# DESIGN AND VALIDATION OF QUATERNARY ARITHMETIC CIRCUITS

**Approved by:**

Dr. Mitchell Thornton (Chair & Dissertation Director)

Dr. Sukumaran Nair

Dr. David Matula

Dr. Ping Gui

Dr. Stephen Szygenda

# DESIGN AND VALIDATION OF QUATERNARY ARITHMETIC CIRCUITS

A Dissertation Presented to the Graduate Faculty of

School of Engineering

Southern Methodist University

in

Partial Fulfillment of the Requirements

for the degree of

Doctor of Philosophy

with a

Major in Computer Engineering

by

Satyendra Ravi Prasad Raju Datla

(B.S., IETE, New Delhi)
(M.S. Computer Engineering, SMU, Dallas)

December 19th 2009

## ACKNOWLEDGEMENTS

I want to take the opportunity to thank all the people who supported me in this endeavor. In particular, I owe a great deal of gratitude to my advisor, Professor Mitchell Thornton, who encouraged me all throughout the Ph.D. program. This dissertation would not have been possible without the expert guidance of Dr. Thornton. I have known Dr. Thornton for around 7 years and I learned many good things in life from my association with him and I have tremendous respect for him. I also sincerely thank Prof. Sukumaran Nair for motivation and guidance he provided me during last 6-7 years of my association with him. I also worked closely with Prof. David Matula for last few years on arithmetic problems and I thank Dr. Matula for his support and guidance.

My special thanks also go to my committee: Prof. Sukumaran Nair, Prof. David Matula, Prof. Ping Gui and Prof. Stephen Szygenda for readily agreeing to be on the dissertation committee. I also thank them for providing very useful suggestions during the project work. I would like to offer special thanks to Synopsys for providing simulation tools at SMU which helped me during the project. I hereby thank all the friends and staff in CSE, EE, Research Departments and my colleagues at Texas Instruments for helping me during the project and for sharing their knowledge and experiences. I also thank Luther Hendrix and Dave Henderson for the discussions during the project.

I thank my parents, Ramasita and Trinatha Raju, and grandparents, especially my grandmother, Subbayamma, for providing an excellent and vibrant childhood, which I cherish my entire life. They also taught me the importance of the education and social consciousness. I thank my elder brother, Krishnam Raju, and all my cousins for encouraging me all throughout my education.

I also thank my wife, Sumitra, for her unwavering support during the many long days that went into this endeavor. It took pretty good understanding and support from her, to enable me to undertake the part-time Ph.D. program in addition to my full time working position. I would like to tell our little 2.5 years old son, Sanjith, that he brought great deal of joy and happiness to our family. I am blessed to have such a lovely family. I am also grateful to all of my other family members and so many well wishers whose love and care upon me always encouraged me. And last, but not least, I would like to thank God Shriram for everything in my life.

Datla, Satyendra R. P. Raju B.S., IETE, New Delhi, 1999
M.S.Comp.Engg, SMU, Dallas, 2004

Design and Validation of Quaternary Arithmetic Circuits

Advisor: Professor Mitch Thornton

Doctor of Philosophy conferred December 19th, 2009

Dissertation completed December 4th, 2009

Arithmetic circuits play a very critical role in both general-purpose and application specific computational circuits. Multiple Valued Logic (MVL) provides the key benefit of a higher density per integrated circuit area compared to traditional two-valued binary logic. Quaternary (Four-valued) logic also offers the benefit of easy interfacing to binary logic because radix 4 (=$2^2$) allows for the use of simple encoding/decoding circuits. The functional completeness is proven with a set of fundamental quaternary cells. The library of cells based on the Supplementary Symmetrical Logic Circuit Structure (SUSLOC) are designed, simulated, and used to build several quaternary fixed-point arithmetic circuits such as adders, multipliers. These SUSLOC circuit cells are validated using SPICE models and the arithmetic architectures are validated using System Verilog models for functional correctness. Quaternary (radix-4) dual operand encoding principles are applied to optimize power and performance of squaring circuits using standard CMOS gates in 130nm and 90nm technologies.

This dissertation summarizes the study conducted to research the MVL circuits and feasibility of design and validation of quaternary arithmetic circuits using SUSLOC technology and also quaternary dual recoding squaring circuits using CMOS gates. The research indicates that the quaternary circuits do offer the benefit of lower power consumption compared to the traditional two-valued (binary) logic circuits.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

# LIST OF TABLES

Dedicated to my family, who has been source of unwavering support, and also to

Swami Vivekananda, whose inspirational messages always motivated me

# Chapter 1

# INTRODUCTION

Integrated Circuit (IC) design has been evolving from ages of “Small scale integration” (SSI), which involves tens of transistors, to the present days of “Very Large Scale Integration” (VLSI), that involves millions of transistors on a single-chip. Several different circuit technologies have been used in this journey of IC design evolvement. Metal Oxide Semiconductor (MOS) technology offered several key benefits along with an easy fabrication of basic transistor switch and allowed the scaling (sizing down the size of transistor) resulting in IC size reduction periodically. Complementary Metal Oxide Semiconductor (CMOS) technology brought the low power consumption advantage which revolutionized semiconductor design applications. While Analog designs heavily rely on continuous signal response of transistors, digital designs rely on discrete logic levels of the signals. The introduction microprocessor designs along with their surrounding applications significantly increased the realization of complex real world applications using digital ICs for the past couple of decades. Two-level logic namely “binary logic” is the decoding method used heavily. Binary logic essentially involves only two logic levels: 0 and 1. Another advance in the logic decoding is the use of multiple levels namely more than two discrete levels representing the signals.

Multiple-Valued Logic (MVL) circuits offer several potential opportunities for improvement of present VLSI circuit designs [D99].

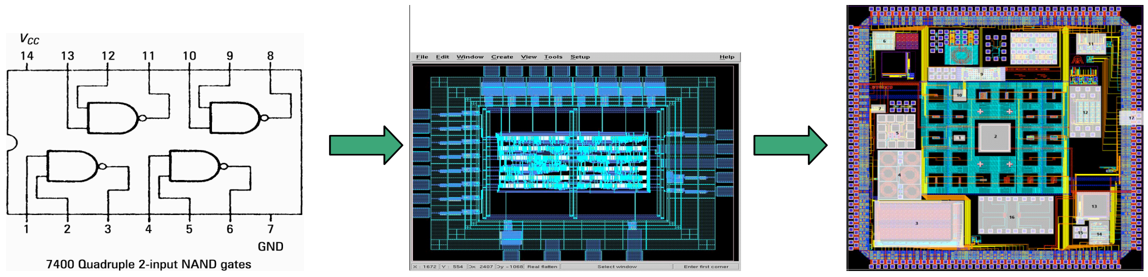

$V_{CC}$
14 13 12 11 10 9 8
1 2 3 4 5 6 7
GND
7400 Quadruple 2-input NAND gates
File Edit Window Create View Tools Setup Help

**Figure 1: LSI to MSI to VLSI migration**

Fixed point adder and multiplier circuits are fundamental building blocks of practically every algorithm ranging from simple arithmetic to graphics and signal processing applications. Increased data density, reduced dynamic power dissipation, and increased computational ability are among some of the key benefits of Multiple-Valued Logic (MVL) [MT08, H84, K90].

## 1.1. MVL

As elaborated in [MT08], Multiple-Valued Logic (MVL) is a discipline of discrete $p$-valued systems where $p>2$, or in other words, non-binary valued systems. In general sense, both binary-valued and discrete-valued variables with an infinite number of values can be considered as MVL systems. Hence forth in this report MVL shall be referred to as the system to utilize variables that can take on a discrete set of values with cardinality of three or more. MVL principles and methods are general and independent from the actual underlying implementation of the circuits. MVL has several applications in the modern IC design methods and Electronic Design Automation (EDA) tools. MVL has long history of use in EDA-CAD tools and Hardware Description Languages (HDL) for digital circuit simulation and synthesis. While several MVL applications are detailed in [MT08], specifically, MVL circuit design applications are focused in the context of this report. Figure 2 illustrates the example logic levels for binary (radix-2), ternary (radix-3) and quaternary (radix-4) valued circuits.

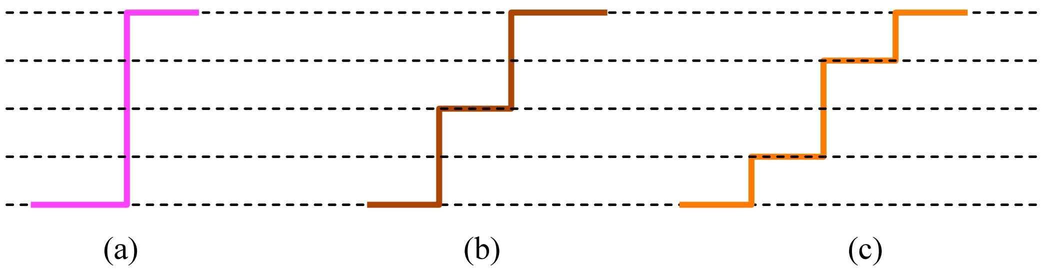


**Figure 2: a) Two valued b) Three valued and c) Four valued**

Higher radix allows representing more functions compared to smaller radix. The generic formula for number of "*n*" variable functions that can be represented using radix "*r*" is $r^{r^n}$. Figure 3 depicts the number of 1-variable (*n*=1) functions possible for each radix starting from two. Increased number of functions is what makes the MVL circuits offer higher logic density compared to binary circuits.

| **r** | **$r^r$** |
|---|---:|
| 2 | 4 |
| 3 | 27 |
| 4 | 256 |
| 5 | 3125 |
| 6 | 46656 |
| 7 | 823543 |
| 8 | 16,777,216 |

**Figure 3: No. of 1-Variable Functions of Radix r**

## 1.2. SUSLOC Technology

Several implementation methods have been proposed in the recent past to realize the MVL circuits. The MVL circuits can fundamentally be categorized as: Current-mode, Voltage-mode and Mixed-mode circuits. Current-mode circuits [B91,CC95] have been popular and offer several benefits. But, their power consumption is high when compared to Voltage-mode circuits due to their inherent nature of constant current flow during the functional operation.

Alternatively, Voltage-mode circuits consume a large majority of power only during the logic level switching plus any additional leakage currents that may be present. Hence, Voltage-mode circuits do offer lesser power consumption which has been the key benefit of traditional CMOS binary logic circuits from the perspective of dynamic switching activity. Lower power consumption has been the key benefit of traditional CMOS binary logic circuits for several technology nodes. Because of increased proliferation of portable battery powered personal computation devices, reduced power dissipation is an important design constraint and motivates us to explore Voltage-mode multiple-valued circuits. As indicated before, MVL technology primarily refers to circuit technologies where more than two logic levels are used to represent signal levels design larger circuits. There were several MVL architectures proposed and circuits designed using the ternary (three levels), quaternary (four levels) penternary (five levels). Several approaches for MVL circuit design have been proposed [CC95, B91, C+02, EI88, KK88, S88, I98]. Recently, a self-sustaining and consistent circuit architecture called the Supplementary Symmetrical Logic Circuit (referred as SUSLOC) structure is proposed and patented [O00, OC00]. This proposed circuit architecture also allows the use of readily available circuit elements to construct logic circuits based on any radix number system. Previous work has utilized SUSLOC technology for ternary systems [OC00, AS+03, OC01].

As described in [O00, DT+09], three requirements must be met by the circuit structure to design and fabricate quaternary MVL circuits using SUSLOC:

1) there must be three different sources of power available, with each source of power representing one of three different logic levels with the ground plane representing the fourth level
2) there must be one controllable path, or branch, from a source of power to an output terminal of the circuit per output logic level and
3) only one controllable path, or branch, conducts from a source of power to an output terminal per input logic level, contiguous group of input logic levels, or a unique combination of input logic levels.

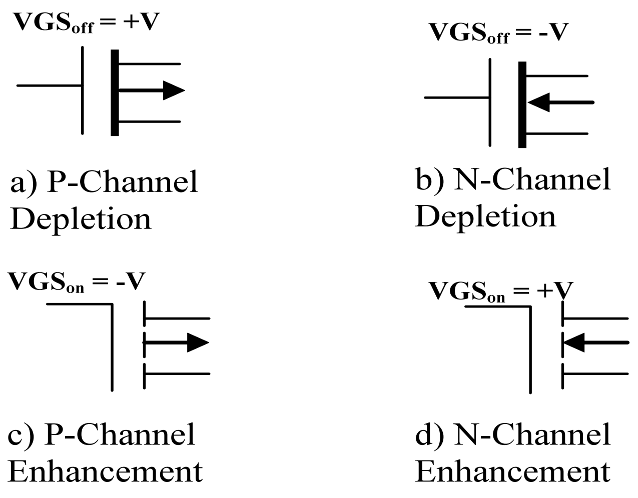


**Figure 4: FETs used for SUSLOC Structure**

Due to their low cost and high reliability, Insulated Gate Field Effect Transistors, (IGFETs, FETs) were chosen in the implementation of ternary logic as described in [OC00, AS+03, OC01]. For similar reasons, the same types of transistors are used for the work described here. Figure 4 depicts the symbols used for the four FETs that are used for the quaternary gate implementations. This need creates the requirement of the ability to vary doping levels to obtain the different $V_T$ values to create different types of transistors. These four types of transistors with different threshold voltage selections can be used in designing SUSLOC circuits based on any radix [O00]

P-channel enhancement mode transistor shown in Figure 4 has a gate threshold voltage, $V_{GS\ on}$ of –V. The term $V_{GS\ on}$ indicates the relative gate threshold voltage at which the P-channel enhancement mode transistor turns ON. The relative voltages are the gate input voltage and the source voltage. If the gate input voltage differs from the source voltage by at least the gate threshold voltage $V_{GSon}$ of –V, the P-channel enhancement mode transistor is ON and the source voltage will be conducted to the drain. Similarly, N-channel enhancement mode transistor has $V_{GS\ on}$ of +V.

Whereas, P-channel depletion mode transistor has a relative gate threshold voltage, $V_{GS\ off}$ of +V. If the gate input voltage differs from the source voltage by at least +V, the P-channel depletion mode transistor is off and no conduction will occur between the source and the drain. Otherwise, the transistor is on and conducts the voltage from source to drain. On the other hand, N-channel depletion mode transistor has got the threshold voltage $V_{GS\ off}$ of -V.

The maximum and minimum power supply voltages (the output voltages) for SUSLOC circuits are determined by the output requirements of the circuit and/or the specifications of the switches being employed. The threshold voltages, $V_{GS(TH)}$, of P-channel FETs selected or fabricated to be a percentage of a logic level above the highest input logic level to which they are to conduct. The threshold voltages, $V_{GS(TH)}$, of N-channel FETs are selected or fabricated to be a percentage of a logic level below the highest input logic level to which they are to conduct. The suggested percentage of each should be in the range of 50% to 75% of the logic step voltage (LSV) such that an overlap of on branches is obtained when the circuit is switching from one output logic level to another. This percentage is called the "overlap percentage" (OP) and should be the same for all switches used in digital applications. Analog applications may require that the $V_{GS(TH)}$ and/or OP and/or the LSV be variable.

When developing a logic function or logic synthesizing a circuit, it is necessary to calculate the appropriate or required threshold voltages for each of the FETs. In order to calculate the $V_{GS(TH)}$ for a particular FET, the appropriate equation is selected according equation is selected according to the FETs channel type from the following two equations:

P-channel: $V_{GS(TH)} = V_i - (V_o - (OP*LSV))$ and

N-channel: $V_{GS(TH)} = V_i - (V_o + (OP*LSV))$

Where:

Vi is the input logic level voltage limit (upper or lower as appropriate) to which the branch responds;

Vo is the output logic level voltage;

LSV is the logic step voltage; and

OP is the selected overlap percentage preferably in the range of 55% to 75%.

The noise immunity of a SUSLOC circuit ranges from approximately 45% of a logic level to several logic levels due to the logic level domains, tolerances of the switches and power supplies, the high impedance of FETs, and the overlap percentage. The output of some functions change one logic level with an input change of two or more logic levels, hence the noise immunity in SUSLOC can range several logic levels.

## 1.3. Quaternary Logic

Quaternary logic system is a four valued (radix-4) logic meaning that there are four possible values for each digit.

### 1.3.1. Radix Selection

Several factors have influence in deciding the best radix usable. Obviously, in theory, higher radix would be best to represent as many numbers as possible. But, in practice, the limits of usability and availability of suitable devices limits the usability of higher radix based MVL circuits. Following three factors capture the tradeoffs in choosing appropriate radix.

**Area** : Increased data density of multiple valued logic circuits does help, in principle, to reduce the area when compared to equivalent binary circuits [MT08]. Each of the circuits stores more information per bit. The net result is that the large amount of data sets can be combined and implemented in lesser area. However, at smaller circuits, the additional overhead of "supplementary" logic in the proposed SUSLOC circuit structures does increase the area when compared to their equivalent binary gates. This is depicted in the Figure 5.

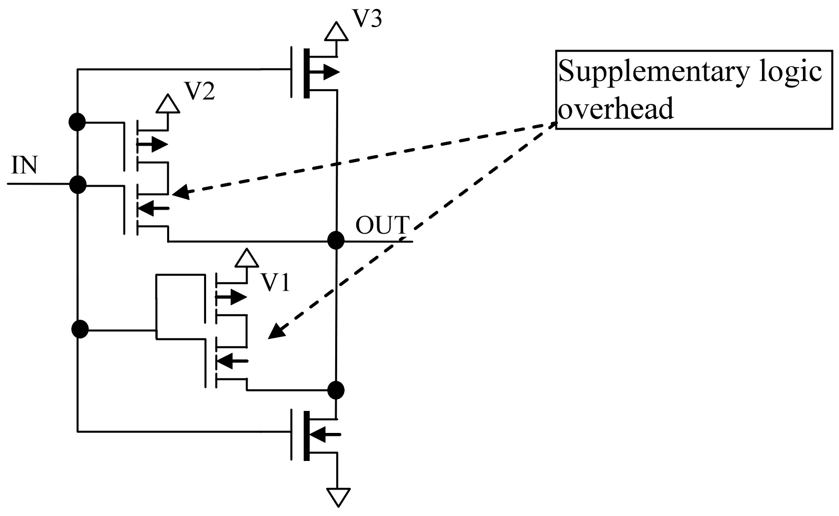


**Figure 5: Area Impact of Higher Radix**

So, the area advantages can only be seen in larger circuits. The logic duplication due to binary logic spread is avoided in MVL circuits. Also higher radices would allow the increased number of functions that can be implemented, making it easier for larger and more complex functions implementation.

Another important advantage is the reduction of signal connections/wires. The reduced wires would reduce the size of the chip and also improve the routability of the design. One of the critical challenges in the Deep Sub-Micron technologies is the routing congestion and also the printability (fabrication) of close proximity of the wires. The limitations of the existing fabrication equipment would create several manufacturing defects like shorting of the wires, open of the wires etc. creating lot of part defects and yield loss. So, reducing the number of wires would significantly improve the device manufacturability and area improvement.

One key metric to consider for any MVL circuits though is the interface logic to the traditional binary circuits. The interface to and from binary logic to the MVL logic does need to have the level conversion to allow successful integration. Circuits namely "radix converters" help to address the cross-region interface needs. The radix conversion is relatively easy for radices which are power of two. (example, radix-2 (binary), radix-4 (quaternary) and radix-8, radix-16 etc.) The radix conversion process gets complex and requires more careful handling for other radices like radix-3, radix-5, radix-6, radix-7, radix-9 etc.

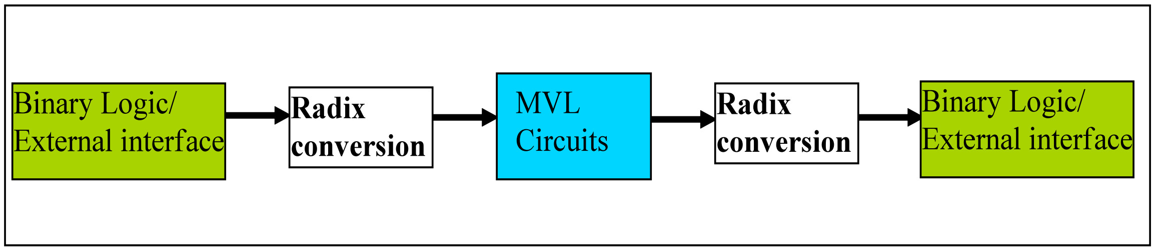
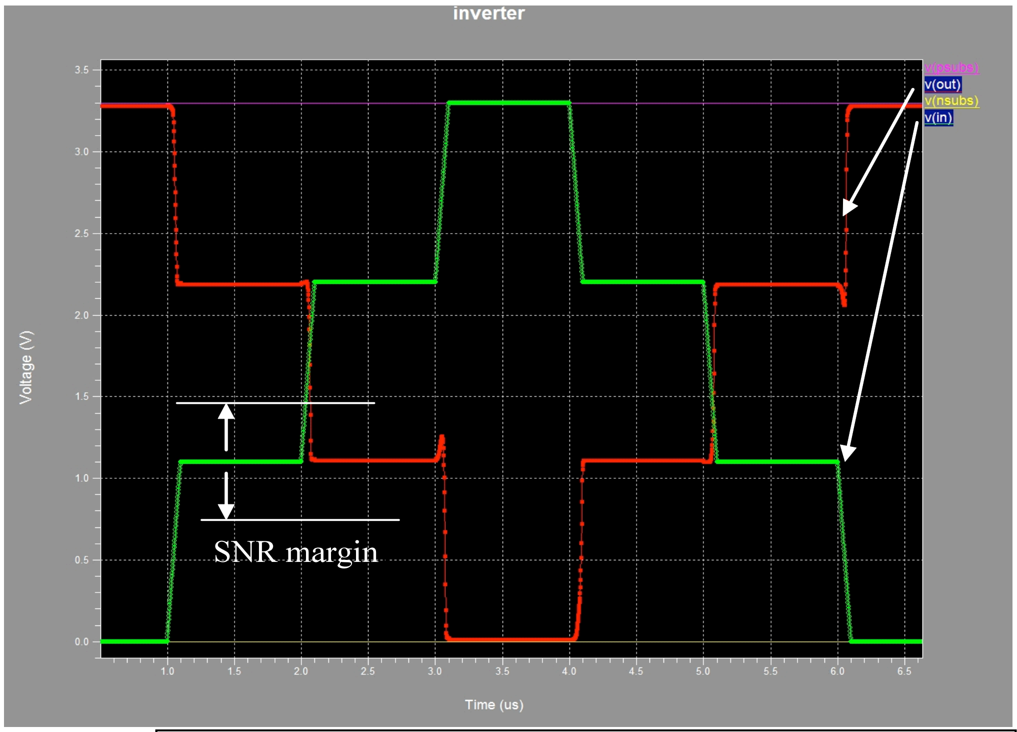

Binary Logic/ External interface
Radix conversion
MVL Circuits
Radix conversion
Binary Logic/ External interface

**Figure 6: Area Impact due to Radix Conversion**

So, in terms of area, the higher the radix, the better it is. Higher radix (like 5 and above) would improve the area. But the radices that are not power of two tend to need more complicated interface logic than for the radices which are power of two.

**Signal-to-noise ratio**: Multiple voltage values are used to represent various logic levels used in MVL circuits. Increased radix would mean increased number of logic levels. Each logic level shall have a dedicated voltage source. And the most important issue is how to have a sufficient signal-to-noise ratio that guarantees the circuit operation in the presence of noise. Noise in the signals can be caused by several factors in the circuits like power supply noise, cross coupling noise etc.

Due to advances in the Deep-Sub Micron (DSM) technologies, the nominal operation voltage of circuits is scaling downwards to 1.0V-1.2V range. If MVL circuits are to be implemented at these technologies, the voltage step value significantly comes down to the range of 300mV to 400mV for quaternary logic. The concept is illustrated in Figure 7 which shows the simulation output of the quaternary inverter. Here the voltage step value is 1.1V. We can readily see that reducing the step value to 300-400mV would leave very small SNR margin to clearly distinguish various logic levels in the circuit.



Courtesy: Omnibase logic simulation of Quaternary inverter, I implemented

**Figure 7: Impact on Signal-to-Noise Ratio**

The reduced voltage step eliminates the available signal-to-noise margin. The signal transitions are to be well constrained to limit the potential voltage swings during the logic level switching. The inability to handle the noise would restrict the use of MVL technology. So, this poses serious threat to the higher radix circuits.

**Cost due to need for several threshold values**: Each logic level adds the complexity of creating new threshold values for the transistors to be able to switch effectively between the logic levels. The increased number of threshold voltages would increase the cost of the fabrication due to additional implementation steps. Also increased manufacturing variations in the deep submicron technologies would increase the variations of the threshold voltages. The multiple valued circuits implemented with FETs are very sensitive to the variations in the threshold voltages. Any changes in Vth values could directly overlap the close-by logic levels causing catastrophic device failures. This poses the threat of yield reduction which again increases the cost of products. So, effectively, higher the radix, the costlier it is to manufacture the devices.

**Performance:** Performance of multiple valued circuits gets better with the increased radix. But, again, larger the radix, the more complicated the timing analysis would get because we need to account for several design margins for various physical and electrical effects. Some examples of the effects are the increase in crosstalk glitches at DSM technologies as we had shown in [DS+03] and [D04]. But, primarily, the increase in radix would achieve better performance with some caveats of increased complexities in the actual timing closure.

**Power consumption :** Power primarily consists of three parts : Dynamic power, Active leakage power and Standby leakage power. The Figure 8 illustrates the three powers in the context of standard CMOS inverter circuit.

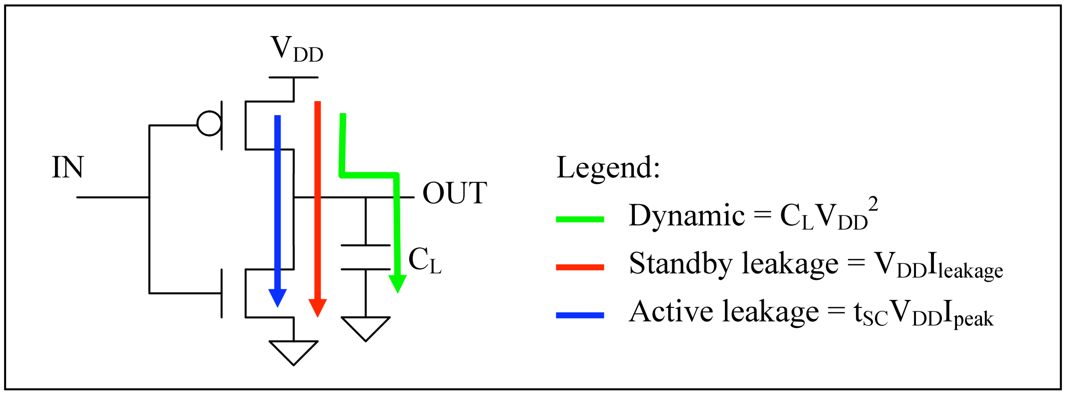
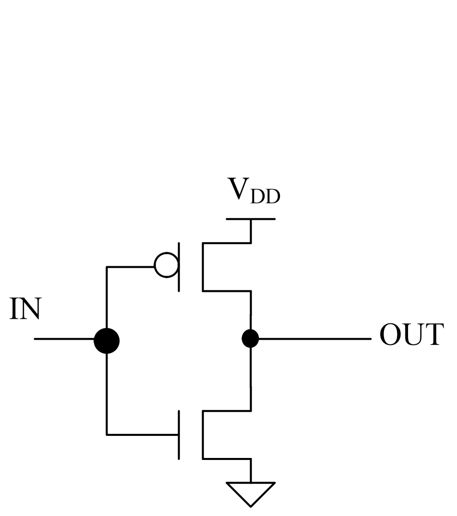
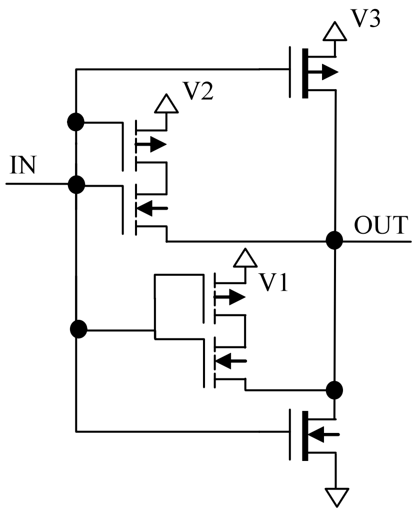

$V_{DD}$
IN
OUT
$C_L$
Legend:
🟩 Dynamic = $C_L V_{DD}^2$
🟥 Standby leakage = $V_{DD} I_{leakage}$
🟦 Active leakage = $t_{SC} V_{DD} I_{peak}$

**Figure 8: CMOS Inverter**

The Dynamic power is also referred to as the Switching power. Typically, this power is dominant of total power consisting of 70-75% of the total power. The Active leakage being the next higher component, generally consisting of 15-20% range. The remaining 5-10% power is the leakage power.



a) CMOS inverter



b) Quaternary Radix-4 inverter

**Figure 9: Inverter Comparison : Radix-2 vs Radix-4**

The comparison of power can be done using the Quaternary inverter case. As can be seen from the Figure 9, Radix-4 circuit (SUSLOC) requires additional two transistors for each logic level. Each additional transistor introduces the additional input capacitance. However, for correct comparison, each SUSLOC inverter is equivalent of two CMOS binary inverters.

So, assuming that each transistor's input capacitance is approximated as **C**, the total capacitance for SUSLOC inverter gate is = 6C

Whereas, the total capacitance for two CMOS inverters is = 2*2C=4C

Now, consider the case of logic switch from 0 to 1 in CMOS inverter case. The voltage supplies are 0V and 3.3V. Assuming the frequency of operation is f and a switching factor of 0.5, the total switching power of the transition from 0 to 1 is

$CV^2f = (4C)*(3.3)^2*f *0.5 = 21.78*C*f$

For the case of quaternary SUSLOC, the equivalent transition is from 0 to 1.1V to 2.2V to 3.3V. The average switching voltage in any given cycle is 1.83V. The average toggle factor is 0.75 because there are total 12 switching cases in the total 16 possible transition states of quaternary gates. The switching power for each of the transition is

$$CV^2f = (6C)(1.83)^2*f*0.75 = 15.07*C*f$$

This shows that the Radix-4(SUSLOC) circuits have switching power consumption savings of over 30% when compared to equivalent CMOS binary circuits. However, the requirement of larger number of transistors in the quaternary circuits would tend to increase the leakage power for the radix-4 case due to excessive standby leakage. But, because switching power is the dominant portion, the radix-4 circuits' fare well compared to binary circuits when total power is measured. The previously described factors estimated for a 16-bit adder circuit reference and are depicted in Figure 10.

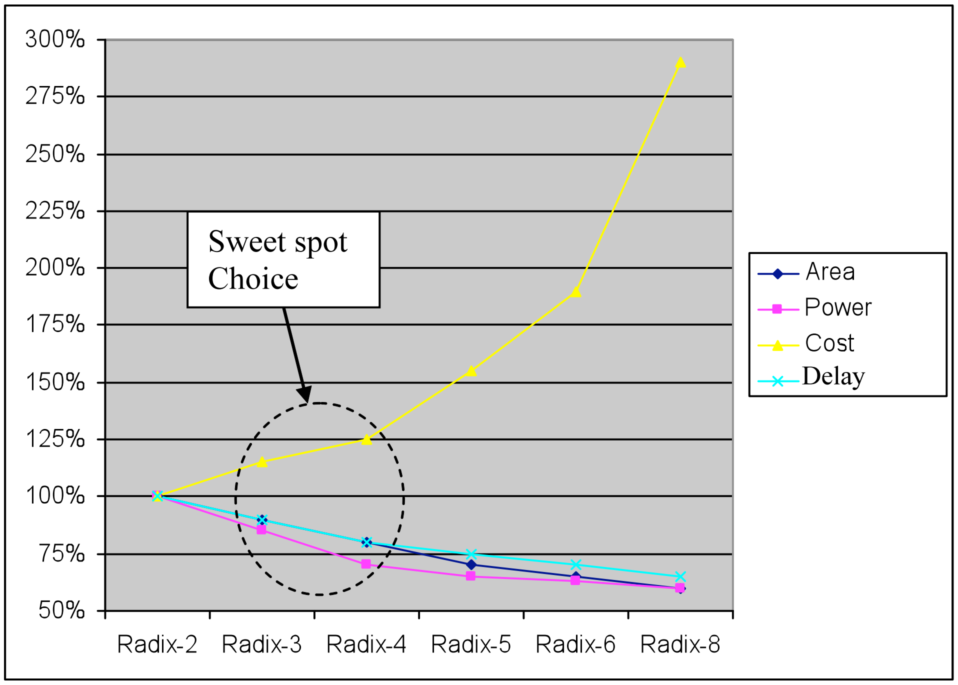


**Figure 10: Cost vs. Benefit Comparison of Various Radices**

For all the factors, radix-2 (binary) is chosen as a reference. As can be seen from this estimation, radices above 4 would have exponential cost increase due to the cost factors discussed before. While the benefits of area and performance keep getting better, exponential cost would prohibit the usage of higher radix MVL circuits with existing components and existing technology. May be the new and evolving technologies like Quantum devices would allow higher radix MVL circuits to be implemented with lesser cost when compared with radix-2 circuits. Also, Brain H. reported in [H01] as shown in the Figure 11 that overall benefits would start deteriorating for higher radices.

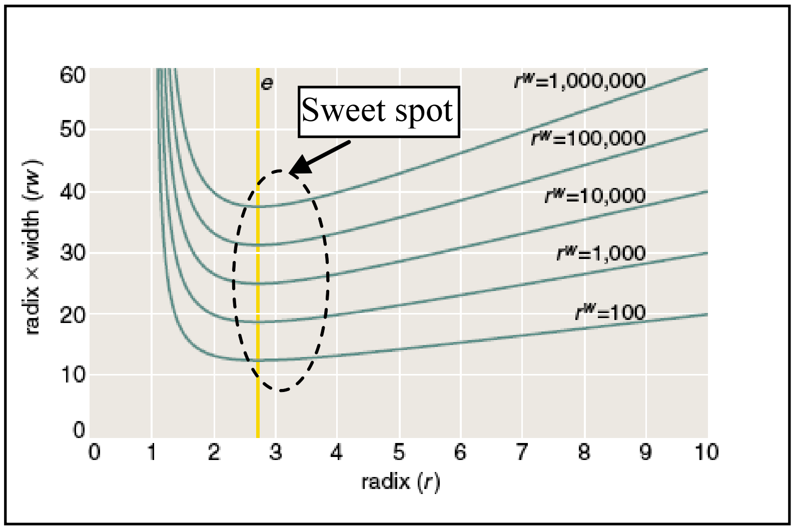


**Figure 11: Selection of Radix**

Also, radix-4 is a power of two allowing for efficient interfacing with binary circuits and because the Voltage-mode operation offers reduced power dissipation characteristics as compared to Current-mode technologies.

Radix-4 (quaternary) is chosen as the radix for this work. The quaternary arithmetic circuits are implemented by extending the basic SUSLOC structures as described in [O00] with the addition of four literal-selection one-place logic gates. The logic gates designed are functionally complete set in the sense that any arbitrary quaternary switching function can be realized using them. Several different addition circuits (ripple-carry, carry-look ahead, and carry-select) and multiplier circuits (serial and parallel) are designed and simulated. The area, power, and estimated performance are analyzed both among themselves and with equivalent binary versions.

For the quaternary circuit implementation, the voltage step value chosen as 1.1V. The threshold voltage values for each transistor calculated according to the following formula obtained from [O00]:

$$V_{GS}(th) = V_i - (V_o + OP \times LSV)$$

Where $V_i$ is input voltage level, $V_o$ is output voltage level, $OP$ is overlap percentage and $LSV$ is logic step voltage. Table 1 lists the voltage sources used for various logic levels in the construction of the quaternary gates.

**Table 1: Quaternary Voltage Sources**

| Logic level/Supply Voltage | Value (V) |
|---|---|
| 0/GND | 0 |
| 1/V1 | 1.1 |
| 2/V2 | 2.2 |
| 3/V3 | 3.3 |

## 1.4. Quaternary (radix-4) Recoding Based Binary Squaring Circuits

Squaring circuits are widely used in several applications such as signal processing, graphics and other arithmetic intensive operations. In several applications, existing multiplier circuits are used to perform squaring operation as well. But, a dedicated squaring circuit would enable optimized area and power when there is extensive usage of squaring operations. Several squaring circuit approaches were proposed previously [C71, D85]. Recent proposal in [M09] takes the advantage of symmetry in squaring operation and applies the dual operand radix-4 recoding to optimize the power and performance of the squaring circuits.

We have conducted research on squaring methods and implemented the proposed quaternary (radix-4) dual operand recoding squaring method [DTM09]. The synthesis experiments were conducted using the 90nm and 130nm technologies. The results indicate that quaternary recoded squaring circuits do yield lower power and higher performance when compared to equivalent regular binary multiplier circuits.

## 1.5. Validation of Quaternary Circuits

The methodology used is to design the quaternary cells at the transistor level and using SPICE simulations to characterize their behavior. The quaternary cells are then used to manually construct the addition and multiplier circuit architectures. The architectures are functionally simulated using the System Verilog language that allows for efficient modeling capabilities for the description and simulation of large MVL circuits [AG+07]. Analysis of resulting circuits is performed using commercially available Synopsys tools. The equivalent two-valued (binary) logic circuits are coded in Verilog HDL and synthesized using Synopsys design compiler for benchmarking. Area (number of transistors), Switching power and logic depth (number of stages in timing critical path) are used as metrics for comparison of quaternary circuits with their binary counterparts.

## 1.6. Notations used for Quaternary Algebra

The quaternary logic circuits used in this work are modeled by a four-valued switching algebra over variables that may take on the values $\{0,1,2,3\}$. The notations for the different operators used in the quaternary equations in this report are defined in Table 2. The notation also includes the explicit specification of state values used.

**Table 2: Quaternary Operator Definitions**

| **Operator** | **Definition** |
|---|---|
| ● | Quaternary *MIN* |
| + | Quaternary *MAX* |
| {} | Indicates logic literal values of the corresponding quaternary variable |

The decisive literal of a quaternary variable, x, is denoted by $x^{\{0\}}$, $x^{\{1\}}$, $x^{\{2\}}$, or $x^{\{3\}}$ [E93] where $x^{\{a\}}=3$ when $x=a$, otherwise $x^{\{a\}}=0$. There are many possible algebraic operators for this logic system: 256 one-place operators and in excess of four billion two-place operators. The small subset of operators used in this work corresponds to relatively easily implemented SUSLOC circuits. We use five different one-place operators; the quaternary inverter denoted as $\overline{x}$ where $\overline{x}=3-x$, and the four decisive literal functions, $x^{\{a\}}$, described previously. In terms of two-place operators, the *MIN* and *MAX* functions are utilized as defined in [P21]. The *MIN* function is denoted as $x{\cdot}y$ where $x{\cdot}y=x$ if $x<y$, otherwise $x{\cdot}y=y$. When two literals or terms appear next to one another, the *MIN* operator, ·, is implied to be present. The *MAX* function is denoted by $x+y$ where $x+y=x$ if $x>y$ else $x+y=y$.

## 1.7. Comparison to Current Methods

Several approaches to MVL circuit design have been proposed in the past as indicated in [S88, S93, KK+87, C90]. Recently, there have been several implementation techniques proposed for Current-mode MVL circuits. Initial Current-mode multiple valued logic circuits using $I^2L$ or CMOS have been proposed [DM+77, KK+88] but suffer from the current mirror errors and threshold detector errors. Various authors have worked on the realization of different types of circuits using Pass Transistor Logic. In [I86] the authors proposed pass transistor networks with threshold voltage. Such circuits are very much suitable for multiple-valued logic.

The realization of multiple-valued flip-flops (MVFF) also has been studied by different authors [ZW90, AH75]. In [EI74] the authors pointed out different approaches of realization of MVFFs. In [BZ+04] MVFF using Pass Transistor Logic has been proposed and reported increase performance when compared with traditional CMOS circuits. System design methodologies and Circuit design methodologies both were discussed in [S93]. Both Decoder-Logic-Encoder (DLE) and Theta ($\theta$) based circuit design methodologies were discussed. As highlighted in [S93], standard CMOS based Current-mode MVL circuits can be slower compared to their binary counterparts.

The adiabatic design methods for MVL circuits are proposed in [CO+96]. But, the limiting factor of the proposed low-energy logic schemes is the problem of efficiently generating and distributing the multiple phase power clocks. The scalability of the same principles to larger circuits would be the main challenge. Recently, there has been an increased interest and research of emerging technologies such as quantum circuits and reversible circuits and application of the quantum principles to the MVL circuit design

[MT08, QC04, CS+89]. While these technologies provide promising advantages, they are still relatively in early stages of large circuit design and adoption into mainstream applications. However, quantum ternary logic has the limitation that conventional binary logic functions cannot be very easily represented using the ternary base and the developed methods are applicable only for logic functions expressed in ternary base. A very promising alternative is quaternary logic, using which, besides quaternary logic functions, binary logic functions can be expressed by grouping 2-bits together into quaternary values.

Etiemble nicely summarized in [E92] the comparison of development of MVL circuits to that of binary circuits. While the argument centers around Current-mode circuits, most of the conclusions can be attributed to Voltage-mode MVL circuits as well. The main conclusion in this work [E92] suggests that MVL circuits do need to offer benefits in the front of interconnect optimization and also in performance and power improvements in order for them to be widely used. Also, one of the main suggestions of Etiemble is to compare the performance in terms of number of logic levels within the same technology. Both of these suggestions are adopted for our work.

There has been increased interest in quaternary circuits due to their unique advantage of ease of interfacing to binary circuits. Mangin et. al [MC86] demonstrated the fabricated quaternary encoder and decoder circuits on a gate-array integrated circuit. Shanbhag et. al [SN+90] fabricated additional quaternary circuits such as quaternary registers using the 2-um CMOS technology. Recently, Shirahama et. al [SH08] proposed quaternary adders based on output generator sharing for improving performance of both Current-mode and Voltage-mode circuits, and highlighted the observation that Voltage-mode circuits do consume low-power when compared to their Current-mode counterparts. The voltage swings of 0.4V in the proposed approach are very tiny which pose the threat of signal noise margin and hence are not easily extendable to larger circuit implementations. Design and verification of quaternary adders in Field Programmable Gate Arrays (FPGA) are demonstrated in [DW08]. Results presented indicate that the quaternary signed digit (QSD) adders do provide improved performance compared with ripple carry based adders. But, it is also concluded that the area complexity of QSD adders grows along with size of the operands and is more than the other types of adders. Hence, the proposed QSD adders do suffer from higher power consumption.

Recently, an approach structurally similar to that of a static-CMOS binary circuit, called the Supplementary Symmetrical Logic Circuit structure (SUSLOC), was proposed and patented [O00, OC00, OC01]. SUSLOC MVL circuits theoretically offer a stable circuit implementation structure for any integer radix, and also allow the use of readily available circuit elements to construct logic circuits. Previous work has utilized SUSLOC technology for ternary systems [OC01, KA+03]. Insulated gate Field Effect Transistors (FET) are used as the basic building blocks for SUSLOC circuits.

We tried to address the key concern of modern circuit design: Power. We have chosen Voltage-mode to effectively address the power challenge. At the same time, we wanted to implement the circuits using the readily available CMOS technologies for more immediate applications. We have adopted the proven SUSLOC technology which was successfully demonstrated the manufacturability of ternary circuits using the standard CMOS process technology. We, however, adopted the quaternary as our radix for the ease of interface with the binary logic to allow mixing of the design circuits with the normal binary circuits. With these, we believe we take the advantage of benefits of MVL circuits and still come up with readily usable and most importantly, lower power, arithmetic circuits. We also applied the quaternary principles into squaring circuits and realized low power and high performance squaring circuits.

## 1.8. Organization of the Dissertation

This dissertation report is organized into several chapters. Following are the details about the subsequent chapters.

Quaternary Switching Functions: The background details about topics in quaternary switching theory and the functionally completeness proof are elaborated in this chapter.

Quaternary Library Circuits: Details about the implementation of set of preliminary quaternary circuits is provided in this chapter. Transistor level implementation details of functionally complete quaternary gates and their SPICE circuit simulation results are detailed. The preliminary set is extended to other gates like quaternary sequential cell and 4-to-1 quaternary multiplexer cell as described in this chapter.

Quaternary Arithmetic Circuit Architectures: This chapter starts with details of quaternary half adder and full adder circuits. The quaternary arithmetic circuit implementation is then extended to higher levels of arithmetic circuits. Different adder architectures are implemented using the preliminary quaternary gate set. Two varieties of multiplication circuit schemes namely serial and parallel are researched upon whose details are provided as well in this chapter.

Approximate Squaring Circuit: The details on the work to implement approximate squaring circuits utilizing the radix-4 (quaternary) dual recoding are provided in this chapter.

Quaternary Circuits Modeling and Validation: This chapter elaborates on the proposed validation techniques for quaternary circuits using the System Verilog. Details of System Verilog offerings with which quaternary circuit modeling is attained are included in this chapter.

Results: Quaternary circuits are benchmarked against the equivalent binary circuits. Area, timing and power are analyzed for quaternary circuits and are compared with the binary circuits. The benchmarking comparisons for approximate squaring circuits are included as well.

Conclusions and Future Challenges: The final conclusions on the research work on quaternary circuits, their implementation and validation methods are summarized in this chapter. The remaining challenges for successful quaternary circuits' wider adoption are summarized as well in this chapter.

# Chapter 2

# QUATERNARY SWITCHING FUNCTIONS

A necessary requirement for the practical use of SUSLOC quaternary logic for circuit design is the identification of a set of primitive logic gates that allows any arbitrary switching function to be realized. Furthermore, each logic gate should be easy to implement and efficient in SUSLOC technology. A representation of a quaternary switching function that is analogous (but not identical) to the binary sum-of-minterms representation can be formulated using the *MAX* function in place of the binary inclusive-OR function, and product terms may be formed using the *MIN* function in place of the binary *AND* operation. The truth tables for two-variable *MAX* and *MIN* functions are given in Figure 12.

| | | **A** | | | |
|---|---|---|---|---|---|
| | | **0** | **1** | **2** | **3** |
| **B** | **0** | 0 | 1 | 2 | 3 |
| | **1** | 1 | 1 | 2 | 3 |
| | **2** | 2 | 2 | 2 | 3 |
| | **3** | 3 | 3 | 3 | 3 |

MAX

| | | **A** | | | |
|---|---|---|---|---|---|
| | | **0** | **1** | **2** | **3** |
| **B** | **0** | 0 | 0 | 0 | 0 |
| | **1** | 0 | 1 | 1 | 1 |
| | **2** | 0 | 1 | 2 | 2 |
| | **3** | 0 | 1 | 2 | 3 |

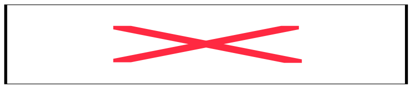
MIN

**Figure 12: Truth table of Two-input MAX and MIN Functions**

To show that the collection of operators over the quaternary-valued logic defined previously is sufficient to specify all possible switching functions, some definitions are provided [MW86].

*Definition 1*: A specific *product selection term* is denoted by $J_{Pi}$ where the integer index $P_i$ denotes the particular product selection term for which $J_{Pi}$ is non-zero. $P_i$ is the numeric value (in the decimal system) given by the digit string $k_1k_2\ldots k_n$ where $k_i \in \{0, 1, 2, 3\}$ and is in the range $[0,4^n]$. □

As an example, consider the case for $n$=4, then $J_{35}=x_1^{\{0\}}x_2^{\{2\}}x_3^{\{0\}}x_4^{\{3\}}$ since $0\times4^3+2\times4^2+0\times4^1+3\times4^0=35$.



(1)

*Definition 2*: A *minterm* of a quaternary switching function is formed by combining the $P_i^{\text{th}}$ logic value in the truth table with the $J_{Pi}$ product selection term using the *MIN* operation. □

*Definition 3*: A *sum-of-minterms* (SOM) form of a quaternary function is a form consisting of all minterms combined with a *MAX* operator. □

As an example, consider a quaternary switching function of $n$ variables, $f$, where each function range value corresponding to the $P_i$th product term in the truth table is denoted as $f_{Pi}$. The function $f$ can be expressed in SOM form, $f_{\mathrm{SOM}}$, as shown in Equation (2).

$$f_{SOM} = f_{P0} \bullet J_{P0} + f_{P1} \bullet J_{P1} + \ldots + f_{P4^n} \bullet J_{P4^n} \tag{2}$$

<u>*Lemma 1*</u>: The SOM form of a quaternary function is a canonical representation.

<u>*Proof*</u>: The truth table for a switching function consists of rows with all possible variable assignments and their corresponding function range values. For a completely specified quaternary switching function of $n$ variables, the corresponding truth table representation contains $4^n$ rows. The truth table representation can be expressed as a column vector of function range values with each vector component in a well-defined order corresponding to the particular assignment of domain variables, the $P_i$th term, denoted as $\underline{\mathbf{F}}$. Any permutation of the vector component values or any change in one or more of the vector component values yields a vector (or corresponding truth table) that represents a different switching function. Thus, every switching function has a unique vector representation. Because each product selection term is non-zero for one and only one product assignment, Equation (2) can be formed as the inner product of a row vector whose components are all unique selection product terms (denoted as $\underline{\mathbf{J_P}}$) and $\underline{\mathbf{F}}$ as $f_{\mathrm{SOM}} = \underline{\mathbf{J_P}} \cdot \underline{\mathbf{F}}$. Since each $\underline{\mathbf{F}}$ vector is unique for a given switching function and since Equation (2) can be formed using $\underline{\mathbf{F}}$, the proof is complete. □

*Theorem 1*: The collection of quaternary logic operators consisting of the two-place *MIN* and *MAX* functions, the four decisive literal functions $\{x^{\{0\}}, x^{\{1\}}, x^{\{2\}}, x^{\{3\}}\}$, and the constants {0, 1, 2, 3} form a functionally complete set over all possible quaternary-valued switching functions.

*Proof*: Consider an arbitrary quaternary switching function of $n$ variables of the form $f$: $\{0, 1, 2, 3\}^n \rightarrow \{0, 1, 2, 3\}$. For a completely specified function, $f$, the range values associated with the $P_i$th minterms are constants from the set {0,1,2,3} denoted by $f_{Pi}$. Clearly, Equation (2) is a general representation of any arbitrary quaternary switching function since the constants denoted by $f_{Pi}$ can represent any assignment from {0, 1, 2, 3}. From Lemma 1, Equation (2) is also proven to be canonic. Therefore, any completely specified function could be represented in the form of Equation (2), which is based only on the quaternary *MIN*, *MAX*, decisive literal functions,and the set of constants {0, 1, 2, 3}. □

# Chapter 3

# QUATERNARY LIBRARY CIRCUITS

We are interested in elementary quaternary circuits that have sufficient representative capability to efficiently implement quaternary arithmetic circuits. We extended the SUSLOC implementation principles from [O00] to design a basic set of quaternary circuits to form a library of circuits. This set of gates is designed using existing CMOS FET components and does not require any special technologies or materials to fabricate them [O00,O99]. Subsequent sections in this chapter provide implementation details for these circuit library elements and their corresponding SPICE simulation results.

## 3.1. Quaternary One-variable Circuits

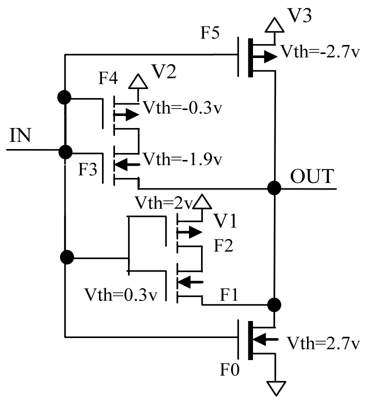
In general, for a single variable function in radix $r$, $r$ possible output values are available for each of the $r$ possible input values [S88]. Accordingly, there are $r^r$ such one-variable functions. Therefore, there are a total of $4^4$(=256) possible unary functions possible in quaternary logic, which means that there is considerable room for choice when selecting a subset of quaternary one-place logic cells for implementation. A total of four out of 256 functions have constant output values. The remaining 252 functions produce non-constant output logic value combinations for different logic values at the input.

For the purpose of this work, five of the 252 functions are selected for circuit implementation. These are :

- *r*-*x*-1 inversion circuit
- $J_0$ literal selection circuit
- $J_1$ literal selection circuit
- $J_2$ literal selection circuit
- $J_3$ literal selection circuit

### 3.1.1. The r-x-1 Inversion Circuit

A quaternary circuit that performs the *r*-*x*-1 inversion operation is shown in Figure 13. The *r*-*x*-1 circuit is also referred to as a “diametrical negation” or “inversion” function.



| *IN* | *OUT* |
|---|---|
| 0 | 3 |
| 1 | 2 |
| 2 | 1 |
| 3 | 0 |

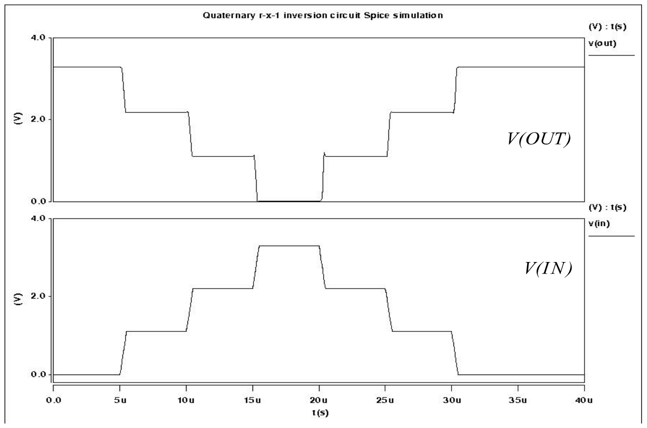


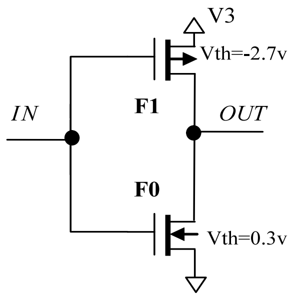
**Figure 13: Quaternary *r*-*x*-1 Inversion Circuit, Truth Table & SPICE Sim Result**

The four-valued variable for which the reference input string is (0,1,2,3) causes the *r-x*-1 circuit to produce output string (3,2,1,0). This circuit is useful in constructing the quaternary *MAX* and *MIN* functions and several larger quaternary circuits.

The operation of quaternary inversion circuit is described here. The logic state "0" at the *IN* input produces the required bias to turn the F5 transistor ON. The output pin *OUT* is then driven by supply *V3*, which forces logic state "3" at the *OUT* pin. On the other hand, logic states "1" and "2" trigger the required bias to turn ON the transistor pairs F3-F4 and F1-F2 respectively. The output pin *OUT* is driven by the *V2* and *V1* supplies to respectively force logic states "2" and "1" at the *OUT* pin. Finally, the voltage value of state "3" at the *IN* pin forces the required bias to F0 transistor to turn ON. This condition forces the *OUT* pin to be shorted to *Ground* which means state "0". The depicted circuit is simulated using SPICE models for the transistors. The waveforms shown in the Figure 13 confirm the circuit's correct functionality.

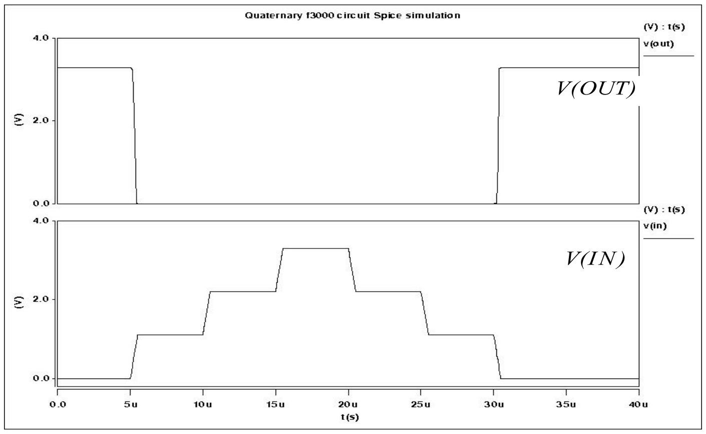
### 3.1.2. Quaternary J0 and J3 Circuits

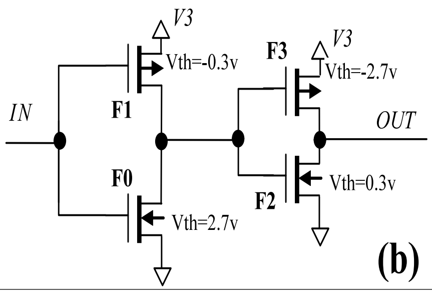
The quaternary one-place literal selection functions $J_0$ and $J_3$, and their circuit implementations are discussed in this section. The two circuits and their simulated behavior are depicted in Figure 14.



| IN | OUT |
|---|---|
| 0 | 3 |
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |

(a)





| IN | OUT |
|---|---|
| 0 | 0 |
| 1 | 0 |
| 2 | 0 |
| 3 | 3 |

(b)

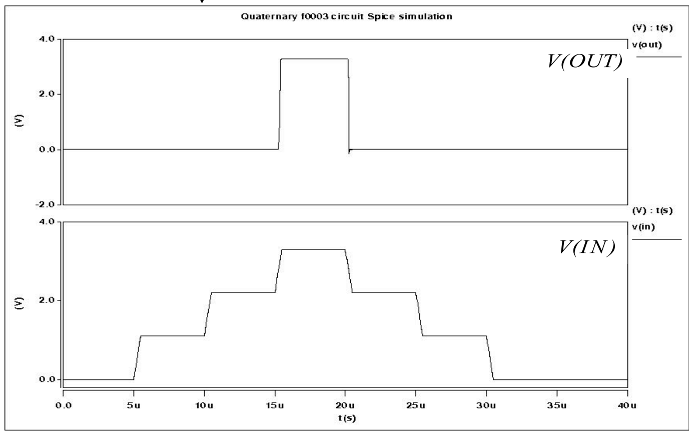
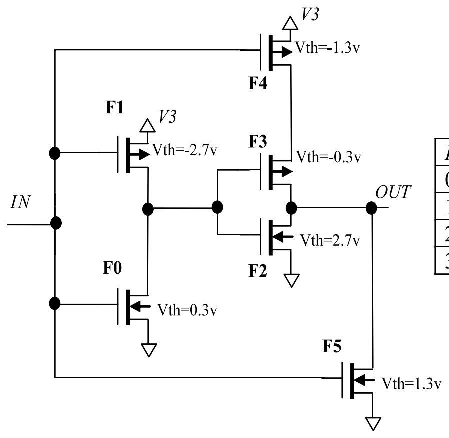


**Figure 14: (a) Quaternary $J_0$ (b) Quaternary $J_3$ Functions & SPICE Sim Results**

The quaternary circuit $J_0$ is very similar to a standard binary CMOS inverter circuit. The threshold voltages of both the P-FET and N-FET are chosen to produce the quaternary $J_0$ function. The $J_3$ circuit is constructed simply by connecting two back-to-back $J_0$ circuits. The threshold voltages of the transistors in this circuit are selected to ensure that the circuit exhibited the required $J_3$ behavior.

### 3.1.3. Quaternary J1 and J2 Circuits

This section discusses the remaining two quaternary one-place literal selection functions, $J_1$ and $J_2$, along with their circuit implementations. The circuits and their simulated behaviors are shown in Figure 16 and Figure 16.



| *IN* | *OUT* |
|---|---|
| 0 | 0 |
| 1 | 3 |
| 2 | 0 |
| 3 | 0 |

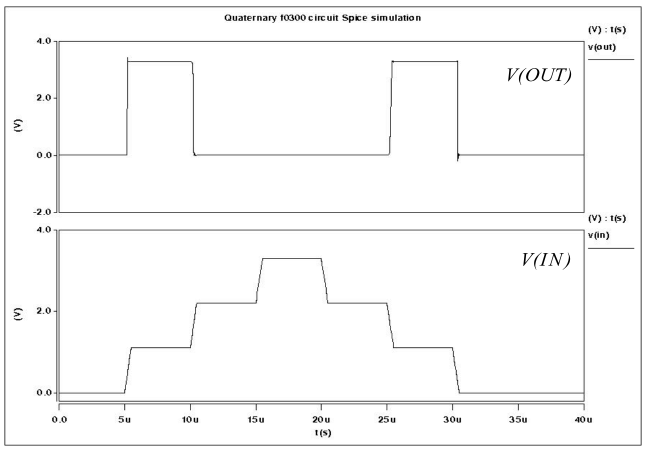
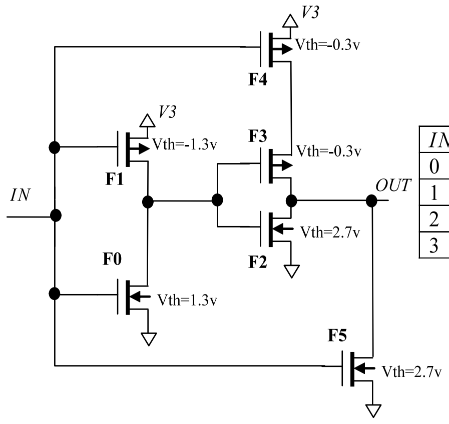


**Figure 15: Quaternary $J_1$ Function & SPICE Sim Result**



| IN | OUT |
|---|---|
| 0 | 0 |
| 1 | 0 |
| 2 | 3 |
| 3 | 0 |

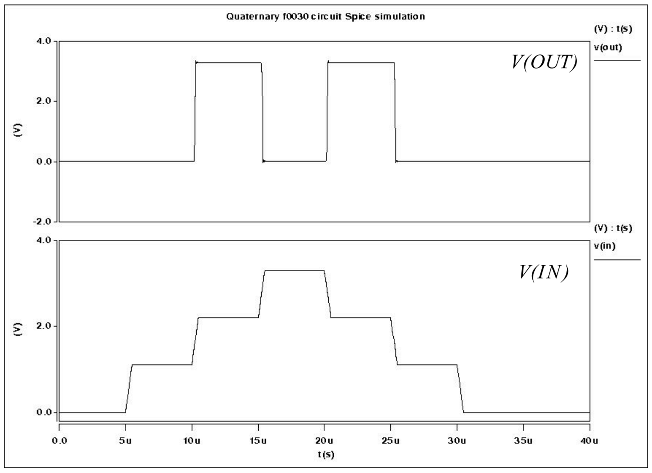


**Figure 16: Quaternary $J_2$ Function & SPICE Sim Result**

The quaternary circuit implementations for functions $J_1$ and $J_2$ are the same, except for the fact that different threshold voltage values are used for the individual transistors. In the case of the $J_1$ circuit, transistor F4 is turned ON only when the logic state at the *IN* input is between states “1” and “2”. This forces supply *V3* to drive the *OUT* pin to logic state “3”.  Alternatively, in the case of the $J_2$ circuit, the logic state at the *IN* input has to be between states “2” and “3” to enforce the same behavior.

## 3.2. Quaternary Two or More Input Circuits

The previous sections discussed the implementation of the one-place quaternary circuit cells. In order to complete the cell library, additional circuits must be included that have two or more inputs. There are a significantly large number of possible two-variable ($4^{16}$) and three-variable ($4^{64}$) functions to consider. We selected a small number of useful two-variable and three-variable quaternary functions to be part of the quaternary circuit library. Two other important circuit elements, the quaternary flip-flop circuit and the 4-to-1 quaternary multiplexer circuit are also implemented. The following is the list of the multiple-variable quaternary circuits implemented:

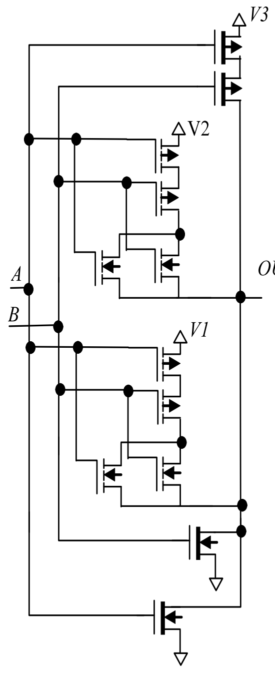
- Two-input *MAX*-inversion and *MAX* circuits
- Two-input *MIN*-inversion and *MIN* circuits
- Three-input *MIN*-inversion and *MIN* circuit
- *D* Flip-flop circuit
- 4-to-1 multiplexer circuit

### 3.2.1. Two-input MAX-inversion Circuit

Traditionally, very commonly used multiple-valued circuits have been the maximum (*MAX*) and minimum (*MIN*) functions [MT08]. For multiple-valued variables $x_i, 1 \leq i \leq n$,

$$MAX(x_1, x_2, ..., x_n) = x_1 + x_2 + ... + x_n, \text{ is the largest } x_i$$

The two-variable maximum (*MAX*) function outputs the larger of the two values presented. The *MAX*-inversion circuit produces an *r*-*x*-1 inverted value of *MAX* function. The two-input *MAX*-inversion circuit structure in Figure 17 is analogous to the two-input *NOR* gate circuit in binary logic. A voltage level of logic state “0” on both of the inputs, *A* and *B,* would force the upper-most transistor pair to be turned ON. This enforces the supply *V3* to drive the *OUT* port and hence produce logic state "3". Alternatively, when either one of the inputs, *A* or *B,* reaches logic state "3", the respective transistor in the bottom-most pair is turned ON. This forces the *OUT* port to be shorted to *Ground* (logic state "0").



| *A* | *B* | *OUT* |
|---|---|---|
| 0 | 0 | 3 |
| 1 | 0 | 2 |
| 2 | 0 | 1 |
| 3 | 0 | 0 |
| 0 | 1 | 2 |
| 1 | 1 | 2 |
| 2 | 1 | 1 |
| 3 | 1 | 0 |
| 0 | 2 | 1 |
| 1 | 2 | 1 |
| 2 | 2 | 1 |
| 3 | 2 | 0 |
| 0 | 3 | 0 |
| 1 | 3 | 0 |
| 2 | 3 | 0 |
| 3 | 3 | 0 |

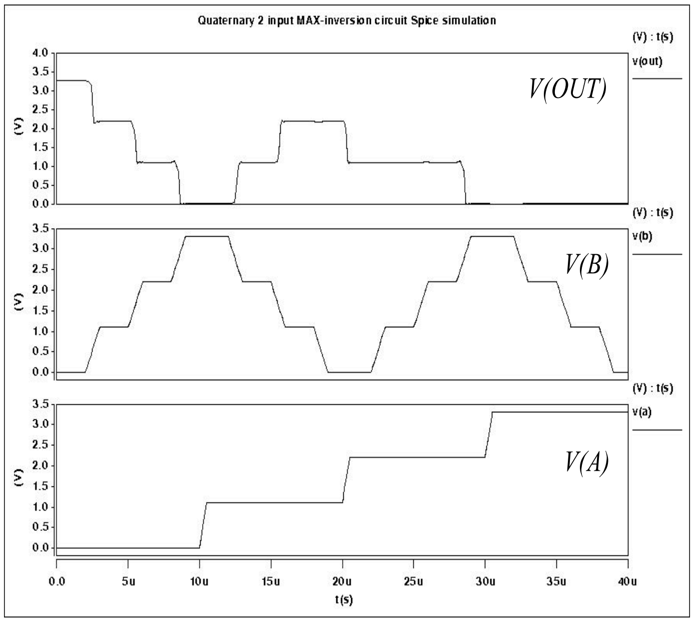


**Figure 17: Quaternary *MAX*-Inversion Circuit, Truth Table & SPICE Sim Result**

When either of the inputs reaches logic state “1” and the other input is less than or equal to logic state “1”, power supply *V2* drives the *OUT* pin. Similarly, when either of the inputs reaches the logic state “2” and the other input is less than or equal to logic state “2”, the power supply *V1* drives the *OUT* pin. The Two-input *MAX* function is implemented by connecting an *r*-*x*-1 inversion circuit at the output of a two-input *MAX*-inversion circuit as shown in Figure 18.

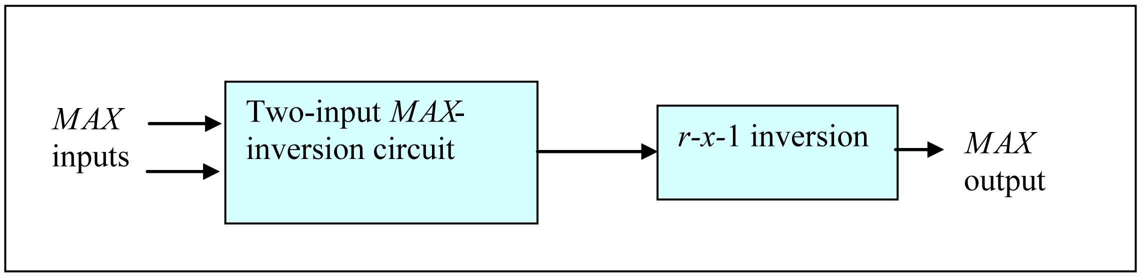


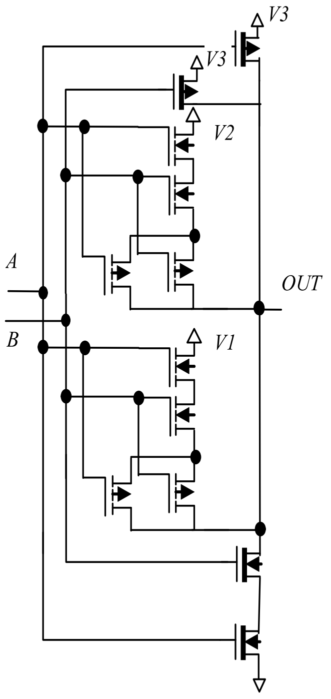
**Figure 18: Quaternary Two-input *MAX* Circuit**

3.2.2. Two-input MIN-inversion Circuit

Another widely used multiple valued function is the *MIN* function. For multiple-valued variables $x_i, 1 \le i \le n$,

$$MIN(x_1, x_2, \ldots, x_n) = x_1 \bullet x_2 \bullet \ldots \bullet x_n, \text{ is the smallest } x_i$$

The two-variable minimum (*MIN*) function outputs the smaller of the two values presented at the inputs. The two-input *MIN*-inversion circuit structure in Figure 19 is analogous to the binary-valued two-input *NAND* gate circuit when implemented using static CMOS technology. The threshold voltages of the transistors are selected appropriately to derive the required logic states at the *OUT* pin. The quaternary two-input *MIN* function is also implemented by connecting an *r*-*x*-1 inversion circuit at the output of the *MIN*-inversion circuit.



| A | B | OUT |
|---|---|---|
| 0 | 0 | 3 |
| 1 | 0 | 3 |
| 2 | 0 | 3 |
| 3 | 0 | 3 |
| 0 | 1 | 3 |
| 1 | 1 | 2 |
| 2 | 1 | 2 |
| 3 | 1 | 2 |
| 0 | 2 | 3 |
| 1 | 2 | 2 |
| 2 | 2 | 1 |
| 3 | 2 | 1 |
| 0 | 3 | 3 |
| 1 | 3 | 2 |
| 2 | 3 | 1 |
| 3 | 3 | 0 |

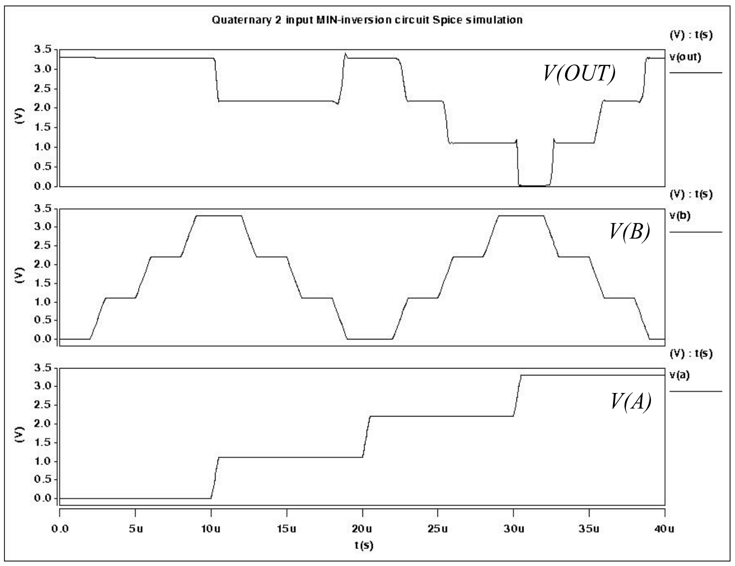
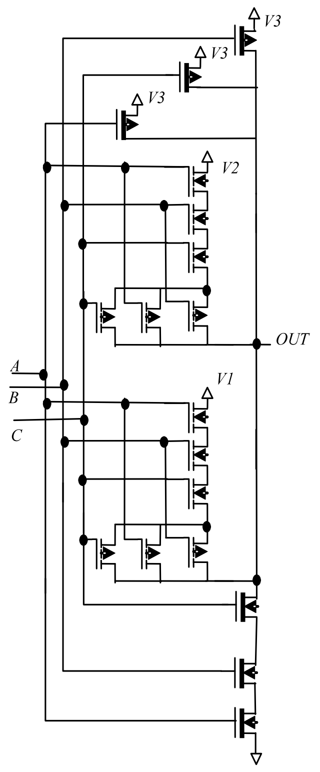


**Figure 19: Quaternary *MIN*-inversion Circuit, Truth Table, and SPICE Sim Result**

### 3.2.3. Three-input MIN-inversion Circuit

Another useful quaternary function that is implemented as part of the library is the three-input *MIN*-inversion circuit. Essentially, the three-input *MIN*-inversion circuit is a simple extension of the two-input *MIN*-inversion circuit with an additional parallel path from the third input. The circuit, its truth table, and the SPICE simulation results are shown in Figure 20. The functionality of this circuit is the same as the two-input circuit, except for the fact that there are three inputs from which the minimum value was selected. The size of the truth table is significantly larger than that for the two-input circuit.



| INPUTS | | | | INPUTS | | | | INPUTS | | | | INPUTS | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **A** | **B** | **C** | **OUT** | **A** | **B** | **C** | **OUT** | **A** | **B** | **C** | **OUT** | **A** | **B** | **C** | **OUT** |
| 0 | 0 | 0 | 3 | 0 | 0 | 1 | 3 | 0 | 0 | 2 | 3 | 0 | 0 | 3 | 3 |
| 0 | 1 | 0 | 3 | 0 | 1 | 1 | 3 | 0 | 1 | 2 | 3 | 0 | 1 | 3 | 3 |
| 0 | 2 | 0 | 3 | 0 | 2 | 1 | 3 | 0 | 2 | 2 | 3 | 0 | 2 | 3 | 3 |
| 0 | 3 | 0 | 3 | 0 | 3 | 1 | 3 | 0 | 3 | 2 | 3 | 0 | 3 | 3 | 3 |
| 1 | 0 | 0 | 3 | 1 | 0 | 1 | 3 | 1 | 0 | 2 | 3 | 1 | 0 | 3 | 3 |
| 1 | 1 | 0 | 3 | 1 | 1 | 1 | 2 | 1 | 1 | 2 | 2 | 1 | 1 | 3 | 2 |
| 1 | 2 | 0 | 3 | 1 | 2 | 1 | 2 | 1 | 2 | 2 | 2 | 1 | 2 | 3 | 2 |
| 1 | 3 | 0 | 3 | 1 | 3 | 1 | 2 | 1 | 3 | 2 | 2 | 1 | 3 | 3 | 2 |
| 2 | 0 | 0 | 3 | 2 | 0 | 1 | 3 | 2 | 0 | 2 | 3 | 2 | 0 | 3 | 3 |
| 2 | 1 | 0 | 3 | 2 | 1 | 1 | 2 | 2 | 1 | 2 | 2 | 2 | 1 | 3 | 2 |
| 2 | 2 | 0 | 3 | 2 | 2 | 1 | 2 | 2 | 2 | 2 | 1 | 2 | 2 | 3 | 1 |
| 2 | 3 | 0 | 3 | 2 | 3 | 1 | 2 | 2 | 3 | 2 | 1 | 2 | 3 | 3 | 1 |
| 3 | 0 | 0 | 3 | 3 | 0 | 1 | 3 | 3 | 0 | 2 | 3 | 3 | 0 | 3 | 3 |
| 3 | 1 | 0 | 3 | 3 | 1 | 1 | 2 | 3 | 1 | 2 | 2 | 3 | 1 | 3 | 2 |
| 3 | 2 | 0 | 3 | 3 | 2 | 1 | 2 | 3 | 2 | 2 | 1 | 3 | 2 | 3 | 1 |
| 3 | 3 | 0 | 3 | 3 | 3 | 1 | 2 | 3 | 3 | 2 | 1 | 3 | 3 | 3 | 0 |

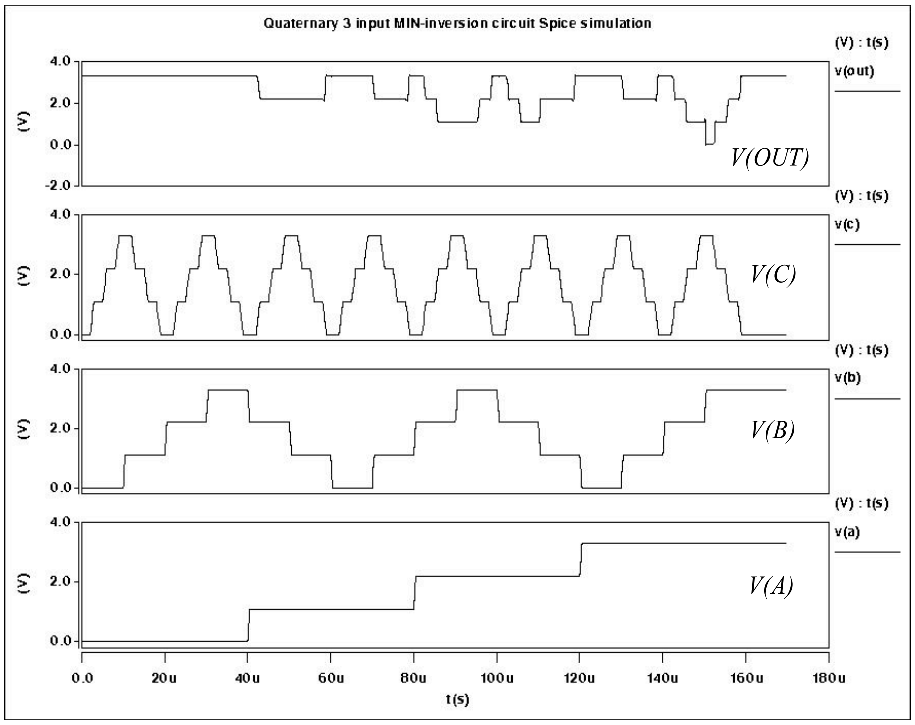
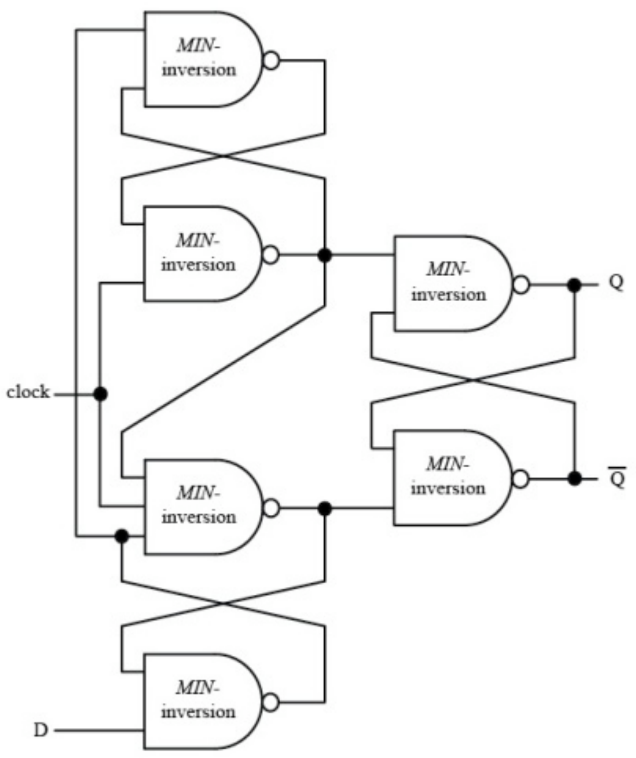


**Figure 20: Quaternary Three-input *MIN*-Inversion Circuit & SPICE Sim Result**

### 3.2.4. Quaternary D-Flip Flop Circuit with Binary Clock

The quaternary circuits discussed thus far can be used to design any combinational function since they form a functionally complete set of operations (with constants) from a mathematical view. The implementation of synchronous sequential circuits such as counters, and shift registers require memory elements. Flip-flops can be used as one-digit memory elements and are essential elements of synchronous sequential circuits. As part of the quaternary library, a *D*-type quaternary flip-flop circuit with a binary clock input is designed. The block diagram and SPICE simulation results for the quaternary flip-flop are shown in Figure 21.



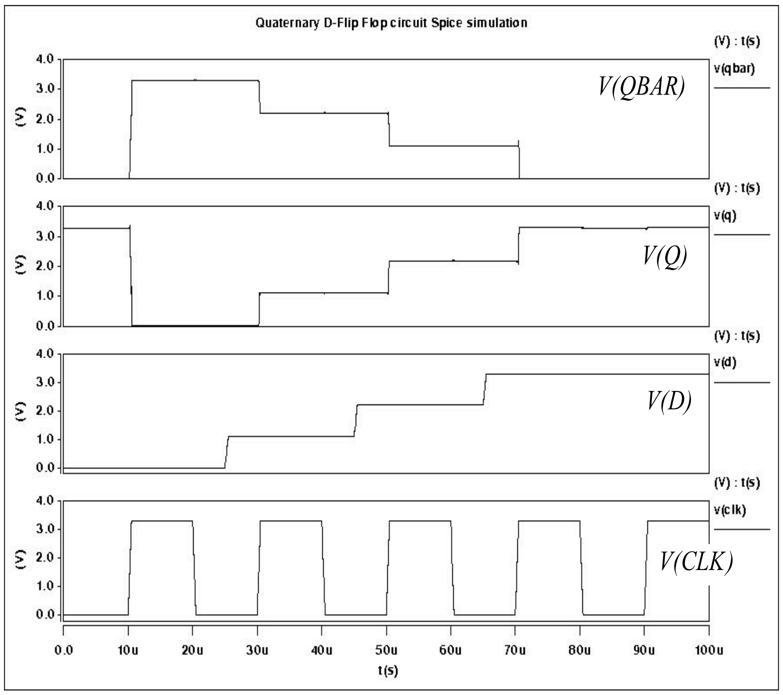


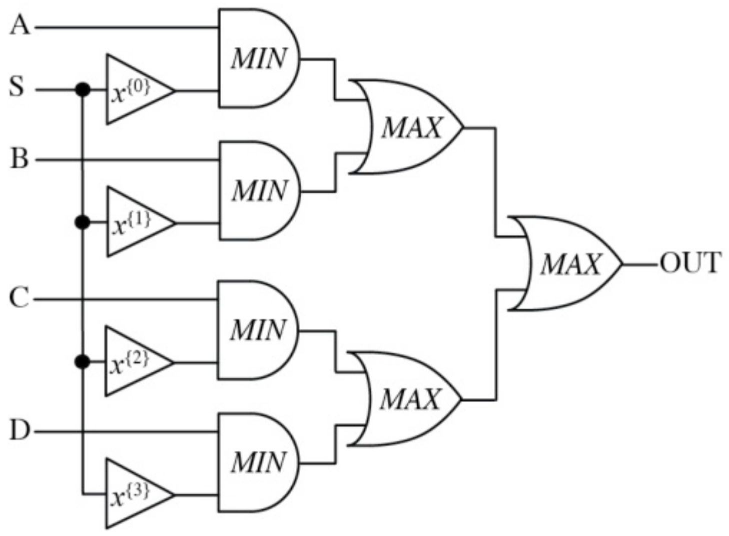
**Figure 21: Quaternary *D* Flip-Flop Circuit and SPICE Simulation Result**

The quaternary flip-flop circuit retains four possible values. A rising edge (transition from logic state “0” to logic state “3”) on the *CLK* input pin latches the logic state at the *D* input. The stored value is maintained at the output pin *Q* with the help of an internal feedback loop. The output signal *QBAR* is the *r*-*x*-1 inverted value of the output *Q*. This circuit’s functionality is verified by supplying a series of pulses on the *CLK* input. In addition, the voltage state at the *D* input is altered periodically such that all four possible voltage values are validated. As illustrated in the simulation waveforms, for every rising edge transition at the *CLK* input, the logic state at the *D* input is correctly stored and held constant at the *Q* and *QBAR* outputs.

### 3.2.5. Quaternary 4-to-1 Multiplexer Circuit

Another important function that is useful for implementing complex quaternary functions is the data selector or multiplexer. A multiplexer circuit selects one of many input signals and forwards the selected input to a single output line. In the case of quaternary logic, a multiplexer of $4^n$ inputs has *n* select bits, which are used to select one of the inputs to be sent to the output. The quaternary 4-to-1 multiplexer circuit shown in **Error! Reference source not found.** is another addition to the quaternary library.

The multiplexer is designed using the *MIN*, *MAX*, and the other basic one-variable functions ($J_0$, $J_1$, $J_2$ and $J_3$). The one-variable functions $J_0$, $J_1$, $J_2$ and $J_3$ are used to detect the voltage value at the select (*S*) input. The outputs of these single variable functions are input to the two-input *MIN* circuits along with their respective inputs. This configuration forces the selected input logic value along with three logic “0” values. Three additional two-input *MAX* circuits then generate the final output at the *OUT* pin. This four-to-one multiplexer circuit is verified using the SPICE simulator by supplying a series of voltage state changes at each primary input along with different voltage values on the *S* pin. The waveforms captured by the SPICE simulation are shown below.



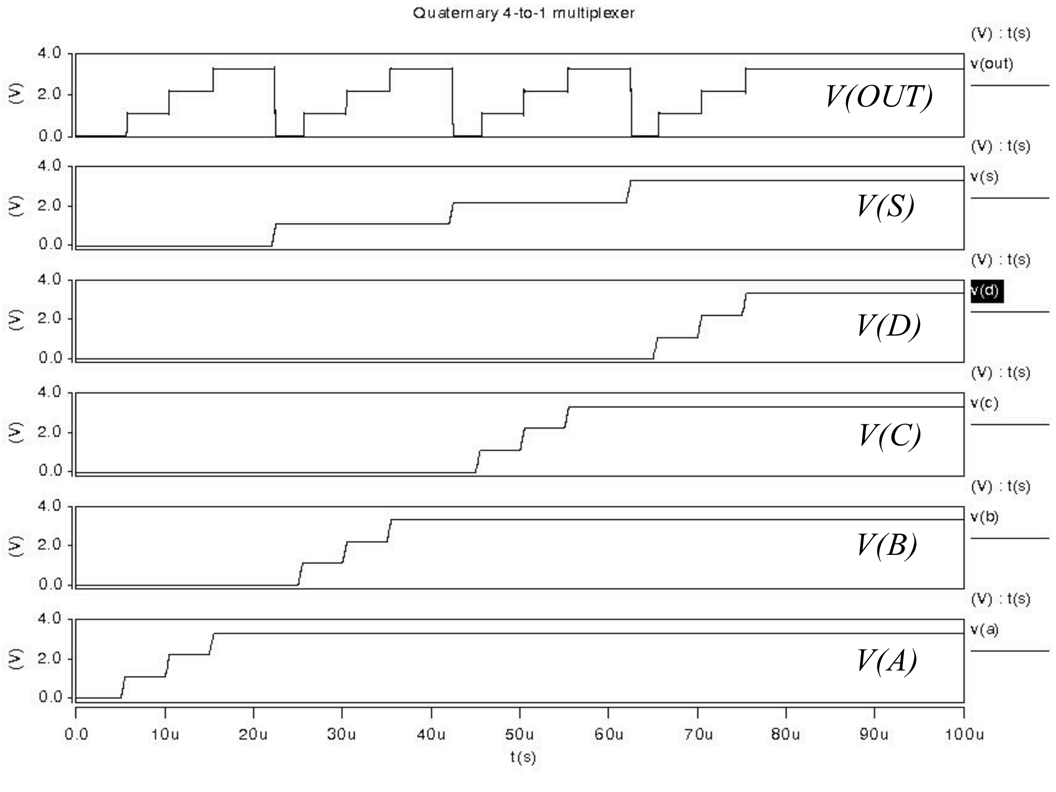


**Figure 22: Quaternary 4-to-1 Multiplexer Circuit & Spice Simulation Result**

# Chapter 4

# QUATERNARY ARITHMETIC CIRCUIT ARCHITECTURES

The focus of this work is to design the quaternary arithmetic circuits for primitive addition (+) and multiplication (*) operations using circuits from the cell library described in Chapter 3. Several architectural options are available for implementing addition and multiplication arithmetic circuits. Different adder and multiplier architectures are analyzed by implementing the quaternary circuits whose details are described in this chapter.

## 4.1. Quaternary Adder Circuit Architectures

An adder is one of the most common arithmetic circuits and also serves as a building block for realizing many other arithmetic operations. Single-digit half- and full-adders are versatile building blocks that are used in larger adders and many other types of arithmetic circuits. Three different adder circuit architectures are implemented using half- and full-adder circuits as basic building blocks [DT+09]. These are the Ripple Carry (RC), Carry look ahead (CLA), and Carry Select (CS) addition circuits.

### 4.1.1. Quaternary Half-adder

A binary half-adder (HA) receives two input bits, *x* and *y*, producing a sum output bit $s = x \oplus y$ and a carryout bit $c = xy$. In the case of quaternary logic, the half-adder circuit receives two input quaternary digits and produces quaternary sum and carryout digits. A half-adder can be viewed as a single-digit quaternary adder that produces the two-digit sum of its single-digit inputs, namely, $x + y = (c_{out}s_{out})_4$. A truth table for the quaternary half-adder circuit is shown in Table 3.

**Table 3: Quaternary Half-adder Truth Table**

| $x$ | $y$ | Outputs | |
|---|---|---|---|
| | | $c_{out}$ | $s_{out}$ |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 2 | 0 | 2 |
| 0 | 3 | 0 | 3 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 2 |
| 1 | 2 | 0 | 3 |
| 1 | 3 | 1 | 0 |
| 2 | 0 | 0 | 2 |
| 2 | 1 | 0 | 3 |
| 2 | 2 | 1 | 0 |
| 2 | 3 | 1 | 1 |
| 3 | 0 | 0 | 3 |
| 3 | 1 | 1 | 0 |
| 3 | 2 | 1 | 1 |
| 3 | 3 | 1 | 2 |

The equations used to represent the sum ($s_{out}$) and carry ($c_{out}$) outputs of the half-adder circuit are shown in Figure 23 in sum-of-minterms (SOM) form.

$$s_{out} = 1 \bullet [A^{\{0\}} \bullet B^{\{1\}} + A^{\{1\}} \bullet B^{\{0\}} + A^{\{2\}} \bullet B^{\{3\}} + A^{\{3\}} \bullet B^{\{2\}}]$$
$$+ 2 \bullet [A^{\{0\}} \bullet B^{\{2\}} + A^{\{1\}} \bullet B^{\{1\}} + A^{\{2\}} \bullet B^{\{0\}} + A^{\{3\}} \bullet B^{\{3\}}]$$
$$+ 3 \bullet [A^{\{0\}} \bullet B^{\{3\}} + A^{\{1\}} \bullet B^{\{2\}} + A^{\{2\}} \bullet B^{\{1\}} + A^{\{3\}} \bullet B^{\{0\}}]$$

$$c_{out} = 1 \bullet [\mathrm{A}^{\{0\}}\mathrm{B}^{\{3\}} + \mathrm{A}^{\{1\}} \bullet \mathrm{B}^{\{3\}} +$$
$$A^{\{2\}} \bullet B^{\{2,3\}} + A^{\{3\}} \bullet B^{\{1,2,3\}}]$$

**Figure 23: Quaternary Half-adder Equations**

Cells from the quaternary library are used to implement the half-adder circuit structure. The equations are optimized using algebraic manipulations and one-hot encoding, including optimizing SOM sets [BK99]. WE also used 4-to-1 quaternary multiplexers to realize the half-adder function, as shown in Figure 24.

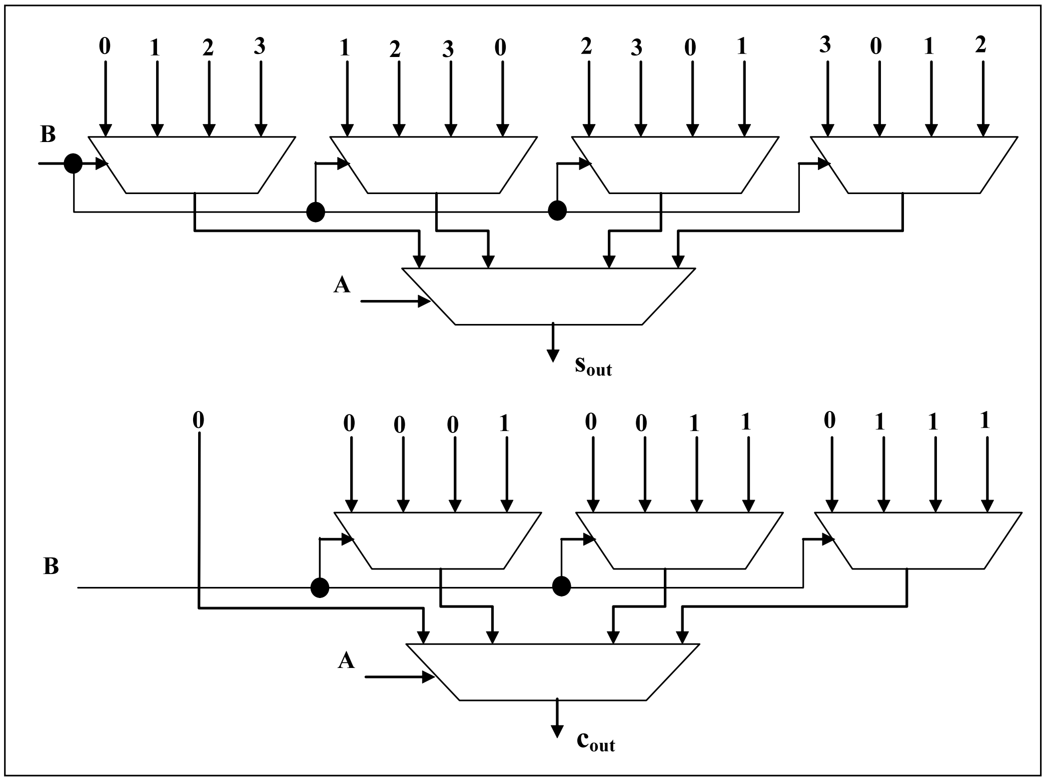


**Figure 24: Quaternary Half-adder Based on Multiplexer Implementation**

The advantage of multiplexer based implementation is that it is simple and straightforward. However, a multiplexer based circuit architecture may suffer the disadvantage of being larger in area (number of transistors) than an implementation architecture that uses the quaternary library gates. Moreover, the performance and dynamic power may be also be worse due to non-optimized circuit design. The quaternary half-adder circuit that is shown here is one of the fundamental building blocks of complex adder and multiplier circuits.

### 4.1.2. Quaternary Pseudo Full-adder

The quaternary pseudo full-adder is an extension to the quaternary half-adder circuit. It includes an additional input called the carry-in, referred to as $c_{in}$. This circuit adds two quaternary digits and also the quaternary $c_{in}$ digit to generate quaternary the sum and carry outputs. The circuit is referred to as "pseudo" because the $c_{in}$ pin is assumed to only have values of "0" and "1". This assumption simplifies the circuit's implementation and still results in a useful circuit for implementing quaternary adders and multipliers. Table 4 depicts the truth table of the pseudo full-adder.

**Table 4: Quaternary Pseudo Full-adder Truth Table**

| | | $c_{in}= 0$ | | $c_{in} = 1$ | |
|---|---|---|---|---|---|
| $x$ | $y$ | $c_{out}$ | $s_{out}$ | $c_{out}$ | $s_{out}$ |
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 2 |
| 0 | 2 | 0 | 2 | 0 | 3 |
| 0 | 3 | 0 | 3 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 2 |
| 1 | 1 | 0 | 2 | 0 | 3 |
| 1 | 2 | 0 | 3 | 1 | 0 |
| 1 | 3 | 1 | 0 | 1 | 1 |
| 2 | 0 | 0 | 2 | 0 | 3 |
| 2 | 1 | 0 | 3 | 1 | 0 |
| 2 | 2 | 1 | 0 | 1 | 1 |
| 2 | 3 | 1 | 1 | 1 | 2 |
| 3 | 0 | 0 | 3 | 1 | 0 |
| 3 | 1 | 1 | 0 | 1 | 1 |
| 3 | 2 | 1 | 1 | 1 | 2 |
| 3 | 3 | 1 | 2 | 1 | 3 |

When pseudo full-adders are configured as multi-digit operand adder circuits, the output signal values "0" and "1" are the only possible values for the carry output. The equations for the sum and carry outputs of the quaternary pseudo full-adder circuit are shown in Figure 25. As expected, the carryout function is optimized due to the limited range of output values. Further optimization of the circuit implementation could be possible by extending the quaternary library to include more two- or three-input circuit cells.

$$
\begin{aligned}
s_{out} &= 1 \bullet [A^{\{0\}} \bullet (B^{\{1\}}C^{\{0\}} + B^{\{0\}}C^{\{1\}}) + \\
&A^{\{1\}} \bullet (B^{\{0\}}C^{\{0\}} + B^{\{3\}}C^{\{1\}}) + \\
&A^{\{2\}} \bullet (B^{\{3\}}C^{\{0\}} + B^{\{2\}}C^{\{1\}}) + \\
&A^{\{3\}} \bullet (B^{\{2\}}C^{\{0\}} + B^{\{1\}}C^{\{1\}})] \\
&+ 2 \bullet [A^{\{0\}} \bullet (B^{\{2\}}C^{\{0\}} + B^{\{1\}}C^{\{1\}}) + \\
&A^{\{1\}} \bullet (B^{\{1\}}C^{\{0\}} + B^{\{0\}}C^{\{1\}}) + \\
&A^{\{2\}} \bullet (B^{\{0\}}C^{\{0\}} + B^{\{3\}}C^{\{1\}}) + \\
&A^{\{3\}} \bullet (B^{\{3\}}C^{\{0\}} + B^{\{2\}}C^{\{1\}})] \\
&+ 3 \bullet [A^{\{0\}} \bullet (B^{\{3\}}C^{\{0\}} + B^{\{2\}}C^{\{1\}}) + \\
&A^{\{1\}} \bullet (B^{\{2\}}C^{\{0\}} + B^{\{1\}}C^{\{1\}}) + \\
&A^{\{2\}} \bullet (B^{\{1\}}C^{\{0\}} + B^{\{0\}}C^{\{2\}}) + \\
&A^{\{3\}} \bullet (B^{\{0\}}C^{\{0\}} + B^{\{3\}}C^{\{1\}})]
\end{aligned}
$$

$$
\begin{aligned}
c_{out} &= 1 \bullet [\mathrm{A}^{\{0\}}\mathrm{B}^{\{3\}}\mathrm{C}^{\{1\}} + \\
&\mathrm{A}^{\{1\}} \bullet (\mathrm{B}^{\{3\}}\mathrm{C}^{\{0\}} + \mathrm{B}^{\{2,3\}}\mathrm{C}^{\{1\}}) + \\
&A^{\{2\}} \bullet (B^{\{2,3\}}C^{\{0\}} + B^{\{1,2,3\}}C^{\{1\}}) + \\
&A^{\{3\}} \bullet (B^{\{1,2,3\}}C^{\{0\}} + C^{\{1\}})]
\end{aligned}
$$

**Figure 25: Quaternary Pseudo Full-adder Equations**

### 4.1.3. Quaternary Full-adder

The quaternary full-adder circuit includes the complete full-adder functionality supporting all possible logic values for $c_{in}$. The full-adder circuit is often referred to as a 3-to-2 compressor circuit, since the circuit compresses three quaternary inputs into two quaternary outputs [CIL06, WT89, SH08].

**Table 5: Quaternary Full-adder Truth Table**

| | | $c_{in}$=0 | | $c_{in}$=1 | | $c_{in}$=2 | | $c_{in}$=3 | |
|---|---|---|---|---|---|---|---|---|---|
| $x$ | $y$ | $c_{out}$ | $s_{out}$ | $c_{out}$ | $s_{out}$ | $c_{out}$ | $s_{out}$ | $c_{out}$ | $s_{out}$ |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 2 | 0 | 3 |
| 0 | 1 | 0 | 1 | 0 | 2 | 0 | 3 | 1 | 0 |
| 0 | 2 | 0 | 2 | 0 | 3 | 1 | 0 | 1 | 1 |
| 0 | 3 | 0 | 3 | 1 | 0 | 1 | 1 | 1 | 2 |
| 1 | 0 | 0 | 1 | 0 | 2 | 0 | 3 | 1 | 0 |
| 1 | 1 | 0 | 2 | 0 | 3 | 1 | 0 | 1 | 1 |
| 1 | 2 | 0 | 3 | 1 | 0 | 1 | 1 | 1 | 2 |
| 1 | 3 | 1 | 0 | 1 | 1 | 1 | 2 | 1 | 3 |
| 2 | 0 | 0 | 2 | 0 | 3 | 1 | 0 | 1 | 1 |
| 2 | 1 | 0 | 3 | 1 | 0 | 1 | 1 | 1 | 2 |
| 2 | 2 | 1 | 0 | 1 | 1 | 1 | 2 | 1 | 3 |
| 2 | 3 | 1 | 1 | 1 | 2 | 1 | 3 | 2 | 0 |
| 3 | 0 | 0 | 3 | 1 | 0 | 1 | 1 | 1 | 2 |
| 3 | 1 | 1 | 0 | 1 | 1 | 1 | 2 | 1 | 3 |
| 3 | 2 | 1 | 1 | 1 | 2 | 1 | 3 | 2 | 0 |
| 3 | 3 | 1 | 2 | 1 | 3 | 2 | 0 | 2 | 1 |

As shown in Table 5, all four possible values of $c_{in}$ are supported. This form of adder is necessary for addition circuits that support multiple operand words summation. For two-operand adders, the pseudo full adder is sufficient since the only possible carryout values that can occur are {0,1}. The circuit in Figure 26 illustrates the multiplexer based implementation of the quaternary full-adder circuit and it consists of three logic levels for both outputs.

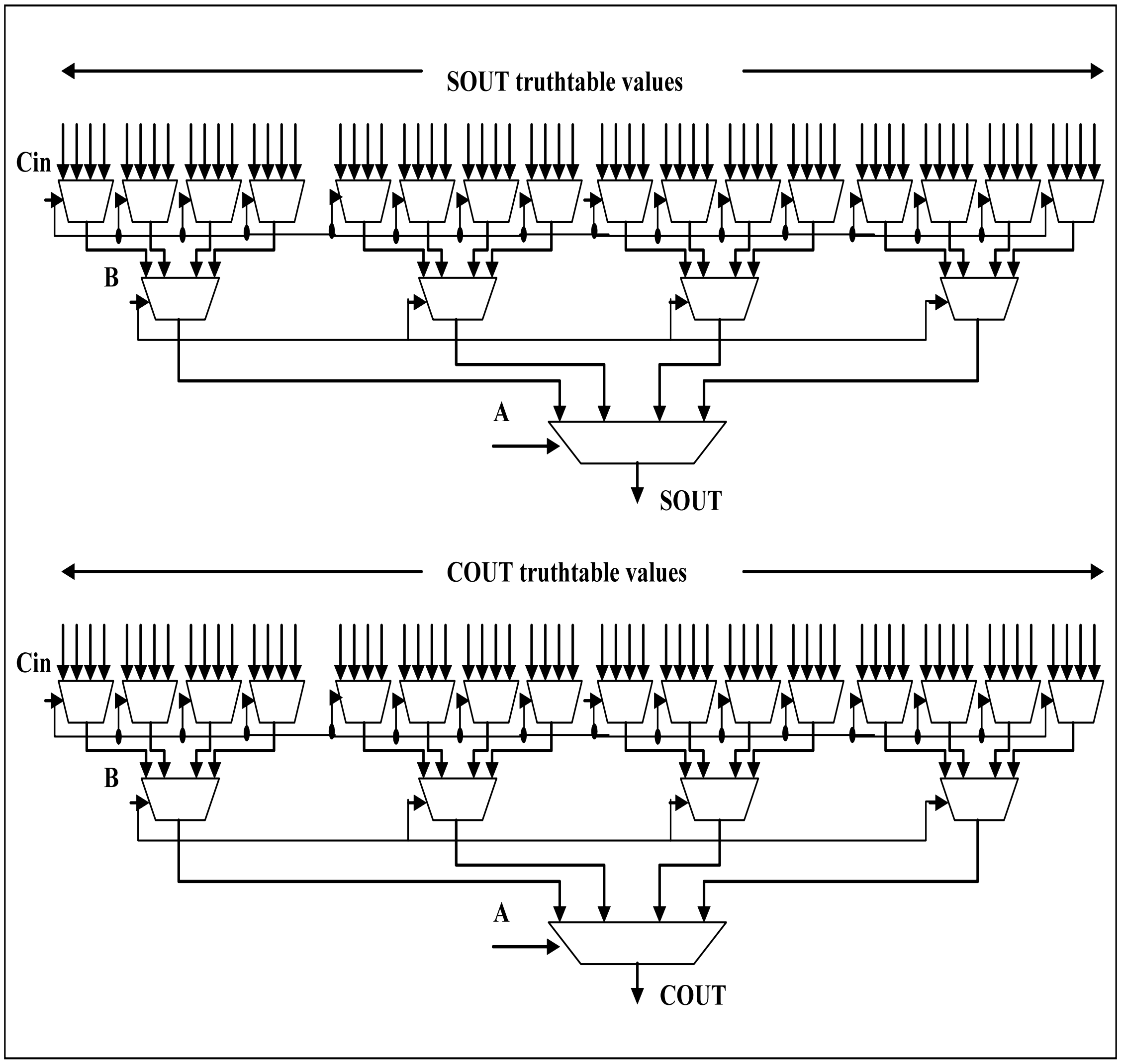


**Figure 26: Multiplexer Based Implementation of Quaternary Full-adder Circuit**

The sum and carry output circuits each contain 21 four-to-one quaternary multiplexers. As is the case with a half-adder circuit, the full-adder circuit can also be implemented using the quaternary *MAX*, *MIN*, and other one-variable functions

4.1.4. Quaternary Ripple-Carry Adder

The quaternary half-, pseudo full-, and full-adder circuits are used to design arithmetic circuit types such as adders and multipliers with multi-digit operands.

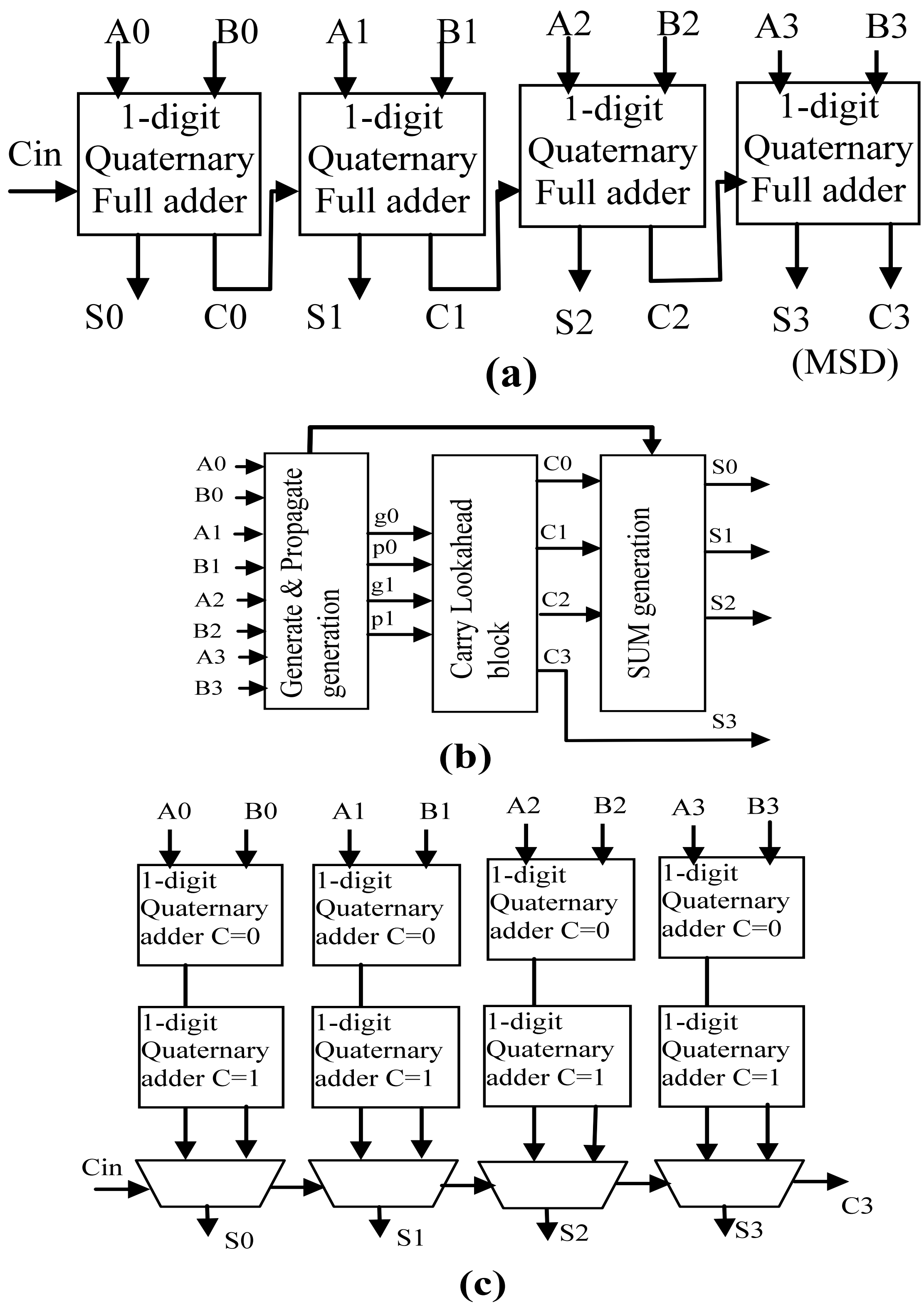

A0
B0
A1
B1
A2
B2
A3
B3
Cin
1-digit Quaternary Full adder
S0
C0
S1
C1
S2
C2
S3
C3
(MSD)
(a)
Generate & Propagate generation
g0
p0
g1
p1
Carry Lookahead block
SUM generation
(b)
1-digit Quaternary adder C=0
1-digit Quaternary adder C=1
(c)

**Figure 27: Four-digit Quaternary Adder Architectures**

Three different multi-digit adder architectures are implemented: the Ripple Carry (RC), Carry Look-Ahead (CLA), and Carry Select (CS) versions. Figure 27 shows all three adder architectures with a four-digit word size example. The ripple-carry adder is constructed using a serial cascade of one-digit full-adder circuits and is depicted in Figure 27a.

The quaternary RC adder is simple and relatively easy to implement as compared to the CLA and CS architectures. It is also the most area efficient when compared to the other two types. The well known disadvantage of the RC adder is its limited performance (speed of operation). The decrease in this circuit's speed of operation is caused by the serial propagation of the carry digit. As depicted in Figure 27a, the Most Significant Digit (MSD) result depends on the carryout digit from the previous stages. Essentially, the summation output for each digit is delayed until the carry out from the previous digit is available. This results in a speed of operation that is directly proportional to the operand word size.

### 4.1.5. Quaternary Carry Look-Ahead Adder

The CLA architecture is a widely used adder architecture that achieves high performance by utilizing the so-called carry generate and propagate logic functions. A quaternary carry look-ahead adder is proposed in [TS+01] and is implemented here using the newly developed SUSLOC cell library. A 32-digit quaternary carry look-ahead adder was designed using the SUSLOC gates. The overall structure of the addition circuit is illustrated with a four-digit example in Figure 27b. The "generate" and "propagate" circuit blocks produce the required $g0$, $g1$, $p0$, and $p1$ signals. The equations for the quaternary case are shown in Figure 28. Note that the quaternary carry-out output of the two-digit adders can only have “0” or “1” as a value, for example, $3_4 + 3_4 = 12_4$. Hence, the pseudo full-adder circuit is employed in the carry-out path to conserve circuit area.

$$g0 = A^{\{0\}} + B^{\{0\}} + A^{\{1\}}B^{\{1,2\}} + A^{\{2\}}B^{\{1\}}$$

$$g1 = A^{\{1\}}B^{\{3\}} + A^{\{2\}}(B^{\{2\}} + B^{\{3\}}) + A^{\{3\}}B^{\{1\}}$$

$$p0 = g0$$

$$p1 = A^{\{0\}}B^{\{3\}} + A^{\{1\}}B^{\{2\}} + A^{\{2\}}B^{\{1\}} + A^{\{3\}}B^{\{0\}}$$

**Figure 28: Equations for the Generate & Propagate Signals of Radix-4 Adder Digits**

The 32-digit adder circuit is implemented with the necessary carry look-ahead logic resulting in increased area as compared to the RC adder due to the carry generation and propagation logic. Alternatively, the CLA adder has a greater speed of operation as compared with the RC adder.

### 4.1.6. Quaternary Carry Select Adder

The carry select adder is another well known adder circuit architecture that deals with the carry propagation challenge in a different way. The addition operation is duplicated with the assumption of different input carry possibilities, and the correct adder output is finally selected based on the actual carry value. A four-digit example of the pseudo quaternary carry select adder is shown in Figure 27c. In this case, only values of “0” and “1” are assumed as possibilities at $c_{in}$. The need for redundant adders in this adder architecture would result in an area increase. However, the performance of the CS adder is improved because the addition operation outputs with different possible carry values are readily available and can be selected when the actual carry value is available. This feature significantly reduces the carry propagation time. The circuit area could be further optimized by replacing the full-adders with equivalent custom half-adders due to the presence of the constant values at the carry input.

## 4.2. Quaternary Multiplier Circuit Architectures

The multiplier is another important arithmetic circuit that is used in almost every modern application and algorithm [B51, R55, R75, D65, W64, E90, CC94, CC95, IO+97]. Two types of quaternary multiplier circuits are designed, a digit serial and parallel version. In the architecture of the serial multiplier, the multiplication output is obtained by performing multiplication and addition in an iterative manner. The net result is obtained after *n* clock cycles, where *n* is the width of the multiplicand or multiplier, whichever is larger. On the other hand, in the case of a parallel multiplier architecture, the multiplication output is generated in a single clock cycle [D65,W64].

The advantage of a serial multiplier is its smaller area and smaller power dissipation compared to a parallel multiplier. The advantage of a parallel multiplier is its speed of operation because the whole multiplication operation is complete in one cycle.

### 4.2.1. Quaternary Serial Multiplier Circuit

The quaternary serial multiplier circuit architecture is depicted in the block diagram of Figure 29. The architecture used in this circuit is a sequential "*multiply, accumulate, and add*" architecture.

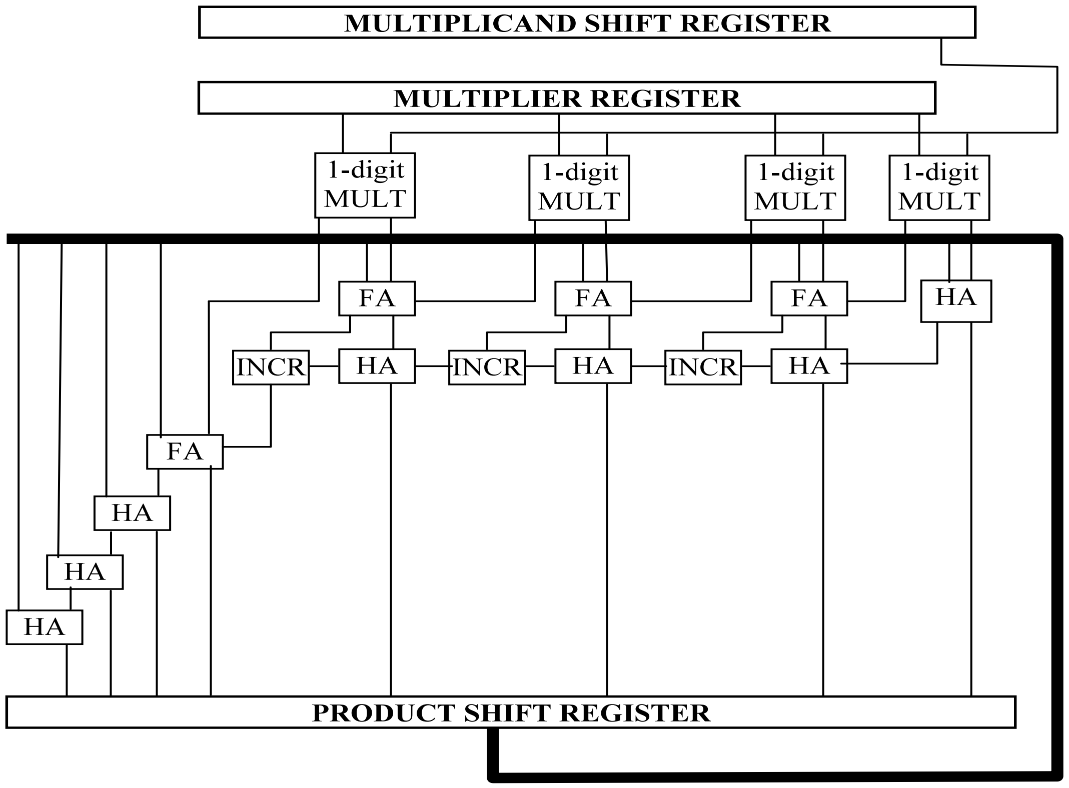


**Figure 29: Quaternary Sequential Multiplier Block Diagram**

In each clock cycle, one digit of the multiplier operand is used to multiply the entire multiplicand value by using several single-digit quaternary multiplier cells. The result is then added to the accumulated partial product. The partial product is stored in a quaternary register after every clock cycle. This quaternary register also acts as a shift register to shift the partial product value by two-bits (One-quaternary digit) in every clock cycle. Therefore, after *n* clock cycles of "*multiply, add, and shift*" operations, the net multiplication result is available. There are only two sets of quaternary quantities added at a time. Because there are only two quantities to be added each time, half-adder and incrementer circuits can be used in the adder architecture to reduce the total area of the multiplier instead of using the more costly full-adder. The two important elements used in the serial multiplier's architecture are a single-digit quaternary multiplier circuit and a quaternary incrementer circuit.

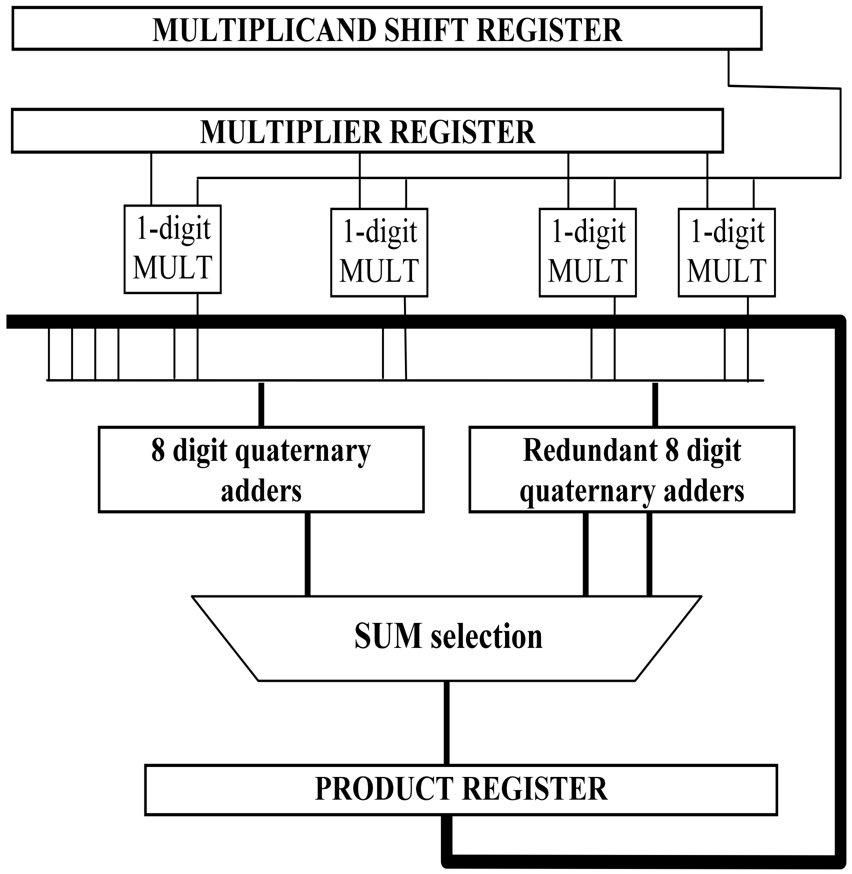


**Figure 30: Quaternary Serial Multiplier using Carry-Select Adders**

The serial multiplier shown in Figure 29 is using the serial ripple carry type of adder architecture for adding the intermediate partial products in each clock cycle. The two other adder architectures, Carry-Select and Carry Look-Ahead adders are also used in the serial multiplier architecture as shown in Figure 30 and Figure 31. The comparison metrics of area, performance and power of serial multipliers are all benchmarked for the multipliers with all three different adder architectures embedded.

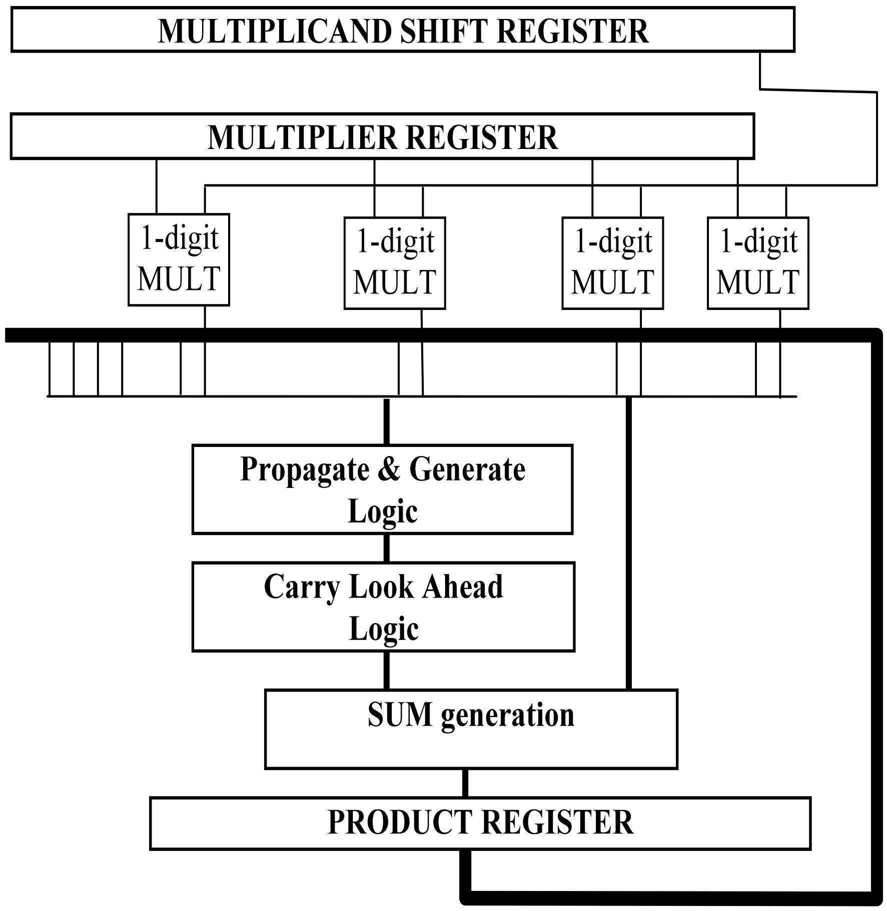


**Figure 31: Quaternary Serial Multiplier using Carry-Look-Ahead (CLA) Adders**

4.2.2. Quaternary Single-digit Multiplier Circuit

The quaternary single-digit multiplier circuit uses two single-digit quaternary inputs and produces the two-digit product output. The quaternary single-digit multiplier is implemented using the *MIN*, *MAX*, and *r*-*x*-1 inverter gates. The truth table for the quaternary single-digit multiplier circuit is provided in Table 6.

**Table 6: Quaternary Single-digit Multiplier Truth Table**

| *x* | *y* | **Single-digit quaternary Multiplier** | |
|---|---|---|---|
| | | MSD | LSD |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 2 | 0 | 0 |
| 0 | 3 | 0 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 2 | 0 | 2 |
| 1 | 3 | 0 | 3 |
| 2 | 0 | 0 | 0 |
| 2 | 1 | 0 | 2 |
| 2 | 2 | 1 | 0 |
| 2 | 3 | 1 | 2 |
| 3 | 0 | 0 | 0 |
| 3 | 1 | 0 | 3 |
| 3 | 2 | 1 | 2 |
| 3 | 3 | 2 | 1 |

The algebraic equations used to describe the MSD and LSD digits of the quaternary single-digit multiplier circuit are shown in Figure 32a. A multiplexer based implementation of the quaternary single-digit multiplier is shown in Figure 32b.

$$MSD = 1 \bullet [A^{\{2\}} \bullet (B^{\{2,3\}}) + A^{\{2\}} \bullet (B^{\{2\}})] +$$
$$2 \bullet [A^{\{3\}} \bullet B^{\{3\}}]$$

$$LSD = 1 \bullet [(A^{\{1\}} \bullet B^{\{1\}}) + (A^{\{3\}} \bullet B^{\{3\}})] +$$
$$2 \bullet [([A^{\{1\}} \bullet B^{\{2\}}) + (A^{\{2\}} \bullet B^{\{1,3\}}) + (A^{\{3\}} \bullet B^{\{2\}})] +$$
$$3 \bullet [A^{\{1\}} B^{\{3\}} + A^{\{3\}} B^{\{1\}}]$$

**(a)**

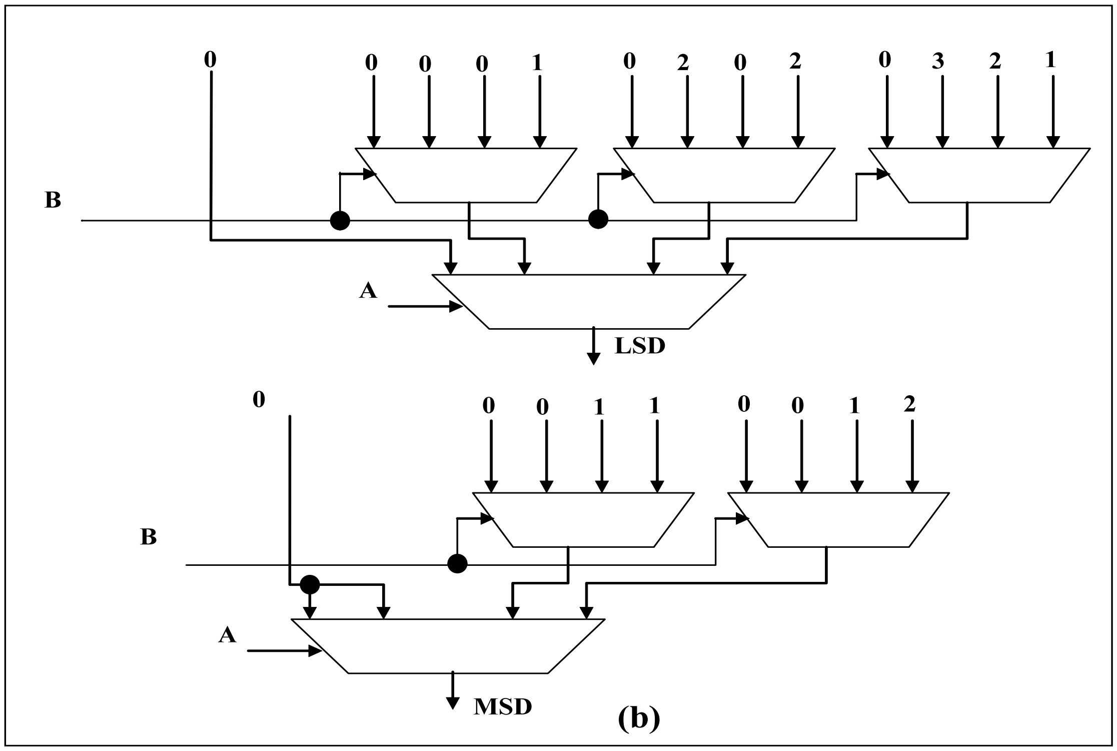


**Figure 32: One-digit Multiplier (a) Equations (b) Multiplexer Based Implementation**

### 4.2.3. Quaternary Incrementer Circuit

In the serial multiplier architecture, the outputs of the single-digit quaternary multipliers are added to the $c_{out}$ outputs of the half-adder circuits. As seen in the quaternary single-digit multiplier truth table, values of "0", "1", and "2" are the only possible logic values for the MSD output. It is also the case that $c_{out}$ can only have values of "0" or "1" at the output of the quaternary half-adder circuit. These observations allow the creation of an area optimized quaternary increment circuit whose truth table is shown in Table 7. Implementation equations corresponding to Table 7 are shown in Figure 33. The circuit is implemented using the cells from the quaternary library. The main difference between the incrementer circuit and the half-adder circuit is that there is no carryout generated in the case of the increment circuit.

**Table 7: Quaternary Incrementer Truth Table**

| *A* | *B* | *INCROUT* |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 2 |
| 2 | 0 | 2 |
| 2 | 1 | 3 |

$$\begin{aligned} INCROUT = {} & 1 \bullet [A^{\{0\}} \bullet B^{\{1\}} + A^{\{1\}} \bullet B^{\{0\}}] \\ & + 2 \bullet [A^{\{1\}} \bullet B^{\{1\}} + A^{\{2\}} \bullet B^{\{0\}}] + 3 \bullet [A^{\{2\}} \bullet B^{\{1\}}] \end{aligned}$$

**Figure 33: Quaternary Incrementer Equations**

### 4.2.4. Quaternary Parallel Multiplier

The entire multiplication operation is performed in a single clock cycle [P00, EL04, D65, W64, SH08] in the parallel multiplication circuit. Single cycle operation is achieved since all partial products are generated simultaneously followed by summing them together with a multi-operand adder. Figure 34 contains details of the partial product generator designed using an array of single digit quaternary multipliers where each partial product is generated in carry-save format consisting of a four-digit carry word and a separate four-digit sum word.

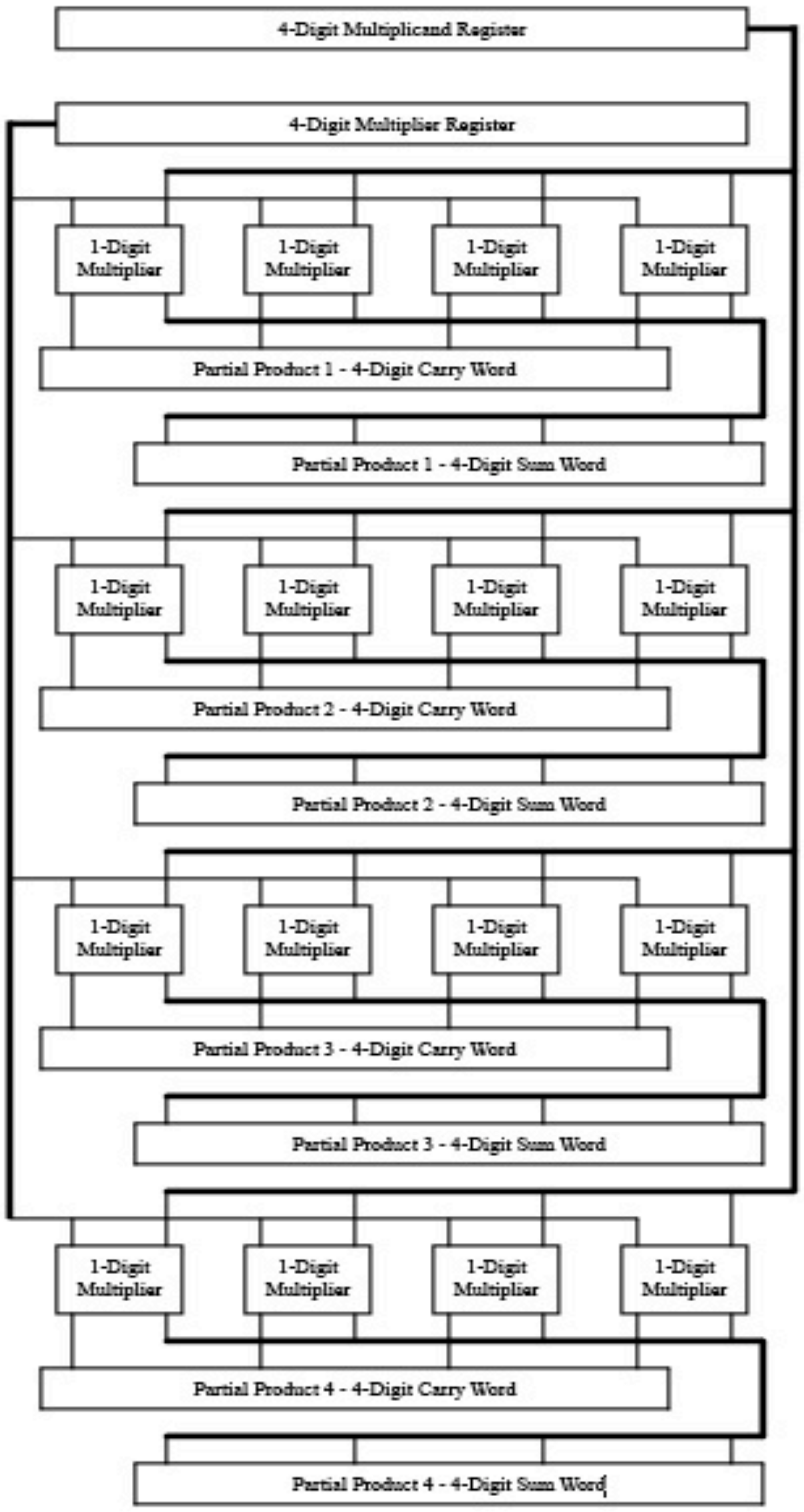

4-Digit Multiplicand Register
4-Digit Multiplier Register
1-Digit Multiplier
Partial Product 1 - 4-Digit Carry Word
Partial Product 1 - 4-Digit Sum Word
Partial Product 2 - 4-Digit Carry Word
Partial Product 2 - 4-Digit Sum Word
Partial Product 3 - 4-Digit Carry Word
Partial Product 3 - 4-Digit Sum Word
Partial Product 4 - 4-Digit Carry Word
Partial Product 4 - 4-Digit Sum Word

**Figure 34: Diagram of 4*4 Partial Product Generator**

Figure 35 contains a diagram of the circuit used to accumulate all partial products to form the final eight-digit product word. Quaternary half-adders and full-adders are used appropriately to reduce and optimize the area of the multi-operand addition tree used to sum the partial products. The form of the array of full-adders (FA) and half-adders (HA) in Figure 35 is described in [P71] and the internal configuration of the FAs and HAs is described in [DT+09]. Inputs to the addition array are denoted by *n*[*s*,*c*]*i* where *i* represents the *i*th digit of the *n*th partial product and [*s*,*c*] indicates whether the digit is from the carry or sum word of each partial product.

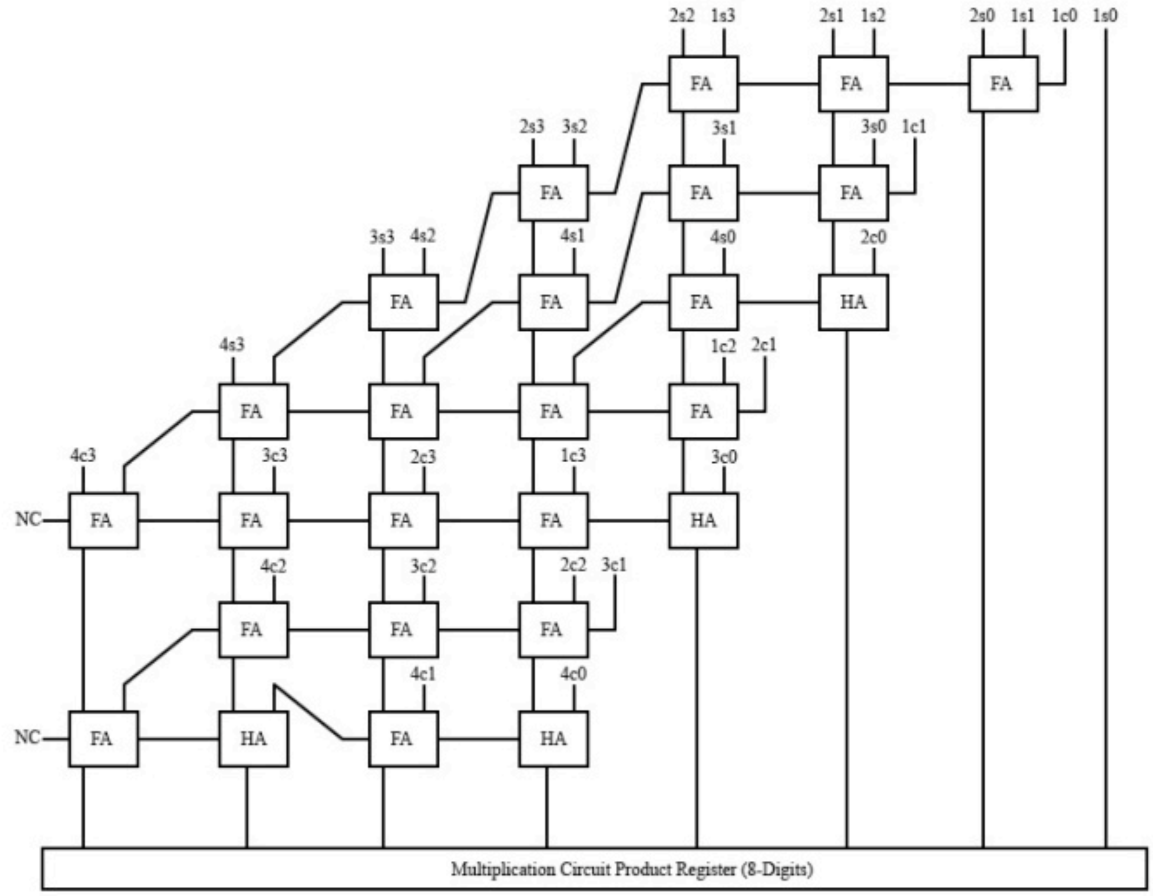


**Figure 35: Diagram of 4*4 Partial Product Accumulation Array**

The parallelism in the circuit increases the area of the multiplier circuit. At the same time, the benefit of parallelism is that the entire multiplication operation is completed within one clock cycle. The parallel multiplier does provide higher efficiency when used in the context of large multiply operations because each multiplication operation is finished in a single clock cycle. However, its higher area is the main disadvantage of the parallel multiplier when compared to the serial multiplier.

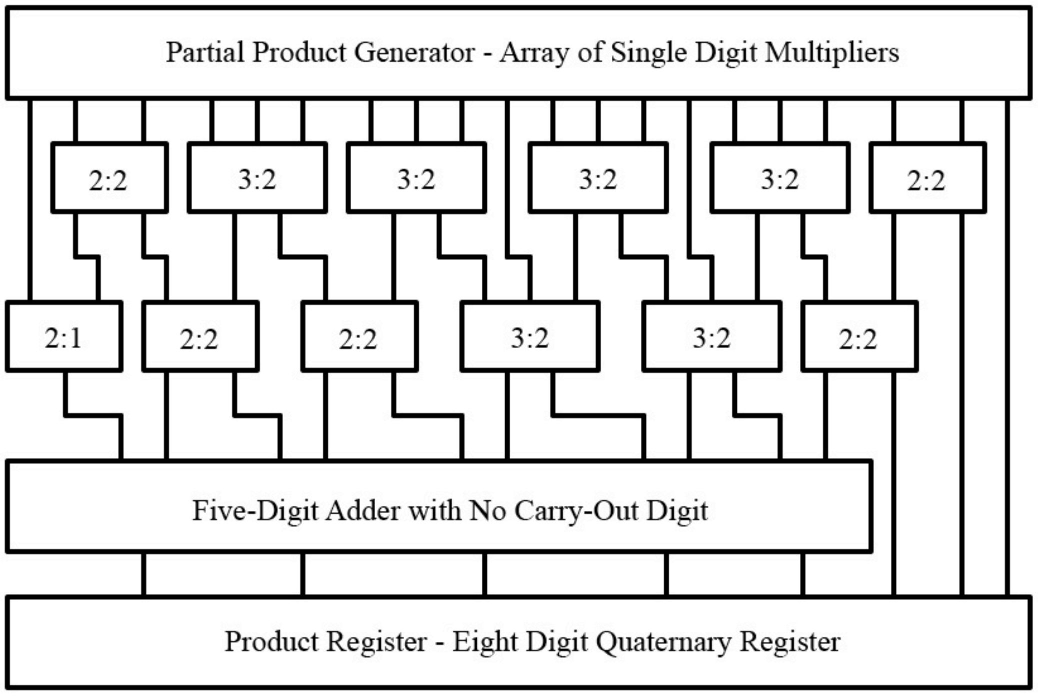

Partial Product Generator - Array of Single Digit Multipliers
2:2
3:2
3:2
3:2
3:2
2:2
2:1
2:2
2:2
3:2
3:2
2:2
Five-Digit Adder with No Carry-Out Digit
Product Register - Eight Digit Quaternary Register

**Figure 36: 4*4 Parallel Multiplier with Wallace-Tree Addition**

There are alternate ways to implement the addition operation of the generated partial products. One of the main disadvantages of the array type of adder architecture is its long propagation delays. We implemented the Wallace-Tree [W64] adder architecture for a 4-digit quaternary parallel multiplier as shown in Figure 36 to compare the differences in area, timing and power of both architectures.

# Chapter 5

# APPROXIMATE SQUARING CIRCUIT

Approximate squaring circuits have numerous applications as mentioned in [FW01, KG98, PA96, YK+97, WS+99] such as cryptography, computation of Euclidean distance among pixels for a graphics processor or in rectangular to polar conversions in several signal processing circuits where full precision results are not required. As indicated in [PA96, LF00], customized squaring modules do have important applications in digital signal processing. Specifically, in [AB+99], a method is described where resolution can be increased during a graphics blend operation through the incorporation of a squaring operation implemented by a multiplier followed by a truncation circuit. Clearly, the approach described in this work allows for improvement in such application. In [JM95], a method for frame synchronization in a digital radio is described where a digital squaring circuit is integral to the process. Hardware transcendental function designs [F81] have also employed approximate squaring circuits. These are just a few examples where high-performance approximate squaring circuits are desirable.

Often, designers implement a squaring operation using a multiplier circuit. Approximate multiplication has been investigated using a truncated multiplier [WS+01]. The multiplier may utilize a radix-4 or radix-8 Booth recoding to reduce the size of the partial product array [P00, K81, EL04]. The squaring operation yields symmetry in the partial product array when compared to a standard multiplier. This property has been investigated to provide optimizations in multiplier design at the bit level [L70, C71]. The design focusing on a squaring circuit employing this symmetry was proposed in [D85], and numerous studies optimizing binary squaring circuits appear in [FW01, KG98, PA96, YK+97, WS+99]. These designs primarily optimized by using hardwired bit product arrangements to reduce array sizes for efficient accumulation, mostly focusing on low precision. Since squaring is a unary operation, lookup tables have also been incorporated in proposed designs of squaring circuits [WS+01, WS98]. Extension to the design of a radix-4 squaring circuit employing Booth recoding and "folding" of the partial products was introduced in [DS01], with further implementation optimization studies discussed in [E03, SD03].

Booth recoded multipliers yield partial products whose formation requires the complexities of both sign extension and two's complementation. The Booth-folded recoded squarer in [DS01] reduces two's complementation to a straightforward one's complementation. Avoidance of two's complementation particularly simplifies the generation of integer squares modulo the integer word size, as further investigated in [MM+08]. In this work, we investigate implementations of a new radix-4 operand dual recoding method [M09] for the squaring operation. The recoding yields non-negative partial squares avoiding need for sign extensions and furthermore yields a radix-16 reduced array of partial products. This recoding is particularly effective for the design of an approximate squaring circuit where the partial squares can be generated with shifts and one's complements using a few guard bits.

## 5.1. Squaring Methodology

As introduced in [C71, D85], the squaring operation for an operand in binary can be realized by a modified partial product array of about one half the size of the full multiplier array without the need for the traditional Booth multiplier recoding. Specifically, for binary squaring of the normalized $p$-bit operand $x = 01.b_1 b_2 \ldots b_{p-1}$, the $p^2$ bit product terms of the multiplication array may be reduced to $p(p+1)/2$ terms employing $b_i^2 = b_i$ and $b_i b_j + b_j b_i = 2b_i b_j$ for $i<j$. This provides an array of just over half the size with a depth $\lceil (p+1)/2 \rceil$. Furthermore, the depth of the array can be reduced to $\lceil (p/2) \rceil$ by incorporating the half adder relation $b_i b_{i+1} + b_i = 2b_i b_{i+1} + b_i(\sim b_{i+1})$ in the array formation.

For a state-of-the-art approximate radix-2 squarer we employ this reduced depth array design from [PBD97] and truncate the lower order half of the array, except for a couple of columns of guard bits to tightly bound the approximate square. This optimized approximate radix-2 squarer array is of order about 1/4th the size of a comparable full multiplier array, or equivalently about ½ the size of the truncated approximate multiplier.

Recently [M09] an operand "dual recoded" radix-4 squaring method has been introduced which is particularly suited for approximate squaring. We adopt the method from [M09] and perform various implementation studies. Before proceeding to performance comparison of this high-radix squarer, we summarize the methodology and foundation for the new dual recoded radix-4 squarer.

For the squaring operation, the single operand assumes both the role of the multiplier and multiplicand. The high radix dual recoding recognizes these distinct asymmetric roles of the single operand. The dual recoding concurrently provides a "squarer" digit string in the high radix and a corresponding sequence of successively truncated "squarands" in binary form. The $i^{th}$ squarer digit multiplies the $i^{th}$ squarand in the $i^{th}$ partial square generator, with the array of partial squares summed to generate the square. The following summary is taken from [M09].

For the left-to-right leading digit dual recoding, the $i^{th}$ squarand is determined only from bits of lesser or equal significance to the bits determining the $i^{th}$ high radix squarer digit. The catalyst for characterizing the left-to-right higher radix dual recoding is the sequence of two's complement tails of the operand.

*__Definition:__* Given the $p$-bit normalized operand $x = 01.b_1 b_2 \ldots b_{p-1}$, the radix-4 two's complement tails of $x$ are $t_0 = x = 1.b_1 b_2 \ldots b_{p-1}$ and $t_i = (\bar{b}_{2i-1} b_{2i} . b_{2i+1} \ldots b_{p-1})$ for $1 \le i \le \frac{p+1}{2}$, where $\bar{b}_{2i-1} = -b_{2i-1}$.

The two's complement tails are related to the Booth radix-4 representation.

*__Observation 1:__*

$$x = \sum_{i=0}^{\left\lfloor \frac{p+1}{2} \right\rfloor} (t_i - t_{i+1} \bullet 4^{-1}) 4^{-i} = \sum_{i=0}^{\left\lfloor \frac{p+1}{2} \right\rfloor} d_i 4^{-i}$$

where $di = (t_i - t_{i+1} \bullet 4^{-1})$ is the $i^{th}$ Booth radix-4 digit of $x$.

*__Proof:__* Note that: $t_i - t_{i+1} \bullet 4^{-1} = (\bar{b}_{2i-1}\, b_{2i} . b_{2i+1}) - (0.\bar{b}_{2i+1})$ so

$t_i - t_{i+1} \bullet 4^{-1} = -2\, b_{2i+1} + b_{2i} + 2 b_{2i+1} \bullet 2^{-1} = d_i$, and $d_i \in \{-2, -1, 0, 1, 2\}$ is recognized as the $i^{th}$ Booth recoded radix-4 digit.

The squares of the two's complement tails are now shown to provide the foundation for our operand dual recoding.

*Theorem 1:* Let $q_i = t_i + t_{i+1} \bullet 4^{-1}$ for $0 \le i \le (p+1)/2$. Then $x^2 = \sum_{i=0}^{\left\lfloor \frac{p+1}{2} \right\rfloor} d_i q_i 16^{-i}$.

Furthermore, when $d_i$ and $q_i$ are both non zero, they have the same sign $(-1)^{b_{2i-1}}$, so then:

$$x^2 = \sum_{i=0}^{\left\lfloor \frac{p+1}{2} \right\rfloor} |d_i||q_i| 16^{-i}.$$

*Proof:* Note that $x^2 = \sum_{i=0}^{\left\lfloor \frac{p+1}{2} \right\rfloor} (t_i{}^2 - t_{i+1}{}^2 \bullet 16^{-1}) 16^{-i}$, and $(t_i{}^2 - t_{i+1}{}^2 \bullet 16^{-1}) = (t_i - t_{i+1} \bullet 4^{-1})(t_i + t_{i+1} 4^{-1}) = d_i q_i$ It is readily shown that: $d_i = (-1)^{b_{2i-1}} |d_i|$, and $q_i = (-1)^{b_{2i-1}} |q_i|$, so $d_i q_i = |d_i||q_i|$ for $0 \le i \le \left\lfloor \frac{p+1}{2} \right\rfloor$.

By performing the two's complement when dictated by $b_{2i-1} = 1$ and deleting the resulting sign, we obtain:

$$|q_i| = \begin{cases} b_{2i}.b_{2i+2}b_{2i+3}\ldots b_{p-1} & \text{for} \quad b_{2i-1} = 0 \\ b'_{2i}.b'_{2i+2}b'_{2i+3}\ldots b'_{p-2}b'_{p-1} & \text{for} \quad b_{2i-1} = 1 \end{cases}$$

where $b'_j = (1 - b_j)$ for $1 \le j \le p-2$, and $b'_{p-1} = (2 - b_{p-1})$.

In summary then our dual radix-4 recoding for squaring provides the sequence $|d_0|, |d_1|, \ldots, \left|d_{\frac{p-1}{2}}\right|$ of squarer digits where $d_i = -2b_{2i-1} + b_{2i} + b_{2i+1}$ is the $i$th radix-4 Booth recoded digit for $0 \le i \le \frac{p-1}{2}$. The dual recoding concurrently provides the sequence $|q_0|, |q_1|, \ldots, \left|q_{\frac{p-1}{2}}\right|$ of **squarands**, with $|d_i||q_i|16^{-1}$ the $i$th **partial square** for $0 \le i \le \frac{p-1}{2}$.

The radix-4 operand dual recoding for left-to-right squaring has a number of properties of considerable practical value for applications:

- The partial squares $|d_i||q_i|$ are all non-negative, so no sign extensions are needed.
- The partial squares are each scaled down by another power of 16, so an *n*-term sum provides an approximate square of about 4*n* bits of accuracy.
- The partial square generators are similar in design to Booth radix-4 partial product generators but simpler in two ways – no sign extensions are needed, and, on average, they are about half the size for the same precision.

For more details on the operand dual recoding see [M09]. It is illustrative to consider a sample squaring operation as shown in the tables. Consider the 16-bit squaring operation with $x$ normalized in the interval [½,1), in particular $x=0.1100010110001011_2$ in Example 1. This example uses the Radix-2 optimizations that were discussed before [PBD97]. As can be seen, the array contains 8 rows and 95 terms including guard bits. To visualize the optimizations achieved with the proposed Radix-4 method, the array in Example 2 below can be referred to. This utilizes a radix-4 dual recoding and employs $g$=3 guard bits yielding a result that has a 1½ ulp lower bound on $x^2$. In comparison, the array has only 4 rows and 50 terms respectively.

| | | | | | | | | | | | | | | | | Guard bits | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| | | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| | | | | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| | | | | | | | | | | | | | | 1 | 1 | 0 | | |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |

**Figure 37: Truncated Radix-2 Squarer Array**

| | | | | | | | | | | | | | | | | Guard bits | | | Squarer digits |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 2 |
| | | | | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | -2 |
| | | | | | | | | | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| | | | | | | | | | | | | | 1 | 0 | 1 | 1 | 1 | 0 | -1 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | |

**Figure 38: 16-bit Radix-4 Approximate Squarer Array**

## 5.2. Squaring Circuit Implementation

A block diagram of the squaring circuit is shown in Figure 39. The Verilog™ HDL used to implement the circuits and the Synopsys Design Compiler™ is used to synthesize the circuit using both 130nm and 90nm cell libraries from Texas Instruments. Various squaring circuits were synthesized for operand sizes of $n$=12, 16, and 24 bits. In each of these cases additional guard bits of $g$=2, 2, and 3 respectively were included to ensure that the approximation is bounded by at most 2 ulps accuracy. The resulting circuits were also analyzed for maximum path delay and power dissipation. For comparison purposes, an $n$-bit operand truncated multiplier was also synthesized into the same cell libraries with an $n$-bit product and an appropriate number of guard bits as this type of circuit is commonly used for the generation of an approximate square.

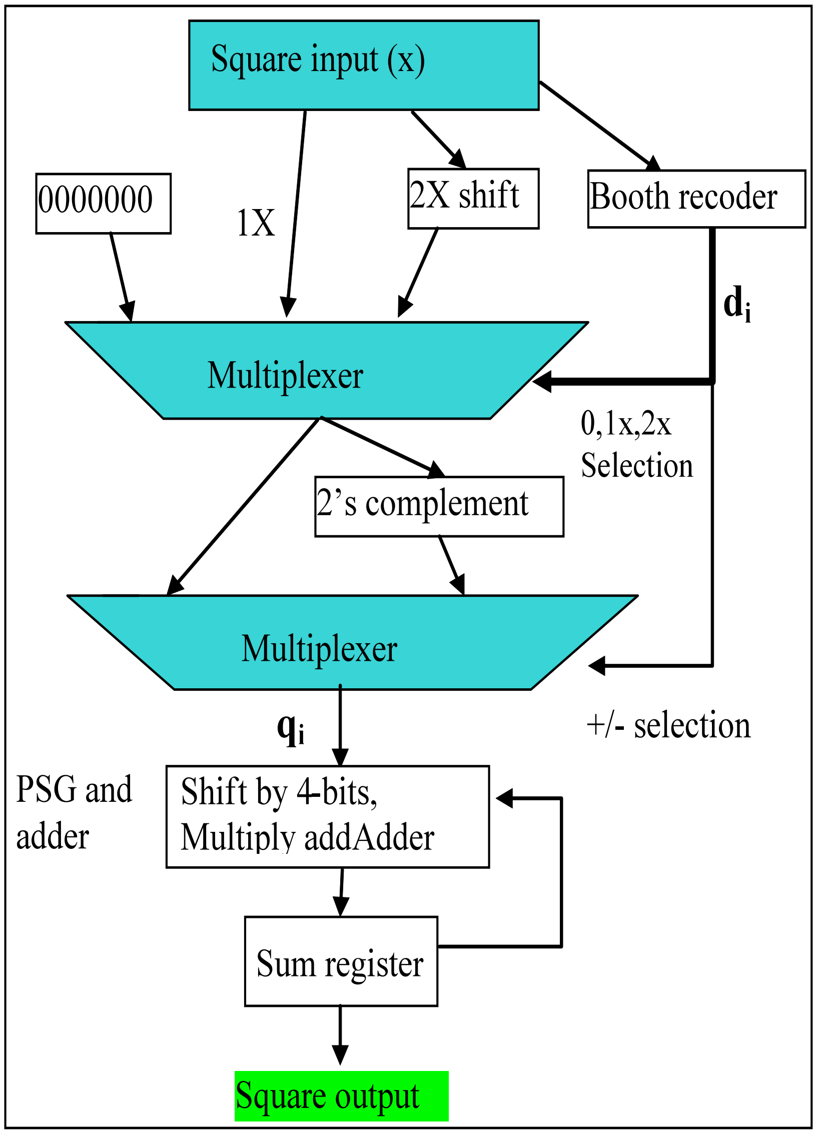


**Figure 39: Block Diagram of Truncated Squaring Circuit**

# Chapter 6

# QUATERNARY CIRCUITS MODELING AND VALIDATION

The functionality of the quaternary arithmetic circuit architectures are validated by simulation. The basic gates and functions are verified using a circuit simulator such as SPICE for transistor level verification. However, SPICE simulation runtime increases significantly with circuit size, prohibiting its use for larger circuits. For this reason, the larger circuits are validated by modeling them with higher level languages and then using a faster simulator [AG+07, GFD03]. Equivalent circuit models are created for the quaternary circuits to validate their functionality.

## 6.1. Quaternary Circuit Modeling

Traditional two-valued (binary) circuits are modeled and verified using a hardware description language such as Verilog or VHDL. However, MVL circuit implementations require extended data types and support for variable types that can handle more than two values. Verilog and VHDL languages both lack such features. For these reasons, we use System Verilog to model and verify the quaternary functions. As introduced in [AG+07], System Verilog is a very effective modeling language for modeling MVL circuits. The functionality of a circuit can be verified by simulating different possible input conditions to verify that the circuit produces valid outputs. The modeling of the basic quaternary functions is shown in this section. System Verilog descriptions of the quaternary one-variable functions are shown in Figure 40. As can be seen from the System Verilog modeling, a variable type integer was used to model the four logic values required for quaternary circuits.

```
function int J0(int a);
 int res;
 res = (a==0) ? 3 : 0;
 return res;
endfunction

function int J1(int a);
 int res;
 res = (a==1) ? 3 : 0;
 return res;
endfunction

function int J2(int a);
 int res;
 res = (a==2) ? 3 : 0;
 return res;
endfunction

function int J3(int a);
 int res;
 res = (a==3) ? 3 : 0;
 return res;
endfunction
```

**Figure 40: Quaternary One-variable Function Modeling with System Verilog**

The preliminary quaternary gates and quaternary half-adder circuit are modeled with System Verilog as shown in Figure 41. System Verilog made it possible to model the *MAX* and *MIN* functions using simple comparison operators. Half-adder functionality is modeled using the “case” statement of System Verilog. The carry-out and sum-out outputs of the half-adder are assigned values that depended directly on the conditions of the “a” and “b” inputs.

```
function int r_x_1(
int in);
int out;
out = ((4-1)-in);
return out;
endfunction
function int MIN(int a,
 int b);
int out;
out = ((a<b) ? a : b);
return out;
endfunction
function int MAX(int a,
 int b);
int out;
out = ((a>b) ? a : b);
return out;
endfunction
```

```
function int hadd(input int a, b,
              output int cout, sout);
 case ({a,b})
  {'h0,'h0}:{cout,sout}={'h0,'h0}; {'h0,'h1}:{cout,sout}={'h0,'h1};
  {'h0,'h2}:{cout,sout}={'h0,'h2};{'h0,'h3}:{cout,sout}={'h0,'h3};
  {'h1,'h0} : {cout,sout} = {'h0,'h1};{'h1,'h1}:{cout,sout}={'h0,'h2};
  {'h1,'h2} : {cout,sout} = {'h0,'h3};{'h1,'h3}:{cout,sout}={'h1,'h0};
  {'h2,'h0} : {cout,sout} = {'h0,'h2};{'h2,'h1}:{cout,sout}={'h0,'h3};
  {'h2,'h2} : {cout,sout} = {'h1,'h0};{'h2,'h3}:{cout,sout}={'h1,'h1};
  {'h3,'h0} : {cout,sout} = {'h0,'h3};{'h3,'h1}:{cout,sout}={'h1,'h0};
  {'h3,'h2} : {cout,sout} = {'h1,'h1};{'h3,'h3}:{cout,sout}={'h1,'h2};
 endcase
endfunction
```

**Figure 41: System Verilog Models for Basic Quaternary Gates and Half-adder**

The quaternary full-adder listing in the System Verilog language is shown in Figure 42. The full-adder functionality was also implemented using the case statement but extended it to three inputs. The case statement became larger because of the increase in the combinations of possible input values. The main advantage of the case statement is that it makes it easier for a designer to debug.

```
function int fadd( input int a, b, c, output int cout, sout);
 case ({a,b,c})
  {'h0,'h0,'h0} : {cout,sout} = {'h0,'h0}; {'h0,'h1,'h0} : {cout,sout} = {'h0,'h1};
  {'h0,'h2,'h0} : {cout,sout} = {'h0,'h2}; {'h0,'h3,'h0} : {cout,sout} = {'h0,'h3};
  {'h1,'h0,'h0} : {cout,sout} = {'h0,'h1}; {'h1,'h1,'h0} : {cout,sout} = {'h0,'h2};
  {'h1,'h2,'h0} : {cout,sout} = {'h0,'h3}; {'h1,'h3,'h0} : {cout,sout} = {'h1,'h0};
  {'h2,'h0,'h0} : {cout,sout} = {'h0,'h2}; {'h2,'h1,'h0} : {cout,sout} = {'h0,'h3};
  {'h2,'h2,'h0} : {cout,sout} = {'h1,'h0}; {'h2,'h3,'h0} : {cout,sout} = {'h1,'h1};
  {'h3,'h0,'h0} : {cout,sout} = {'h0,'h3}; {'h3,'h1,'h0} : {cout,sout} = {'h1,'h0};
  {'h3,'h2,'h0} : {cout,sout} = {'h1,'h1}; {'h3,'h3,'h0} : {cout,sout} = {'h1,'h2};

  {'h0,'h0,'h1} : {cout,sout} = {'h0,'h1}; {'h0,'h1,'h1} : {cout,sout} = {'h0,'h2};
  {'h0,'h2,'h1} : {cout,sout} = {'h0,'h3}; {'h0,'h3,'h1} : {cout,sout} = {'h1,'h0};
  {'h1,'h0,'h1} : {cout,sout} = {'h0,'h2}; {'h1,'h1,'h1} : {cout,sout} = {'h0,'h3};
  {'h1,'h2,'h1} : {cout,sout} = {'h1,'h0}; {'h1,'h3,'h1} : {cout,sout} = {'h1,'h1};
  {'h2,'h0,'h1} : {cout,sout} = {'h0,'h3}; {'h2,'h1,'h1} : {cout,sout} = {'h1,'h0};
  {'h2,'h2,'h1} : {cout,sout} = {'h1,'h1}; {'h2,'h3,'h1} : {cout,sout} = {'h1,'h2};
  {'h3,'h0,'h1} : {cout,sout} = {'h1,'h0}; {'h3,'h1,'h1} : {cout,sout} = {'h1,'h1};
  {'h3,'h2,'h1} : {cout,sout} = {'h1,'h2}; {'h3,'h3,'h1} : {cout,sout} = {'h1,'h3};

  {'h0,'h0,'h2} : {cout,sout} = {'h0,'h2}; {'h0,'h1,'h2} : {cout,sout} = {'h0,'h3};
  {'h0,'h2,'h2} : {cout,sout} = {'h1,'h0}; {'h0,'h3,'h2} : {cout,sout} = {'h1,'h1};
  {'h1,'h0,'h2} : {cout,sout} = {'h0,'h3}; {'h1,'h1,'h2} : {cout,sout} = {'h1,'h0};
  {'h1,'h2,'h2} : {cout,sout} = {'h1,'h1}; {'h1,'h3,'h2} : {cout,sout} = {'h1,'h2};
  {'h2,'h0,'h2} : {cout,sout} = {'h1,'h0}; {'h2,'h1,'h2} : {cout,sout} = {'h1,'h1};
  {'h2,'h2,'h2} : {cout,sout} = {'h1,'h2}; {'h2,'h3,'h2} : {cout,sout} = {'h1,'h3};
  {'h3,'h0,'h2} : {cout,sout} = {'h1,'h1}; {'h3,'h1,'h2} : {cout,sout} = {'h1,'h2};
  {'h3,'h2,'h2} : {cout,sout} = {'h1,'h3}; {'h3,'h3,'h2} : {cout,sout} = {'h2,'h0};

  {'h0,'h0,'h3} : {cout,sout} = {'h0,'h3}; {'h0,'h1,'h3} : {cout,sout} = {'h1,'h0};
  {'h0,'h2,'h3} : {cout,sout} = {'h1,'h1}; {'h0,'h3,'h3} : {cout,sout} = {'h1,'h2};
  {'h1,'h0,'h3} : {cout,sout} = {'h1,'h0}; {'h1,'h1,'h3} : {cout,sout} = {'h1,'h1};
  {'h1,'h2,'h3} : {cout,sout} = {'h1,'h2}; {'h1,'h3,'h3} : {cout,sout} = {'h1,'h3};
  {'h2,'h0,'h3} : {cout,sout} = {'h1,'h1}; {'h2,'h1,'h3} : {cout,sout} = {'h1,'h2};
  {'h2,'h2,'h3} : {cout,sout} = {'h1,'h3}; {'h2,'h3,'h3} : {cout,sout} = {'h2,'h0};
  {'h3,'h0,'h3} : {cout,sout} = {'h1,'h2}; {'h3,'h1,'h3} : {cout,sout} = {'h1,'h3};
  {'h3,'h2,'h3} : {cout,sout} = {'h2,'h0}; {'h3,'h3,'h3} : {cout,sout} = {'h2,'h1};
 endcase
endfunction
```

**Figure 42: Quaternary Full-adder Modeling with System Verilog**

The next illustration in Figure 43 shows the System Verilog modeling for the quaternary four-to-one multiplexer and the quaternary rising edge D flip-flop. The ability to reuse previously modeled functions simplifies the modeling of complex MVL circuits. The flip-flop circuit contains the feedback loops to retain the clocked input value. To avoid a potential race condition in the simulation, delay values are inserted when modeling the feedback loops in the flip-flop circuit.

```
module mux4(out,s,a,b,c,d);
output out;
input s, a, b, c, d;

 int  out, s, a, b, c, d;
 int s0, s1, s2, s3;
 int n0,n1,n2,n3,n4,n5;

 assign s0 = J0(s);
 assign s1 = J1(s);
 assign s2 = J2(s);
 assign s3 = J3(s);
 assign n0 = MIN(s0, a);
 assign n1 = MIN(s1, b);
 assign n2 = MIN(s2, c);
 assign n3 = MIN(s3, d);
 assign n4 = MAX(n0, n1);
 assign n5 = MAX(n2, n3);
 assign out = MAX(n4, n5);

endmodule
```

```
module flop (clk, d, q, qbar);
input clk, d;
output q, qbar;
int clk,d, d_1, d_2, d_3, d_4, q, qbar;

always @(d or d_2)
#1.01 d_1 = min_inv(d,d_2);
always @(d_1 or d_3 or clk)
#1.02 d_2 = min3_inv(d_1,d_3,clk);
always @(clk or d_4)
#1.03 d_3 = min_inv(clk,d_4);
always @(d_1 or d_3)
#1.04 d_4 = min_inv(d_1,d_3);

always @(d_3 or qbar)
#1.05 q = min_inv(d_3,qbar);

always @(q or d_2) begin
#1.06 qbar = min_inv(q,d_2);
end
endmodule
```

**Figure 43: Quaternary 4-to-1 Multiplexer & *D* Flip-Flop System Verilog Models**

The System Verilog listing of the quaternary single-digit multiplier is shown below in Figure 44. The most significant digit (MSD) and least significant digit (LSD) output values of the One-digit multiplier circuit are modeled using the case procedure that has dependency on the circuit inputs “a” and “b”.

```
function int mult1d(input int a, b, output int cout, sout);
 case ({a,b})
  {'h0,'h0} : {MSD,LSD} = {'h0,'h0};   {'h0,'h1} : {MSD,LSD} = {'h0,'h0};
  {'h0,'h2} : {MSD,LSD} = {'h0,'h0};   {'h0,'h3} : {MSD,LSD} = {'h0,'h0};
  {'h1,'h0} : {MSD,LSD} = {'h0,'h0};   {'h1,'h1} : {MSD,LSD} = {'h0,'h1};
  {'h1,'h2} : {MSD,LSD} = {'h0,'h2};   {'h1,'h3} : {MSD,LSD} = {'h0,'h3};
  {'h2,'h0} : {MSD,LSD} = {'h0,'h0};   {'h2,'h1} : {MSD,LSD} = {'h0,'h2};
  {'h2,'h2} : {MSD,LSD} = {'h1,'h0};   {'h2,'h3} : {MSD,LSD} = {'h1,'h2};
  {'h3,'h0} : {MSD,LSD} = {'h0,'h0};   {'h3,'h1} : {MSD,LSD} = {'h0,'h3};
  {'h3,'h2} : {MSD,LSD} = {'h1,'h2};   {'h3,'h3} : {MSD,LSD} = {'h2,'h1};
 endcase
endfunction
```

**Figure 44: Quaternary One-digit Multiplier Modeling with System Verilog**

## 6.2. Testbench Modeling

A testbench is constructed to simulate all possible input combinations in order to verify the quaternary circuit models. System Verilog is also very useful for the development of the testbench modules. For example, Figure 45 illustrates how System Verilog can be used to construct a testbench to verify the functionality of the quaternary rising edge *D* flip-flop. The output response of the design under test is captured in both textual and waveform formats for functional check and debug. The testbenches for other complex circuits are also constructed using System Verilog modeling in a similar fashion.

```
module flop_test;
int clk,d;
int q, qbar;

flop mod (clk,d,q,qbar);
int temp;
initial
begin
d=0;clk=0;
#15;temp=0;d=0;   #1 clk=3;
#15;temp=1;d=1;   #1 clk=0;
#15; temp=0;d=1;  #1 clk=3;
#15;temp=1;d=2;   #1 clk=0;
#15; temp=0;d=2;  #1 clk=3;
#10;d=3;          #1 clk=0;
#10;d=3;          #1 clk=3;
#10;d=0;          #1 clk=0;
end

initial
begin
$display("This is a test message");
$dumpfile("debug_flop");
$dumpvars();
end
always @(clk,d,q,qbar,temp)
$monitor($time,clk,d,q,qbar, temp);

endmodule
```

**Figure 45: Quaternary *D* Flip-Flop Testbench in System Verilog**

All of the System Verilog models are validated using the Synopsys© VCS© environment. The arithmetic circuits are also modeled in a similar fashion using System Verilog to verify their functionality.

# Chapter 7

# RESULTS

The logic library gates and multiplier circuits are modeled and verified using the System Verilog language to validate proper functionality. Additionally, functionality of the library cells is verified at the transistor level using the HSPICE tool. Characterization of the power dissipation, area, and performance of the adders and multipliers is accomplished by benchmarking the circuits with their binary equivalents for operand word sizes. The corresponding binary circuits are synthesized using the Synopsys design compiler synthesis tool for different operand word sizes. A 130 nm standard cell library from Texas Instruments is used for the binary circuits. It is noted that state-of-the-art synthesis optimization algorithms are used for the binary cases; however, no such synthesis tools are available for quaternary circuit synthesis hence the multiplier architectures are designed manually.

## 7.1. Quaternary Adder benchmarking

The comparison in Table 8 summarizes the observations for the various adder architectures for both quaternary and binary circuits.

### 7.1.1. Adders Circuit Area

In terms of area benchmarking, the number of transistors required for each of the quaternary and binary equivalent circuits are used as the metric. The reason for this increase is the fact that the SUSLOC structure adds additional transistor components to guarantee circuit stability and non-overlapping logic state resolution. Moreover, the binary circuits compared here are area optimized using the latest design synthesis tools.

**Table 8: Quaternary Adders Benchmarking Comparison**

| | | QUATERNARY | BINARY |
|---|---|---|---|
| | WORDSIZE | 32-DIGIT | 64-BIT |
| **Ripple-Carry** | Area (trans.) | 15341 | 11792 |
| | Power(uW) | 128.6 | 218 |
| | Levels(trans.) | 224 | 342 |
| **Carry-Select** | Area (trans.) | 32062 | 23936 |
| | Power (uW) | 261.8 | 441 |
| | Levels(trans.) | 116 | 118 |
| **CLA** | Area (trans.) | 16538 | 12454 |
| | Power (uW) | 138 | 238 |
| | Levels(trans.) | 82 | 94 |

It should be noted that the transistors used in the quaternary cells are typically larger than the transistors used for the binary circuit design. However, there is a reduction in signal routing because the quaternary versions of the circuits require fewer data signals due to their increased data density. With the current trends in modern device scaling and routing, the area reduction due to halving the signal conductor traces is very significant but cannot be measured until a design is actually physically implemented on a die. Three power distribution networks are required for the quaternary valued circuits versus the single power distribution network required for the binary circuits. The area requirements due to these differences in interconnects are also not accounted for in the comparison.

### 7.1.2. Adders Circuit Power

Another important comparison is the switching power. As shown in the results, the quaternary adder circuits exhibit lower switching power consumption when compared to the equivalent binary circuits. For the binary circuits, dynamic power analysis was accomplished through the use of the Synopsys PrimeTime-PX tool with the assumption of a 50% switching activity of all internal signals.

The quaternary circuit dynamic power dissipation is estimated by using Equation (3) and techniques in [N94, IY96].

$$P_{\mathrm{dyn}} = C_{\mathrm{tot}} \times a_{sw} \times (V_{avg})^2 \times f \tag{3}$$

$C_{\mathrm{tot}}$ represents the total capacitance estimated from the total number of transistors in the circuit. Total capacitance of the binary circuits is estimated by multiplying the number of equivalent two-input *NAND* gates with their input capacitance as shown in Equation (4). Correspondingly, Equation (4) is also used for the quaternary circuits using two-input *MIN* gate equivalents.

$$C_{\mathrm{tot}} = N_{gates} \times C_{gate} \tag{4}$$

$V_{avg}$ is the average voltage swing occurring during a transition. The quaternary transistor models utilize a 3.3V supply with 1.1V swings between the voltage encoded logic levels of *V*0, *V*1, *V*2, and *V*3. Assuming all logic transitions are equally likely for consistency with the binary case, there are 12 cases when voltage swings occur ranging from swings of 3.3V to 1.1V. Therefore $V_{avg}$=1.83V.

$a_{sw}$ represents the switching activity and is the probability that a non-zero voltage transition occurs within a given clock cycle. Each quaternary signal conveys the equivalent of two bits of information per conductor, thus the equivalent switching activity factor is 75%. The 75% switching factor arises from assuming the quaternary signal is equally likely to transition from its present logic level to any other within a clock cycle. Because there are a total of 12 non-zero transitions out of 16 possibilities, $a_{sw}$ =75%.

The term $f$ refers to the frequency of operation. The frequency used for the power calculations is derived using the Equation (5).

$$f = \frac{1}{2 \times T_{\max}} \tag{5}$$

The $T_{\max}$ term is obtained from the timing analysis results of the equivalent binary circuits. The factor two is used to accommodate enough margin for proper operation of both binary and quaternary circuits. The fundamental reason for this power saving is the fact that quaternary circuits have more intermediate logic levels and require less voltage swing on average when compared to traditional binary circuits. It should also be noted that the amount of switching power can vary per logic-level change in the quaternary circuits since switching from logic value 0 to logic value 3 requires more power than a switch from logic value 0 to logic value 1 due to the larger voltage swing.

### 7.1.3. Adders Circuit Speed

When reporting the performance, the longest gate path from input to output is used as a metric. Total number of transistors in such long timing path is used as benchmarking. In all cases, as shown in Table 8, the quaternary adder circuits exhibited better timing performance than the binary equivalent circuits as measured by the depth (no. of transistors) of the critical path. The savings in number of levels could be attributed to the fact that the quaternary circuits do have higher data integrity and hence reduced number of levels.

## 7.2. Quaternary Multiplier Benchmarking

Both the serial and the parallel multiplier circuits are implemented for three different operand word sizes. For benchmarking, traditional binary circuits with the equivalent operand sizes are also implemented. The three configurations chosen for the benchmarking and their binary equivalent sizes are:

- 16-digit quaternary(32-bit binary)
- 32-digit quaternary (64-bit binary)
- 64-digit quaternary (128-bit binary)

The benchmarking results are summarized in Table 9 as noted in [DT10, DTJ10].

### 7.2.1. Multiplier Circuit Area

Area is compared in the third and sixth rows of Table 9 and is measured in units of transistors. Area due to wiring interconnects is not included since the circuits were not placed and routed in physical design layout. The number of internal signal conductors is reduced by one-half in the quaternary circuits as compared to the binary case; however, there are also two additional supply voltage distribution networks required to supply the *V*1 and *V*2 rail voltages for the quaternary circuits. Based on this observation, it is anticipated that area due to interconnect networks will be approximately equivalent. In the case of the serial multiplication circuits, an average of 26% more transistors are required for the quaternary implementation. However, the quaternary parallel multiplier circuits exhibited a decrease of 13%, 20%, and 25% in number of transistors respectively for the same operand word sizes. The reason that the area decreases as word size increases in the parallel case is due to the exponential growth in the number of transistors with respect to word size (in units of digits) for the binary case and the corresponding word size is smaller for the quaternary circuits.

**Table 9: Quaternary Multiplier Benchmarking Comparison**

| | | QUATERNARY (DIGITS) | | | BINARY (BITS) | | |
|---|---|---|---|---|---|---|---|
| | **WORDSIZE** | 16 | 32 | 64 | 32 | 64 | 128 |
| **Serial** | Area (trans.) | 10619 | 20731 | 40955 | 8238 | 16380 | 32654 |
| | Power (uW) | 90.27 | 176.24 | 348.15 | 154.34 | 306.88 | 611.77 |
| | Levels(trans.) | 134 | 262 | 518 | 189 | 381 | 765 |
| **Parallel** | Area (trans.) | 75052 | 282054 | 1055993 | 85596 | 349744 | 1404789 |
| | Power (mW) | 0.44 | 1.69 | 4.89 | 0.91 | 3.91 | 12.63 |
| | Levels(trans.) | 240 | 442 | 806 | 332 | 638 | 1594 |

### 7.2.2. Multiplier Circuit Dynamic Power

Rows four and seven of Table 9 compare the dynamic (switching) power dissipation among the binary and quaternary multipliers. The serial multipliers indicated on an average a 43% decrease in power dissipation for the quaternary case versus the binary circuits. Whereas in the case of parallel multipliers, a very significant decrease in power dissipation is also observed for quaternary cases, averaging an approximate 60% decrease.

### 7.2.3. Multiplier Circuit Speed

Rows five and eight of Table 9 estimate and compare circuit speed of both quaternary and binary cases. The speed is indicated here as the number of transistor levels in the performance critical paths. The number of transistors in the maximum depth path in each circuit is measured. The smaller the number of transistors, the higher the speed of operation of the corresponding circuit. In all cases, the quaternary multiplication circuits show reduced path length (and hence increased speed of operation) when compared to the binary circuits. In the case of serial multipliers, the quaternary maximum path length was, on average, reduced by an approximate 31%. In the case of parallel multipliers, path length decreases of approximately 28%, 31%, and 50% respectively achieved for the 16-, 32-, and 64-digit multipliers as compared to the equivalent binary circuits.

## 7.3. Quaternary Serial Multiplier Implementation Analysis

As discussed previously in the multiplier section, the adders in the serial multiplier circuit can be implemented using the different possible adder architectures. Three different adder architectures are used to implement the quaternary serial multiplier, Ripple-Carry, Carry-Select (CS) and Carry-look-ahead (CLA) for different operand sizes. Table 10 provides the estimated benchmarking of all three different quaternary serial multiplier implementations.

**Table 10: Quaternary Serial Multiplier Benchmarking Comparison**

| | | QUATERNARY SERIAL MULTIPLIER | | |
|---|---|---|---|---|
| | METRIC | 16-DIGIT | 32-DIGIT | 64-DIGIT |
| **Ripple-Carry Adder Based** | Area (trans.) | 10619 | 20731 | 40955 |
| | Power (uW) | 90.27 | 176.24 | 348.15 |
| | Levels | 134 | 262 | 518 |
| **Carry-Select Adder Based** | Area (trans.) | 22088 | 43120 | 85186 |
| | Power (uW) | 187.76 | 366.57 | 724.15 |
| | Levels | 68 | 134 | 266 |
| **CLA Adder Based** | Area (trans.) | 11522 | 22908 | 46361 |
| | Power (uW) | 96.59 | 192.62 | 390.28 |
| | Levels | 52 | 92 | 146 |

## 7.4. Quaternary Parallel Multiplier Implementation Analysis

As discussed previously, addition operation of the partial products in the quaternary parallel multiplier circuits can be implemented in different ways. We implemented the addition operation using traditional array type adders and also using the Wallace tree architecture. Table 11 captures the results of the both implementations for the 4X4 quaternary multiplier case.

**Table 11: Quaternary 4*4 Parallel Multiplier Architecture Comparison**

| | METRIC | 4-DIGIT |
|---|---|---|
| **Array tree** | Area (trans.) | 22262 |
| | Power (uW) | 122.36 |
| | Levels | 63 |
| **Wallace tree** | Area (trans.) | 20370 |
| | Power (uW) | 92.08 |
| | Levels | 24 |

## 7.5. Results of Squaring Circuit Optimization

Table 12 and Table 13 contain the synthesis results in terms of path delay, power dissipation and area. We note that these results do not include delay due to routing however, since the multiplier adder array is more complicated than the reduced squaring adder arrays, the comparison of the truncated multiplier array to the squaring circuits is likely very conservative after including actual wire delays.

The results show that both the radix-2 and the radix-4 circuits yield a dramatic improvement in performance, power and area compared to the truncated multiplier circuit. The reductions range from a factor of two-to-three for delay, three-to-four for area and five-to-six for power.

**Table 12: Synthesis Results using 130nm Cell Library**

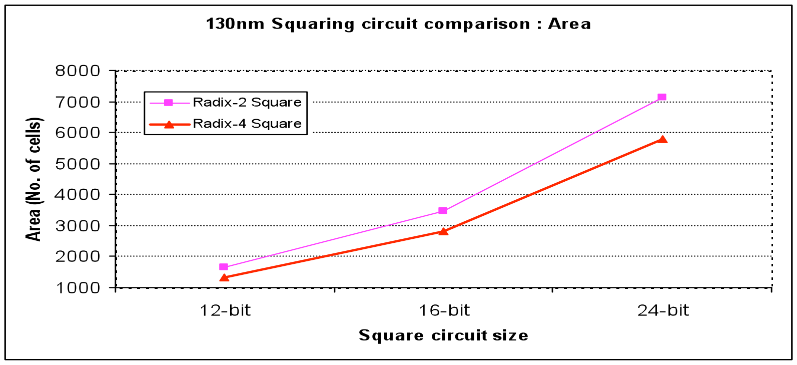
| **Circuit** | **Delay(ps)** | **Freq(Hz)** | **Dynamic Power (mW)** | **Leakage Power (nW)** | **Area (Gate count)** |
|---|---|---|---|---|---|
| 12-bit Truncated Multiplier | 2160 | 4.63E+08 | 15.54 | 48.01 | 4001 |
| 16-bit Truncated Multiplier | 2950 | 3.39E+08 | 41.18 | 105.65 | 8669 |
| 24-bit Truncated Multiplier | 4690 | 2.13E+08 | 133.06 | 258.91 | 22140 |
| 12-bit Radix-2 Squarer | 1030 | 9.71E+08 | 4.43 | 18.54 | 1659 |
| 16-bit Radix-2 Squarer | 1180 | 8.47E+08 | 8.69 | 34.1 | 3480 |
| 24-bit Radix-2 Squarer | 1740 | 5.75E+08 | 23.72 | 76.53 | 7140 |
| 12-bit Radix-4 Squarer | **860** | **1.16E+09** | **3.33** | **14.57** | **1310** |
| 16-bit Radix-4 Squarer | **1030** | **9.71E+08** | **7.71** | **30.65** | **2826** |
| 24-bit Radix-4 Squarer | **1620** | **6.17E+08** | **21.23** | **61.67** | **5790** |

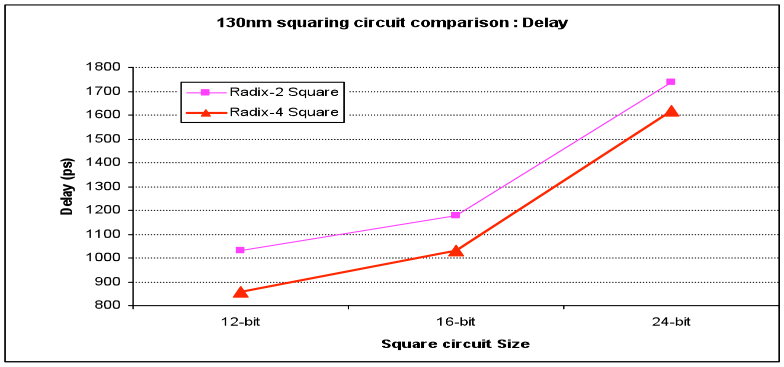
The radix-4 squarer performs better than the radix-2 squarer by about 10-to-20% in all metrics with the greatest reduction being in area. Figure 46 illustrates the comparisons of the radix-2 and radix-4 squarers in more details showing greater improvements as the size of the operand increases.

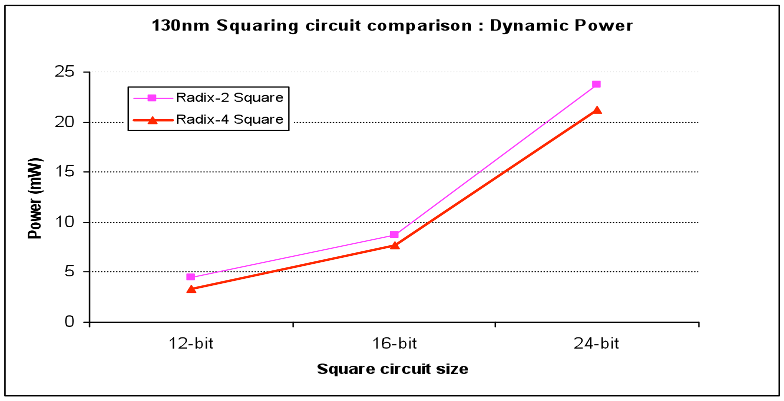
**Table 13: Synthesis Results using 90nm Cell library**

| Circuit | Delay(ps) | Freq(Hz) | Dynamic Power (mW) | Leakage Power (nW) | Area (Gate count) |
|---|---|---|---|---|---|
| 12-bit Truncated Multiplier | 1700 | 5.88E+08 | 7.09 | 98.89 | 4119 |
| 16-bit Truncated Multiplier | 2390 | 4.18E+08 | 17 | 188.54 | 7680 |
| 24-bit Truncated Multiplier | 3380 | 2.96E+08 | 58.81 | 523.62 | 21112 |
| 12-bit Radix-2 Squarer | 770 | 1.30E+09 | 2.16 | 46.3 | 1802 |
| 16-bit Radix-2 Squarer | 930 | 1.08E+09 | 4 | 73.3 | 3262 |
| 24-bit Radix-2 Squarer | 1390 | 7.19E+08 | 10.8 | 160.67 | 6783 |
| 12-bit Radix-4 Squarer | **650** | **1.54E+09** | **1.82** | **38.40** | **1522** |
| 16-bit Radix-4 Squarer | **800** | **1.25E+09** | **3.03** | **59.61** | **2470** |
| 24-bit Radix-4 Squarer | **1250** | **8.00E+08** | **9.97** | **145.47** | **5253** |

To summarize, the savings obtained with the approximate squaring circuit when compared with the equivalent sized radix-2 multiplier, Figure 46 contains comparison charts that illustrate the delay, power and area results of the 12-, 16-, and 24-bit operand approximate squaring circuits. It is clearly seen that the proposed circuit has significant savings and the efficiency of the proposed circuit gets better with increasing operand size.







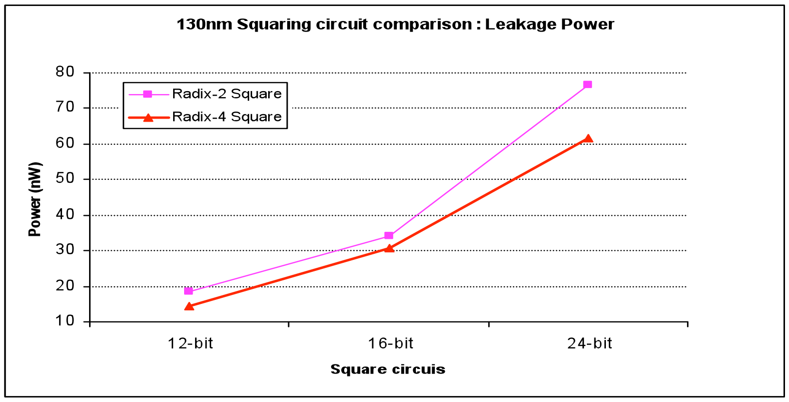


**Figure 46: Metrics Comparison of Radix-4 & Radix-2 Squaring Circuit**

# Chapter 8

# CONCLUSIONS AND FUTURE CHALLENGES

## 8.1. Conclusions

The feasibility of Voltage-mode quaternary circuits based on SUSLOC technology is demonstrated. A library of preliminary quaternary gates is designed using transistors based on the SUSLOC structure. The preliminary quaternary gates are simulated using the SPICE models. The larger quaternary arithmetic circuits are then implemented using the preliminary gates. Modeling techniques using System Verilog are successfully adopted for the modeling of the quaternary circuits. To verify the functionality, the quaternary circuits are also simulated using the System Verilog simulator using the test benches along with required input stimulus.

The quaternary circuits are benchmarked with equivalent two-valued (binary) circuits. The results show that the proposed adder and multiplier quaternary circuits are better than the equivalent binary circuits in the aspects of switching power and logic depth. The area optimization of the quaternary circuits will be possible with the availability of more application specific quaternary cells. In addition, the availability of any design optimization and synthesis CAD tools would further optimize the quaternary circuits. The advantages of lower power, higher performance, and reduced interconnect congestion motivate the use of quaternary circuits in a wide variety of applications.

## 8.2. Future Challenges

The reduced scaling of semiconductor process technologies periodically allow the increased logic integration and reduced power consumption. There are significant circuit implementation challenges, even for binary circuits, at advanced deep submicron technologies when the supply voltages are below 1.0V range(which is true for 65nm, 45nm and beyond). Some of the potential challenges in implementing quaternary (for that matter, any MVL) circuits at DSM technologies are discussed here.

### 8.2.1. Signal-to-Noise Ratio (SNR)

As discussed before in this report, SNR plays very important role in the design of reliable circuits. The quaternary circuits implemented as part of this work use the step voltage value of 1.1V. Larger voltage fluctuations in the signal values would leave very small SNR margin to clearly distinguish various logic levels in the circuit. So, the main challenge of adoption of quaternary circuits is to choose reliable circuit techniques and sufficiently large voltage step values to give sufficient SNR margins.

### 8.2.2. Process (manufacturing) Variations.

Extremely small transistor and wire dimensions are making the circuits (even normal CMOS binary circuits) to be very sensitive to the manufacturing (fabrication) variations. The printability (fabrication) of the transistors with the available laser technology is heavily impacted at ultra-DSM technologies. The MVL circuits would be more sensitive to these variations. The reason is, threshold voltage of transistors can have large local variation in DSM technologies. MVL circuits are very sensitive to any variation in the threshold voltages as it simply makes the circuits non-functional and have logic level overlaps.

### 8.2.3. Threat of Crosstalk Delays.

As it is reported in [DS+03] and [D04], shrinking geometries at the deep submicron technologies are making the circuits to be susceptible to crosstalk noise.

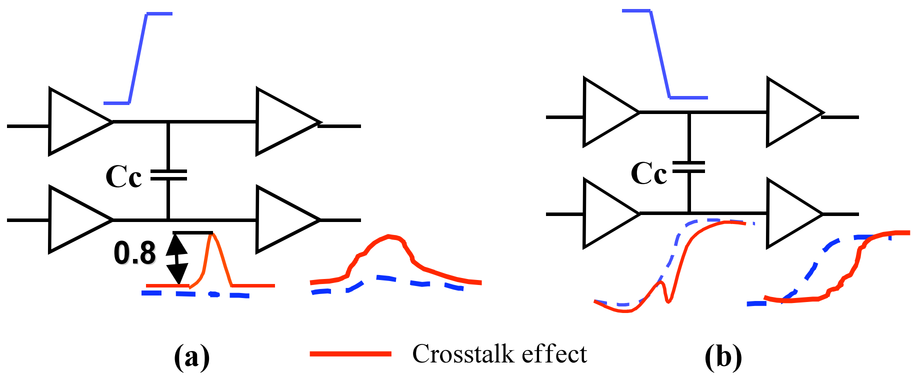


**Figure 47: Crosstalk Challenge**

As shown in the Figure 47, the signal transitions in the neighboring wires would essentially create signal surges and dips depending on the direction of the original transition. This is affecting even the traditional binary CMOS circuits in DSM technologies. These crosstalk effects would pose serious challenges to the MVL circuits at DSM technologies. The increase and decrease in the signal levels would make the receiving circuits incorrectly responding or latching onto unwanted noise.

The ongoing research and potential future research in the field of MVL logic is expected to address the highlighted challenges and other applicable challenges for wider adoption of MVL circuits.

# REFERENCES

[AB+99] S.L. Augustine, D.C. Buhler, and B.G. Prouty, "Computer Graphics System with Improved Blending", *United States Patent*, No. 5,896,136, April 20, 1999.

[AG+07] M. Amoui, D. Groβe, M.A. Thornton, and R. Drechsler, "Evaluation of Toggle Coverage for MVL Circuits Specified in the System Verilog HDL", *Proc. of IEEE Int. Symp. On Multiple-Valued Logic*, Session 8B, Paper 2, 2007.

[AH75] J.I Accha, J.L Huertas, "General Excitation table for a JK multistable", *Electronics Letters*, Vol.11, P. 624, 1975.

[AS+03] M. Aline, T. Saidi, E. Kinvi-Boh, O. Sentieys, E.D. Olson, "Design and Characterization of a Low Power Ternary DSP", *Proc. of International Signal Processing Conference*, 2003.

[B51] Booth, A.D., "A Signed Binary Multiplication Technique", *Quaterly J. Mechanics and Applied Mathematics,* Vol. 4, P.2, pp.236-240, June 1951.

[B91] Butler, J.T. (editor), **Multiple-Valued Logic in VLSI Design**, *IEEE Computer Society Press*, 1991.

[BK99] R.K. Brayton and S.P. Khatri, "Multiple-valued logic synthesis," in *Proc., 12th Int. Conf. on VLSI Design*, Jan. 1999.

[C60] M. Cohn, "Switching Function Canonical Forms over Integer Fields," *Ph.D. Thesis*, Harvard University, 1960.

[C71] T.C. Chen, "A Binary Multiplication Based on Squaring." *IEEE Trans. Computers*, C-20:678-80, 1971.

[C90] K. W. Current, "A CMOS Quaternary Threshold Logic Full Adder Circuit with Transparent Latch," *Proc. 20th Int. Symp. On Multiple Valued Logic,* pp. 168-173, 1990.

[C+02] Castro, H.A. et al., "A 125 MHz Burst Mode 0.18µm 128 Mbit 2 Bits per Cell Flash Memory", *Proceedings of the VLSI Symp. On Technology Circuits*, 2002.

[CBC06] R.G. Cunha, H. Boudinov, and L. Carro, "A Novel Voltage-Mode CMOS Quaternary Logic Design", *IEEE Trans. On Electronic Devices*, vol. 53, no. 6, pp. 1480-1483, 2006.

[CC94] Wei-Shang Chu, Current, W., **"**Quaternary multiplier circuit", *Proc. of IEEE Int. Symp. On Multiple-Valued Logic*, pp.15-18, 1994.

[CC95] Chu, W.-S., Current, W., "Current-mode CMOS quaternary multiplier circuit", *Electronic letters***,** Volume 31, Issue 4, pp. 267 – 268, 16 Feb 1995.

[CIL06] G.R. Cunha, B.H. Ivanov, and C. Luigi, "A low power high performance CMOS voltage-mode quaternary full adder", *IFIP International conference on Very Large Scale Integration,* pp. 187-191, 2006,.

[CO+96] Current, K.W., Oklobdzija, V.G., Maksimovic, D., "Low-energy logic circuit techniques for multiple valued logic", *Proc. of 26th International Symposium on Multiple-Valued Logic*, Page(s):86 – 90, 1996.

[CS+89] Capasso, F., Sen, S., Beltram, F., Lunardi, L.M., Vengurlekar, A.S., Smith, P.R., Shah, N.J., Malik, R.J., Cho, A.Y., "Quantum functional devices: resonant-tunneling transistors, circuits with reduced complexity, and multiple valued logic", *IEEE Transactions on Electron Devices*, Volume 36, Issue 10, pp:2065 – 2082, Oct. 1989.

[D65] Dadda, L., "Some Schemes for Parallel Multipliers", *Alta Frequenza*, Vol. 34, pp. 349-356, 1965.

[D85] L. Dadda, "Squarers for binary numbers in serial form", *Proc. IEEE 7th Symp. Computer Arithmetic*, 1985.

[D99] E. Dubrova, Multiple-Valued Logic in VLSI: Challenges and Opportunities, *Proceedings of NORCHIP'99*, pp. 340-350, 1999.

[D04] Satyendra R. Datla, "Crosstalk Delay Analysis in Very Deep Submicron VLSI circuits", *Masters Thesis*, 2004.

[DM+77] T.T.Dao, E.J.MaCluskey and L.K.Russek "Multivalued Integrated Injection Logic," *IEEE Trans. Comput*ers, v01.C-29, no.12, pp.1233-1241, 1977.

[DS01] D. De Caro and A. G. M. Strollo, "Parallel squarer using Booth-folding technique," *Electronic Letters*, vol. 37, no. 6, pp. 346–347, Mar. 2001.

[DS+03] Satyendra R. Datla, James SW Song, Barry Warren, Yuanqiao Zheng, "Crosstalk Delay Threat: Are you ready?!", *SNUG Boston*, 2003.

[DT10] Satyendra R. Datla, M.A. Thornton, "Quaternary Voltage-Mode Logic Cells and Fixed-Point Multiplication Circuits", Submitted for *IEEE International Symposium On Multiple-Valued Logic,* 2010.

[DTJ10] Satyendra R. Datla, M.A. Thornton, "Design and Validation of Voltage-Mode Quaternary Fixed-Point Arithmetic Circuits", Submitted for *IEEE Trans. On Computers,* 2010.

[DT+09] Satyendra R. Datla, M.A. Thornton, Luther Hendrix, Dave Henderson, "Quaternary Addition Circuits Based on SUSLOC Voltage-Mode Cells and Modeling with SystemVerilog", *IEEE International Symposium On Multiple-Valued Logic,* May 2009.

[DTM09] Satyendra R. Datla, M.A. Thornton, David W. Matula, "A Low Power High Performance Radix-4 Approximate Squaring Circuit", 20th *IEEE International Conference On Application-specific Systems, Architectures and Processors,* July 2009.

[DW08] Dakhole, P.K.; Wakde, D.G., "Multi-digit quaternary adder on programmable device : Design & verification", *Proc. of International Conference on Electronic Design*, pp. 1-4, 2008.

[E03] M. Ercegovac, "Left-to-Right Squarer with Overlapped LS and MS parts", *Conference Record of the 37th Asilomar Conference on Signals, Systems and Computers*, Volume 2, pp.1451-1455. November 2003.

[E90] Ercegovac, M.D. and T. Lang, "Fast Multiplication Without Carry-Propagate Addition", *IEEE Trans. Computers*, Vol. 39, No. 11, pp. 1385-1390, 1990.

[E92] D. Etiemble, "On the Performance of Multivalued Integrated Circuits: Past, Present and Future", *Proc. of Twenty-Second International Symposium on Multiple-Valued Logic*, pp. 156 – 164, May 1992.

[E93] G. Epstein, **Multiple-Valued Logic Design: An Introduction**, *IOP Publishing Ltd*, 1993.

[EI74] Etiemble, D., and Israel, M., "on the realization of multiple-valued flipflops" *Proc. 4th Int. Symp. Multiple-Valued Logic*, pp. 437-548, May 1974.

[EI88] Entieble, D. and Israel, M., "Comparison of Binary and Multivalued ICs According to VLSI Criteria", *IEEE Computer Magazine*, pp. 28-42, 1988.

[EL04] M. Ercegovac and T. Lang, **Digital Arithmetic**, *Morgan Kaufmann Publishers*, 2004.

[F81] P. M. Farmwald, "High Bandwidth Evaluation of Elementary Functions**,**" *Proc. IEEE 5th Symp. Computer Arithmetic*, pp. 139-142. 1981.

[FW01] Y.Yu Fengqi and A. N.Willson,"Multirate digital squarer architectures," in *Proc. 8th IEEE Int. Conf. on Electronics, Circuits and Systems (ICECS 2001)*, Malta, Sept. 2–5, pp. 177–180, 2001.

[GFD03] D. Große, G. Fey, and R. Drechsler, "Modeling Multi-Valued Circuits in SystemC", *International Symposium on Multiple Valued Logic*, pp. 281–286, 2003.

[H84] S. Hurst, "Multiple-valued logic its status and its future", *IEEE trans. On Computers*. C-33(12), pp.1160-1179, 1984.

[H01] Brian Hayes, "Third base", *American Scientist*, Volume 89, number 6, pp. 490, November-December 2001.

[HC98] M.K. Habib and A.K. Cherri, "Parallel Quaternary Signed-Digit Arithmetic Operations: Addition, Subtraction, Multiplication, and Division", *Optics and Laser Technology*, vol. 30, pp. 515-525, 1998.

[I86] Okihiko Ishuzuka, "Synthesis of a Pass Transistor Network Applied to Multi-Valued Logic", *Proc. of the 16th IEEE ISMVL*, pp. 51-57, 1986.

[I98] The Intel Corporation, Intel® StrataFlash™ Memory Technology, Application Note AP-677, 1998.

[IO+97] Ishizuka, O., Ohta, A., Tannno, K., Tang, Z., Handoko, D**, "**VLSI design of a quaternary multiplier with direct generation of partial products"**,** *Proc. of IEEE Int. Symp. On Multiple-Valued Logic*, pp.169-174, 1997.

[IY96] T. Ishihara, H. Yasuura, "Experimental Analysis of Power Estimation Models of CMOS VLSI Circuits," *IEICE Trans. Fundamentals*, vol. E00-A, No. 6, June 1996.

[JM95] J. Junell and K. Mikko, "Frame Synchronization in a Device Receiving Digital Radio Transmissions, *European Patent Application"*, 95108198.3, May 30, 1995.

[K38] S.C. Kleene, "On a notation for ordinal numbers," *The Journal of Symbolic Logic*, vol. 3, pp. 150-155, 1938.

[K81] D. Knuth, **The Art of Computer Programming: Seminumerical Algorithms**, *Addison Wesley*, Vol. 2, 2nd Edition, pp: 441-466, 1981.

[K90] M. Kameyama, "Toward the Age of Beyond-Binary Electronics and Systems", *Proc. of IEEE Int. Symp. On Multiple-Valued Logic*, pp.162-166, 1990.

[KA+03] E. Kinvi-Boh, M. Aline, O. Sentieys, and E.D. Olson, "MVL Circuit Design and Characterization at the Transistor Level Using SUS-LOC", *International Symposium on Multiple Valued Logic*, 2003.

[KB+08] Khan, M.M.M., Biswas, A.K., Chowdhury, S., Tanzid, M., Mohsin, K.M., Hasan, M., Khan, A.I., "Quantum realization of some quaternary circuits", *IEEE Region 10 Conference (TENCON)*, pp. 1 – 5, Nov. 2008.

[KG98] R. K. Kolagotla and W. R. Griescbach, "VLSI implementation of a 350 MHz 0.35 m 8 bit merged squarer," *Electronic Letters*, vol. 34, no. 1, pp. 47–48, Jan. 1998.

[KK+87] S . Kawahito, M. Kameyama, T. Higuchi, and H. Yamada, "A high speed compact multiplier based on multiple-valued bi-directional current-mode circuits," *Proc. 17th Int. Symp. On Multiple Valued Logic,* pp. 172-180, 1987.

[KK+88] Kameyama, M., Kawahito, S., and Higuchi, T., "A Multiplier Chip with Multiple-Valued Bidirectional Current-Mode Logic Circuits", *IEEE Computer Magazine*, pp. 43-56, 1988.

[L20] J. Lukasiewicz, *tr. on three valued logic*, Ruch Filozoficzny, vol 5, pp. 169-171, 1920.

[L70] H. Ling,"High-speed computer multi-plication using a multiple-bit decoding algorithm," *IEEE Trans. Computers*, C-19, pp. 706-709, Aug. 1970.

[LF00] A. Liddicoat and M. J. Flynn, "Parallel Square and Cube Computations", *Conference Record of the 34th Asilomar Conference on Signals, Systems & Computers,* 2000.

[M09] D. W. Matula, "Higher Radix Squaring Operations Employing Operand Dual Recoding", *Proc. IEEE 19th Symp. Computer Arithmetic*, June 2009.

[MC86] J. L. Mangin and K. W. Current, "Characteristics of prototype CMOS quaternary logic encoder-decoder circuits," *IEEE Trans. Comput.,* vol. C-35, pp. 157-161, Feb. 1986.

[MM79] J.C. Muzio and D.M.Miller, "On the minimization of many valued functions," *Proc. 9th Int. Symp. Multiple-Valued Logic*, June 1979.

[MM+08] J. Moore, D.W. Matula, M.L. Thornton, "A Low Power Radix-4 Dual Recoded Integer Squaring Implementation For Use in Design of Application

Specific Arithmetic Circuits", *Conference Record of the 42nd Asilomar Conference on Signals, Systems and Computers,* October 2008.

[MT08] D.M. Miller and M.A. Thornton, **Mutiple-Valued Logic: Concepts and Representations**. *Morgan & Claypool Publishers*, San Rafael, CA, ISBN 10-1598291904, 2008.

[MW86] Muzio, J.C. and Wesselkamper, T.C., **Multiple-Valued Switching Theory**, Adam Hilger, Bristol and Boston, 1986.

[N94] F. N. Najm, "A survey of power estimation techniques in VLSI circuits," *IEEE Transactions on VLSI Systems*, 2(4):446–455, 1994.

[O99] E.D. Olson, "Supplementary Symmetrical Logic Circuit Structure", *International Symposium on Multiple Valued Logic*, pp. 42-47, 1999.

[O00] U.S.Patent, *6,133,754,* Edgar Danny Olson, inventor, Multiple-Valued Logic Circuit Architecture, Supplementary Symmetrical Logic Circuit Structure (SUS-LOC), 2000.

[OC00] Olson, D. and Current, K.W., "Hardware Implementation of Supplementary Symmetrical Logic Circuit Structure Concepts", *Proc. of IEEE Int. Symp. On Multiple-Valued Logic*, pp. 371-376, 2000.

[OC01] Olson, D. and Current, K.W., "Demonstration of Supplementary Symmetrical Logic Circuit Structure Concepts using a MOS Test Chip", *International Journal of Multiple-Valued Logic*, vol. 7, pp. 1-23, 2001.

[P21] E.L.Post, "Introduction to a general theory of elementary propositions," in *American Journal of Mathematics*, June 1921.

[P71] S.D. Pezaris, "A 40-ns 17-bit array multiplier," *IEEE Trans. On Computers*, vol. C-20, pp. 442-447, 1971.

[P74] D.K. Pradhan, "A multivalued switching algebra based on finite fields," *Proc. Int. Symposium on Multiple valued Logic*, pp. 95-111, 1974.

[P00] B. Parhami, **Computer Arithmetic Algorithms and Hardware Designs**, *Oxford University Press*, pp. 143-145, 2000.

[PA96] J. Pihl and E. J. Aas, "A multiplier and squarer generator for high performance DSP applications," in *Proc. IEEE 39th Midwest Symp. on Circuits and Systems*, pp. 109–112, 1996.

[PBD97] A. Pirson, J.-M. Bard, and M. Daoudi, "Squaring Circuit for Binary Numbers", *United States Patent*, No. 5,629,885, May 13, 1997.

[Q55] W. Quine, "A way to simplify truth functions," *Amer. Math. monthly*, vol 62, pp. 627-631, 1955.

[QC04] "Quantum computing roadmap overview", version 2.0, *quantum information science and technology roadmapping project*, April 2004, available at http://qist.lanl.gov/qcomp_map.shtml.

[R55] Robertson, J.E., "Two's complement multiplication in Binary Parallel Computers", *IRE Trans. Electronic Computers*, Vol. 4, No.3, pp118-119, September 1955.

[R75] Rubinfield, L.P., "A Proof of the Modified Booth's Algorithm for Multiplication", *IEEE Trans. Computers*, Vol. 25, No. 10, pp. 1014-1015, 1975.

[S88] Smith, K.C., "Multiple-Valued Logic: A Tutorial and Appreciation", *IEEE Computer Magazine*, pp. 17-27, 1988.

[S93] Summerfield, S.¸"Design methodology of VLSI with multiple valued logic", ISCAS '93, 1993 *Prof. of IEEE International Symposium on Circuits and Systems*, Page(s):1702 - 1705 vol.3, 1993.

[SD03] A.G.H. Strollo and D. De Caro, "Booth Folding Encoding for High Performance Squarer Circuits" *IEEE Trans. Circuits and Systems-I1 Analog and Digital Signal Processing,* 50(5):250-254, 2003.

[SH08] H. Shirahama and T. Hanyu, "Design of High-Performance Quaternary Adders Based on Output-Generator Sharing", *International Symposium on Multiple Valued Logic,* 2008.

[SN+90] Shanbhag, N.R.; Nagchoudhuri, D.; Siferd, R.E.; Visweswaran, G.S., "Quaternary logic circuits in 2-µm CMOS technology", *IEEE Journal of Solid-State Circuits*, Volume 25, Issue 3, pp. 790-799, June 1990.

[TS+01] I.M. Thoidis, D. Soudris, J.M. Fernandez, and A. Thanailakis, The circuit design of Multiple-Valued Logic Voltage-Mode Adders. *Proc. of IEEE International Symposium on Circuits and Systems*, pp.162-165, 2001.

[W64] Wallace, C.S., "A suggestion for a Fast Multiplier", *IEEE Trans. Electronic Computers*, Vol. 13, pp. 14-17, 1964.

[WS01] K. E. Wires, M. J. Schulte, and J. E. Stine, "Combined IEEE Compliant and Truncated Floating Point Multipliers for Reduced Power Dissipation," in Proceedings of the *IEEE International Conference on Computer Design*, Austin, TX, pp. 497-500, September, 2001.

[WS98] C.L. Wey and M.D. Shieh. “Design of a High-Speed Square Generator”, *IEEE Trans. Computers*, vol. 47, no. 9, pp. 1021-1026, 1998.

[WS+99] K.E. Wires, M.J. Schulte, L.P. Marquette: and P.I. Balzola. "Combined Unsigned and Two's Complement Squarers", *Conference Record of the 31st Asilomar Conference on Signals, Systems and Computers*, Volume 2, pp. 1215-1219, 1999.

[WS01] E.G. Walters III, J.S. Schlessman, and M.J. Schulte, “Combined Unsigned and Two’s Complement Hybrid Squarers”, *Conference Record of the 35th Asilomar Conference on Signals, Systems & Computers*, pp. 861-866, November 2001.

[WT89] F. Wakui and M. Tanaka, “Comparison of Binary Full Adder and Quaternary Signed-Digit Full Adder using High-Speed ECL”, *International Symposium on Multiple Valued Logic*, pp. 346-355, 1989.

[YK+97] J.-T. Jae-tack Yoo, K. F. Kent F. Smith, and G. Ganesh Gopalakrishnan, “A fast parallel squarer based on divide-and-conquer,” *IEEE J. Solid-State Circuits*, vol. 32, pp. 909–912, June 1997.

[YT+86] Y. Yasuda, Y. Tokuda, S. Zhaima, K. Pak, T. Nakamura, and A. Yoshida, “Realization of quaternary logic circuits by N-Channel MOS Devices”, *IEEE Journal of Solid State Circuits*, vol. SC-21, no. 1, pp. 162-168, 1986.

[ZW90] N. Zhuang, H.Wu, “Novel ternary JKL flipflop“, *Electronics Letters*,Vol.26,No.15, pp.1145-1146, 1990.