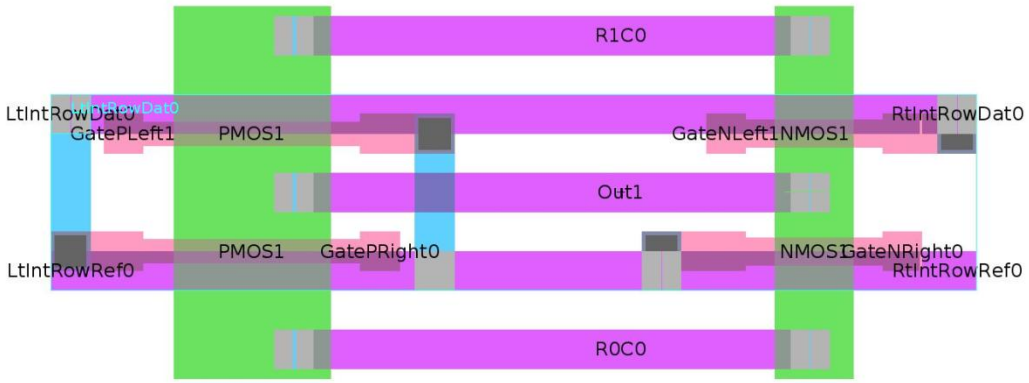

Figure E.2. Layout of 2x1 Array

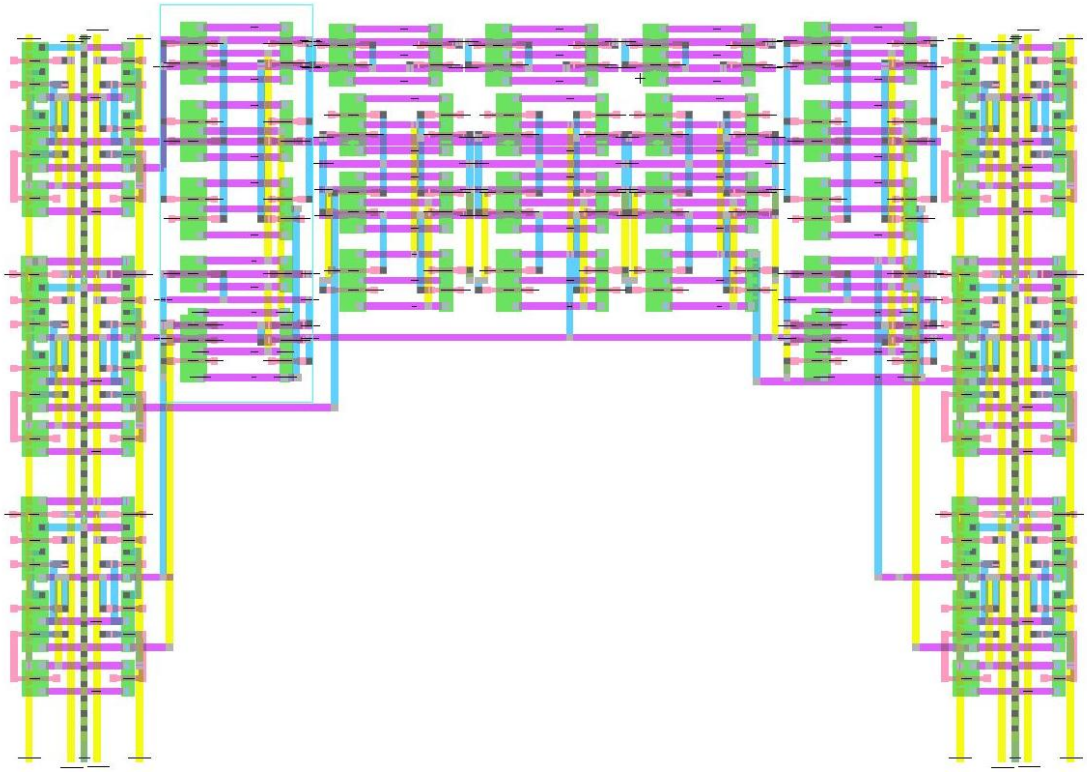

Figure E.3. Layout of ALU Controller Block

APPENDIX F

HARDWARE DEMONSTRATION

Multiple prototype test chips were fabricated in silicon using IBM 130-nm 8RF-DM CMOS technology (MOSIS process with no options selected). The chips included the AND circuitry. A prototype chip is shown in Figure F.1. The operational voltage, i.e. CMOS VDD, is 1.2 V. Each truth table case was reproduced for the AND operation. The quaternary values varied as shown below for each signal level (automotive noise levels and thermal testing 25 °C to 80 °C):

- Data Value 3: 1.2 V + 75 mV
- Data Value 2: 0.8 V + 125 mV
- Data Value 1: 0.4 V + 100 mV
- Data Value 0: 0.0 V + 175 mV

A typical experimental result with noise is shown in Figure F.2. The middle blue and purple curves are driver outputs. The bottom green curve is the input trigger. (The upper yellow trace is not connected.)

Figure F.1. Prototype Chip

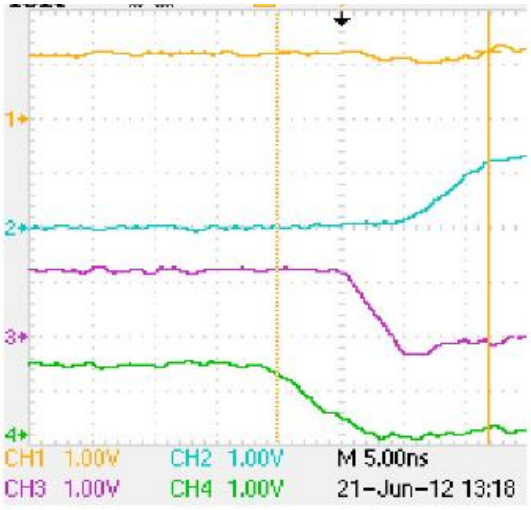

Figure F.2. Experimental Driver Outputs

REFERENCES

1. R. R. Schaller, "Moore's Law: Past, Present and Future," IEEE Spectrum, 34(6), 52-59 (1997).

2. C. A. Mack, "Fifty Years of Moore's Law," IEEE Transactions on Semiconductor Manufacturing, 24(2), 202-207 (2011).

3. R. K. Cavin, "Science and Engineering: Beyond Moore's Law," Proceedings of the IEEE, 100 (Special Centennial), 1720-1749 (2012).

4. Kenneth C. Smith, "Multiple-Valued Logic: A Tutorial and Appreciation," Computer, 21(4), 17-27 (1988).

5. Kenneth C. Smith, "The Prospects for Multi-valued Logic: A Technology and Applications View," IEEE Transactions on Computers, 30(9), 619-634 (1981).

6. Stanley L. Hurst, "Two Decades of Multiple-valued Logic-An Invited Tutorial," 18th International Symposium on Multiple-Valued Logic, Palma de Mallorca, Spain, 24-26 May 1988.

7. Jan Lukasiewicz, "Selected Works," ed. L. Borkowski and Translated O. Wojtasiewicz (North-Holland Publishing Co., Amsterdam, 1970).

8. S. Feferman, "Tarski's Influence on Computer Science," 20th Annual IEEE Symposium on Logic in Computer Science, Chicago, Illinois, 26-29 June 2005.

9. Lofti A. Zadeh, "Fuzzy Sets, Fuzzy Logic, and Fuzzy Systems, Selected Papers by Lofti A. Zadeh," ed. G. J. Klit and B. Yuan, Advances in Fuzzy Systems –Applications and Theory Vol. 6 (World Scientific Publishing Co., Singapore,1996).

10. Santanu Mahapatra and Adrian Mihai Ionescu, "Realization of Multiple Valued Logic and Memory by Hybrid SETMOS Architecture," IEEE Transaction on Nanotechnology, 4(6), 705-714 (2005).

11. Richardo Cunha G. da Silva and Henri Boudinov, and Luigi Carro, "A Novel Voltage-Mode CMOS Quaternary Logic Design," IEEE Transaction on Electron Devices, 53(6), 1480-1483 (2006).

12. A. Herffeld and S. Hentschke, "Quaternary dynamic differential logic with application to fuzzy-logic circuits," 27th IEEE International Symposium on Multiple-Valued Logic, Antigonish, Nova Scotia, Canada 28-30 May 1997.

13. A. N. Gupte and A. K. Geol, "Study of Quaternary Logic Versus Binary Logic," First Great Lakes Symposium on VLSI, 1991, Kalamazoo, Michigan, 1-2 March 1991, 336-337.

14. Mahsa Dornajafi, Steve E. Watkins, Benjamin Cooper, and M. Ryan Bales, "Performance of a Quaternary Logic Design," IEEE Region 5 Technical Conference, Kansas City, Missouri, 18-19 April 2008.

15. Patel K. S. Vasundara and K. S. Gurumurthy, “Design of High Performance Quaternary Adders,” 41st IEEE International Symposium on Multiple-Valued Logic, 23-25 May 2011, Tuusula, Finland.

16. Augustine W. Chang, “SCL Type FPGA with Multi-Threshold Transistors and Method for Forming Same,” U.S. Patent No. 7,375,548 [Issued 20 May 2008].

17. C. Lazzari, P. Flores, J. Monteiro, and L. Carro, "Voltage-mode Quaternary FPGAs: An Evaluation of Interconnections," Proceedings of 2010 IEEE International Symposium on Circuits and Systems (ISCAS), 869-872, 30 May-2 June 2010.

18. R. Rani, L. K. Singh, and N. Sharma, "FPGA Implementation of Fast Adders using Quaternary Signed Digit Number System," International Conference on Emerging Trends in Electronic and Photonic Devices & Systems, 2009. ELECTRO '09, 132-135, 22-24 Dec. 2009.

19. A. N. Nagamani and S. Nishchai, "Quaternary High Performance Arithmetic Logic Unit Design," 14th Euromicro Conference on Digital System Design (DSD), 148-153, 31 Aug.-2 Sept. 2011.

20. P. M. Kelly, T. M. McGinnity, L. P. Maguire, and L. McDaid, "Exploiting Binary Functionality in Quaternary Look-up Tables for Increased Functional Density in Multiple-valued Logic FPGAs," Electronics Letters , 41(6), 300-302, (2005).

21. Benjamin J. Cooper, “Device and Method for Enabling Multi-valued Digital Computation and Control,” U.S. Patent No. 8,513,975 [Issued 20 August 2013].

22. Benjamin J. Cooper, “Device and Method for Enabling Multi-valued Digital Computation,” U.S. Patent No. 8,593,875 [Issued 26 November 2013].

23. R.UMA, Vidya Vijayan, M. Mohanapriya, Sharon Paul, “Area, Delay and Power Comparison of Adder Topologies,” International Journal of VLSI design & Communication Systems (VLSICS), 3(1), February 2012.

24. Biggerstaff, T.J., "Moore's Law: Change or Die," Software, IEEE , 13(1), 4, Jan. 1996.

25. Lazzari, C.; Flores, P.; Monteiro, J.C., "Power and delay comparison of binary and quaternary arithmetic circuits," Signals, Circuits and Systems (SCS), 2009 3rd International Conference, 1-6, 6-8 Nov. 2009.

26. Aitken, A.; MacArthur, A.T.P.; Abbott, R.; Morris, J.D., "The relative performance and merits of CMOS technologies," Electron Devices Meeting, 1976 International, 22, 327-330, 1976.

27. V. Aschoff, “The early history of the binary code,” IEEE Communications Magazine, 21, 4-10, Jan. 1983.

28. Davis, W., "Compatible analog/digital techniques for a monolithic process," Solid-State Circuits Conference. Digest of Technical Papers. 1975 IEEE International, 18, 76, Feb 1975.

29. Dingwall, A.G.F.; Stricker, R.E., "C^2L: A new high speed, high density bulk CMOS technology," Electron Devices Meeting, 1976 International, 22, 188-191, 1976.

30. Vranesic, Z.G.; Smith, K.C., "Engineering aspects of multi-valued logic systems," Computer, 7(9), 34-41, Sept. 1974.

31. Allen, C.M.; Givone, Donald D., "A Minimization Technique for Multiple-Valued Logic Systems," Computers, IEEE Transactions, C-17(2), 182-184, Feb. 1968.

32. Halpern, Israel; Yoeli, Michael, "Ternary arithmetic unit," Proceedings of the Institution of Electrical Engineers, 115(10), 1385-1388, October 1968.

33. Hampel, D., "MultiFunction Threshold Gates," Computers, IEEE Transactions on, vol.C-22(2), 197-203, Feb. 1973.

34. Porat, D.I., "Three-valued digital systems," Proceedings of the Institution of Electrical Engineers, 116(6), 947-954, June 1969.

35. Thoidis, I.M.; Soudris, D.; Fernandez, J.M.; Thanailakis, A., "The circuit design of multiple-valued logic voltage-mode adders," Circuits and Systems, 2001. ISCAS 2001. The 2001 IEEE International Symposium on, 4(4), 162-165, 6-9 May 2001.

36. Neha Umredkar, Prof.M.A.Gaikwad, Prof.D.R.Dandekar, "Design of Low Power Quaternary Adders in Voltage Mode Multi-Valued Logic," IOSR Journal of VLSI and Signal Processing (IOSR-JVSP), 3(1), 15-21, Sep.–Oct. 2013.

37. Kabat, W.C.; Wojcik, A.S., "On the Design of 4-Valued Digital Systems," IEEE Transactions on Computers, C-30(9), 666-671, Sept. 1981.

38. Current, K.W., "A CMOS quaternary latch," Proceedings, Nineteenth International Symposium on Multiple-Valued Logic, 54-57, 29-31 May 1989.

39. Wayne current, K., "A CMOS quaternary threshold logic full adder circuit with transparent latch," Proceedings of the Twentieth International Symposium on Multiple-Valued Logic, 168-173, 23-25 May 1990.

40. Hurst, S.L., "Multiple-Valued Logic— its Status and its Future," IEEE Transactions on Computers, C-33(12), 1160-1179, Dec. 1984.

41. Tirumalai, P.P.; Butler, J.T., "Minimization algorithms for multiple-valued programmable logic arrays," IEEE Transactions on Computers, 40(2), 167-177, Feb 1991.

42. C. A. Mack, "The end of the semiconductor industry as we know it," Proc. SPIE Optical Microlithography XVI (Plenary Address), 5040, 21–31, 2003.

43. G. E. Moore, "VLSI: Some fundamental challenges," IEEE Spectrum, 16(4), 30–37, Apr. 1979.

44. E. Dubrova. "Multiple-valued logic in vlsi: Challenges and opportunities," Proceedings of NORCHIP, 340–350, 1999.

45. Gulak, G., "A review of multiple-valued memory technology," Proceedings. 1998 28th IEEE International Symposium on Multiple-Valued Logic, 222-231, 27-29 May 1998.

46. Burks, Arthur W., "Electronic Computing Circuits of the ENIAC," Proceedings of the IRE, 35(8), 756-767, Aug. 1947.

47. N. Winder, “The Historian’s Dilemma, or Jonah and the Flatworm”, Society for Human Ecology, 6(2), 1999.

48. D. R. Lindbergh, M. T. Ghiselin, “Fact, Theory and Tradition in the Study of Molluscan Origins”, Proceedings of the California Academy of Sciences, 54(27), 663-686, Nov. 2003.

VITA

Indira Priyadarshini Dugganapally was born in Kadapa, India on May 30, 1986. She completed her Bachelor of Technology in Electronics and Communication Engineering at Jawaharlal Nehru Technological University, India in May 2007. In August 2007, Indira joined the Department of Electrical and Computer Engineering at Missouri University of Science and Technology to pursue a masters' degree. She received her Master of Science degree in 2009.

VLSI has been her field of interest and she worked for a California based startup company, Core Memory Circuits, LLC for over a year before she came back to Missouri University of Science and Technology to pursue a PhD under the guidance of Dr.Watkins. She received her Doctor of Philosophy degree in Computer Engineering in May 2015. During her graduate study, as a part of research, she continued working with Core Memory Circuits and has played a major role in developing a processor based on a new generation technology.