2013

Multi-Valued Majority Logic Circuits Using Spin Waves

Sankara Narayanan Rajapandian
University of Massachusetts Amherst

MULTI-VALUED MAJORITY LOGIC CIRCUITS USING SPIN WAVES

A Thesis Presented

by

SANKARA NARAYANAN RAJAPANDIAN

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL AND COMPUTER ENGINEERING

September 2013

Electrical and Computer Engineering

MULTI-VALUED MAJORITY LOGIC CIRCUITS USING SPIN WAVES

A Thesis Presented

by

SANKARA NARAYANAN RAJAPANDIAN

Approved as to style and content by:

Csaba Andras Moritz, Chair

Israel Koren, Member

Mani Krishna, Member

C.V. Hollot, Department Chair
Electrical and Computer Engineering

To my parents and my sister

ACKNOWLEDGMENTS

I would like to first thank my parents and my sister for believing in me and motivating me all these years. Next, I express my sincere gratitude to my advisor Professor Csaba Andras Moritz for his valuable advice, encouragement and suggestions. He has helped me navigate through the tough waters of my research and this thesis would not have been possible without his constant motivation. I also thank Professor Israel Koren and Professor Mani Krishna for being part of the committe and providing invaluable insights and suggestions regarding my thesis work. I would also like to thank all my lab mates and my friends for all their feedback and fruitful discussions during my time at Amherst. Finally, I thank the almighty for providing me this good opportunity.

ABSTRACT

MULTI-VALUED MAJORITY LOGIC CIRCUITS USING SPIN WAVES

SEPTEMBER 2013

SANKARA NARAYANAN RAJAPANDIAN

B.E, COLLEGE OF ENGINEERING GUINDY, ANNA UNIVERSITY, INDIA

M.S.E.C.E., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Csaba Andras Moritz

With increasing data sets for processing, there is a requirement to build faster and smaller arithmetic circuits. One of the ways to improve the performance of higher order arithmetic units is to reduce the carry propagation levels. Multi-valued logic enables this by reducing the number of digits required to represent a range of numbers. Area reduction is also obtained through fewer operations and signals required to realise a function.

Though theoretically multi-valued logic has these advantages, implementation of the multi-valued logic using CMOS has not been efficient. The main reason is because multi-valued logic is emulated in CMOS using binary switches. Two main approaches are followed in CMOS in implementing multi-valued logic using CMOS. Voltage mode logic, where the logic states are encoded using the node voltages suffers from low noise margins and limitation of radix due to the power supply. Current mode logic, where the branch currents are used to represent the logic levels suffers from high power

consumption due to static current flow and requirement of restoration devices. The mindset of the post-CMOS approaches explored so far for multi-valued logic circuit design has been to replace the CMOS switches with their novel nano switches. Hence, they too suffer from the same issues as CMOS implementation.

Our value proposition is through the use of a truly multi-state device based on electron spin. Spin waves, which are a collection of electron spins of an atom enables multi-valued logic by allowing encoding information in the amplitude and phase of the wave. Another advantage of the spin wave fabric is that the computation is through wave propagation and interference which does not involve any movement of charge. This enables building low energy, smaller and faster multi-valued circuits. In this thesis, implementation of the basic building blocks of multi-valued logic using these novel spin wave based devices is shown. Building of arithmetic circuits like adders using these building blocks have also been demonstrated. To quantify the benefits of spin wave based multi-valued circuits, they are benchmarked with CMOS. For 32-bits, our projected comparisons show a 5X increase in performance, 125X area improvement and 1717X power reduction for hexa-decimal spin wave based adders compared to binary CMOS. Similarly, there is a 4X increase in performance of hexa-decimal SPWF multiplier compared to CMOS for 16-bits. Finally, we have implemented the I/O circuits for smooth interface between binary CMOS and multi-valued SPWF logic.

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION AND MOTIVATION

Historically, arithmetic circuits have been built using binary logic where two logic levels were used for representing and manipulating a range of numbers. Higher order arithmetic circuits like adders and multipliers built using binary logic suffer from increased carry propagation levels which increases the delay and the area. One of the attractive solutions to solve this problem is to use higher number of logic levels. Multi-valued logic involves using more than two logic levels and hence allows store more information compared to binary in a single digit. Due to the compressed data representation, the logic area required for implementation a given function is reduced due to the fewer operations. The area of the interconnections is also lowered since fewer signals are now required to represent a range of numbers compared to the binary logic style . The performance of the arithmetic circuits is also greatly increased due to the reduced carry propagation stages.

The first major work on multi-valued algebra was developed by Emil Post in 1920 [6]. Exploration of multi-valued logic circuit design started in the 1950s [6]. Minimization techniques to improve the implementation of multi-valued algebra was developed in 1960s [6]. Today multi-valued implementation of almost all binary logic circuits are available. Though these developments were made, multi-valued circuits in CMOS is not popular since the implementation in CMOS is inefficient. This is because the multi-valued logic is emulated using binary switches in CMOS.

Multi-valued logic is implemented in CMOS in two ways. The first method is by using node voltages to represent the logic levels, also referred to as the voltage mode

logic style. One of the main issues in the voltage mode logic is that the maximum radix that is achievable is limited by the power supply voltage. With the technology scaling, the voltage has also been scaled down which restricts the number of logic levels achievable. Voltage mode logic also suffers from having lower noise margins due to the smaller voltage boundaries between the logic levels [14]. Therefore they are more prone to effects of noise and consequently produce functional errors.

Second and the most popular implementation style in CMOS is through the current mode logic, where the logic levels are represented by the branch currents. It is more popular due to the ease in realizing the addition operation through connecting the branch currents in to a single node. The major drawback in the current mode logic is the presence of static current [5]. This results in an increased power consumption. Additionally, the current mode logic is not self-restoring. Accordingly, restoration circuits are required which consume area and power [14]. Another disadvantage is that the transistor sizes are determined by the threshold levels of current which reduces the performance [10].

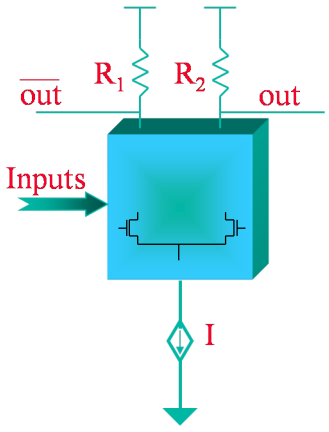

Figure 1.1. Current Mode Logic [5]

The mindset of post-CMOS device research for multi-valued logic is to replace the CMOS switch with their novel smaller/faster nano scale switches [1]. As a result they also face the same issues as CMOS in multi-valued circuit design.

Our value proposition is the use of a novel nano-scale device based on electron spin which inherently supports multi-valued logic. It has multi states which can be used to represent the multi-valued logic. The next big advantage of our approach is that the computation does not involve any charge transfer which enables low power implementation of the multi-valued logic circuits. Thus, we can build faster, compact and low power arithmetic circuits using multi-valued logic.

The key contributions of this thesis are:

1. Implementation of multi-valued logic building blocks using novel nano scale devices based on electron spin

2. Design of multi-valued arithmetic circuits using the multi-valued logic building blocks

3. Benchmarking the multi-valued arithmetic circuits with binary CMOS

4. I/O circuits for converting from binary to r-ary and vice-versa

The rest of the thesis is organized as follows. We introduce the novel nano device and the fabric components. Also, the multi-valued logic encoding is presented in Chapter 2. We provide a background in multi-valued algebra explaining the various operators required in Chapter 3. Chapter 4 describes the implementation of the multi-valued logic operators using spin waves. It also has the benchmarking results of the spin wave implementation with the CMOS version. A brief overview of representing and minimizing multi-valued functions using multi-valued operators is provided in Chapter 5. Building of multi-valued adders using the multi-valued logic operators and their evaluation with CMOS in Chapter 5. Chapter 6 discusses the implementation of multi-valued multipliers and presents the comparison with CMOS for the same. I/O logic circuits using SPWFs for converting from binary to r-ary logic and vice-versa have been explained in Chapter 7. Chapter 8 concludes the thesis.

CHAPTER 2

SPIN WAVE

In this chapter we give a brief background on spin waves and the fabric components. Then we also show the multi-valued logic state encoding using spin waves

A spin wave is a collective oscillation of electron spins in an ordered spin lattice around the direction of magnetization [8]. Information can be encoded in the phase and amplitude of the spin wave. Transmission of information is achieved using wave propagation which does not involve any movement of charges.

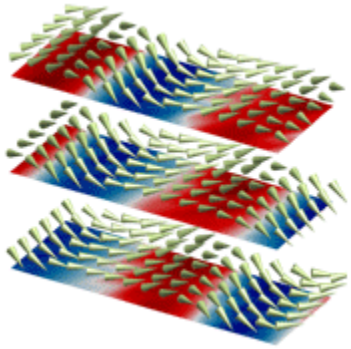

Figure 2.1. Spin Wave [13]

2.1 Fabric components

There are two main components in the spin wave fabric. They are the Spin Wave Bus (SWB) and Magneto-Electric (ME) cell as shown in Figure 2.2. Magneto-Electric cell performs the function of coupling the electrical domain input to the magnetic domain. The amplitude and phase of the input spin wave generated can

be controlled through the input voltage. Additionally, the ME cell also functions as a non-volatile storage device of the spin waves. It has also been shown that the amplification of the spin wave can be done by using the ME cells. Finally, the ME cell also performs the read out mechanism and can convert the output spin wave to the corresponding electrical output. Computation happens in the spin wave bus through propagation and wave interference.

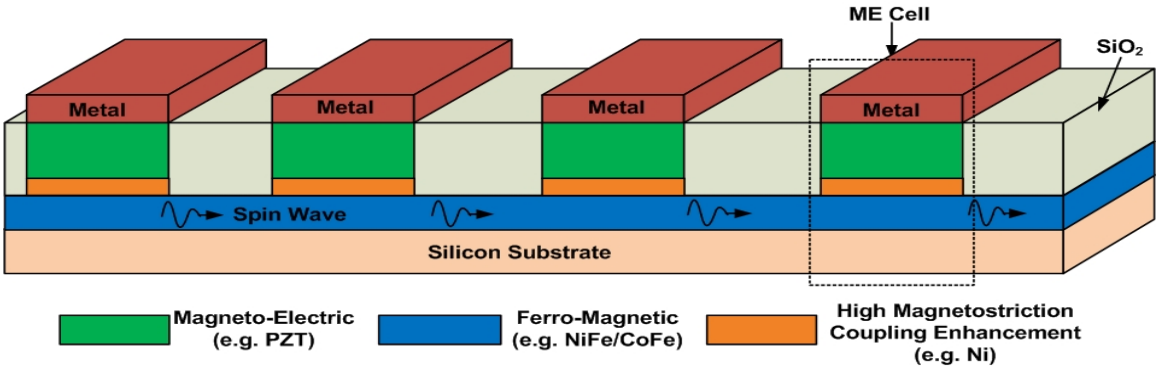

Figure 2.2. Spin wave fabric components - MagnetoElectric(ME) cell and Spin Wave Bus (SWB) [15]

2.2 Multi-valued logic using spin waves

Spin waves of different amplitude and phase can be generated by varying the input voltage. Multi-valued logic is enabled by encoding information in the combination of multiple amplitudes and phases of the spin wave. For this work, we assume that only two phases(phase 0 and phase π) of spin waves are available. Hence to represent a radix r we need r/2 amplitudes if r is even and (r+1)/2 amplitudes if r is odd. For example, to represent quaternary logic, we require two amplitudes. Table 2.1 shows the quaternary logic state representation using spin waves.

	Amplitude 3A	Amplitude A
Phase 0	Logic 0	Logic 1
Phase π	Logic 3	Logic 2

Table 2.1. Quaternary logic encoding using spin waves

2.3 Chapter Summary

A brief overview of spin waves and the multi-valued logic representation using spin waves is shown in this chapter. We discuss about the multi-valued logic operators in the next chapter.

CHAPTER 3

MULTI-VALUED LOGIC OPERATORS

In this chapter, we provide a background in multi-valued logic algebra and the operators required for implementing multi-valued functions.

3.1 Multi-valued logic algebra

One of the main advantages of adopting an approach of developing an algebra is that the operators of the algebra have simple and efficient circuit implementation [9]. This is similar to the concept of building effective operators AND, OR and NOT for realising binary functions. Thus multi-valued algebra provides a framework for expressing and manipulating multi-valued functions [9]. It is identical to the manipulation of binary functions through boolean algebra.

A multi-valued function $f(x)$ has multi-valued inputs and outputs. A r-valued, n-variable function $f(x_0, x_1, ..., x_n)$ [14] can be defined as the mapping $f : R^n -> R$, where set R = {0, 1, 2, ... r-1} and $x_i \epsilon$ R. For a given radix r and n inputs, the number of possible functions are r^{r^n} [12]. We can see that there is an explosion in the number of functions with the increase in radix r. Table 3.1 shows the number of possible two variable functions for radix 2 and 3.

r	r^{r^n}
2	16
3	19683

Table 3.1. Number of possible two variable functions

Since we cannot define and implement all the functions, we need a functionally complete algebra. A set of operators is said to be functionally complete if it can be used to realize any arbitrary function. A set of {AND, NOT} operators are functionally complete for boolean algebra. Similarly it has been proved that for multi-valued algebra a set of {MIN, MAX, LITERAL} operators which are defined below are functionally complete [2]. To implement the above operators efficiently using circuits with sum as the basic operation we require some more operators like identity, complement, truncated difference, upper and lower threshold [16] [11].

3.2 Multi-valued logic operators

In this section, we describe the various multi-valued logic operators.

3.2.1 Identity

The *identity* operator x is defined as x, where $x \epsilon$ R. The truth table of the *identity* operator for quaternary logic is given in Table 3.2

x	0	1	2	3
x	0	1	2	3

Table 3.2. Truth table of *identity* operator

3.2.2 Complement

The *complement* operator $\bar{x}$ is defined as $r - 1 - x$, where $x \epsilon$ R. The truth table of the *complement* operator for quaternary logic is given below

x	0	1	2	3
$\bar{x}$	3	2	1	0

Table 3.3. Truth table of *complement* operator

3.2.3 Upper threshold

The *upper threshold* operator x_a^{r-1} is defined as

r-1 when $x >=$ a

0 else, where $x \epsilon$ R

Table 3.4 shows the truth table of the *upper threshold* operator for quaternary logic

x / a	0	1	2	3
0	3	3	3	3
1	0	3	3	3
2	0	0	3	3
3	0	0	0	3

Table 3.4. Truth table of *upper threshold* operator

3.2.4 Lower threshold

The *lower threshold* operator ${}_a^{r-1}x$ is defined as

r-1 when $x <=$ a,

0 else where $x \epsilon$ R.

The truth table of the *lower threshold* operator for quaternary logic is given by Table 3.5

x / a	0	1	2	3
0	3	0	0	0
1	3	3	0	0
2	3	3	3	0
3	3	3	3	3

Table 3.5. Truth table for *lower threshold* operator

3.2.5 Window literal operator

The *window literal* operator ${}^{a}x^{b}$ is defined as

r-1 when a <= x <= b,

0 else where $x \epsilon$ R.

The *window literal* operator becomes the upper threshold operator when b becomes r-1. Similarly the window literal operator becomes the lower threshold operator when a becomes 0. Therefore the *window literal* operator can be built from the *upper* and *lower threshold* operators.

3.2.6 Truncated Difference operator

The *truncated difference* operator $x \Xi y$ is defined as

$x - y$ if $x > y$

0 else, where $x, y \epsilon$ R

y \ x	0	1	2	3
0	0	1	2	3
1	0	0	1	2
2	0	0	0	1
3	0	0	0	0

Table 3.6. Truth table of *truncated difference* operator

3.2.7 Min operator

The *min* operator $x.y$ is defined as

x if $x < y$

y else , where $x, y \epsilon$ R

The *min* operator is similar to the *AND* operator for the binary logic.

y \ x	0	1	2	3
0	0	0	0	0
1	0	1	1	1
2	0	1	2	2
3	0	1	2	3

Table 3.7. Truth table of *min* operator

3.2.8 Max operator

The *max* operator $x + y$ is defined as

x if $x > y$

y else, where $x, y \epsilon$ R

y \ x	0	1	2	3
0	0	1	2	3
1	1	1	2	3
2	2	2	2	3
3	3	3	3	3

Table 3.8. Truth table of *max* operator

The *OR* operator is equivalent to the *max* operator for binary logic .

3.3 Chapter Summary

Operators of the multi-valued algebra were described in this chapter. The next chapter shows the implementation of the operators using spinwaves.

CHAPTER 4

IMPLEMENTATION OF MULTI-VALUED OPERATORS USING SPIN WAVES

In this chapter, we present the implementation details of MVL operators using spin waves. The basic building block which consists of a spin wave bus is called the spin wave function (SPWF)

4.1 SPWF operator : Identity

Spin Wave Bus (SWB) of lengths equal to integral multiple of the spin wavelength produces output with same phase and amplitude of the input. Figure 4.1 shows this implementation of the *identity* operator x with a simple spin wave bus.

Figure 4.1. SPWF based *identity* operator

4.2 SPWF operator : Complement

Phase inversion of a given input spin wave is equal to its complement. Spin Wave Bus (SWB) of lengths equal to integral multiple of half of the spin wavelength can produce phase inversion and can implement the *complement* operator $\bar{x} = r - 1 - x$

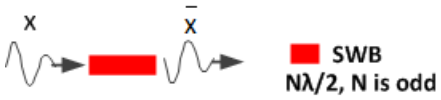

Figure 4.2. SPWF based *complement* operator

4.3 SPWF operator : Upper threshold

The *upper threshold* operator x_a^{r-1} defined as

r-1 when x >=a

0 else, where x ϵ R

It can be implemented using the following circuit Maj [(r-1) Maj [$x, \bar{a}$, r/2]] where (r-1) represents r-1 copies or amplification.

Maj [$x, \bar{a}$, r/2] produces an output wave of positive phase when x >=a while a negative phase wave is generated otherwise. To obtain the right amplitudes, an amplification cell is used to pull up to the highest logic level or pull down to the lowest logic level. Figure 4.3 shows the implementation of the *upper threshold* operatorfor quaternary logic.

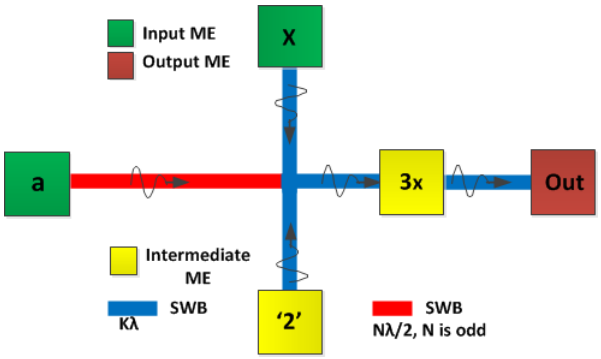

Figure 4.3. SPWF based *upper threshold* operator (Quaternary logic)

If there is no amplification, then the desired amplitude can be obtained by adding r-1 copies of the gate Maj [$x, \bar{a}$, r/2] as shown in Figure 4.4

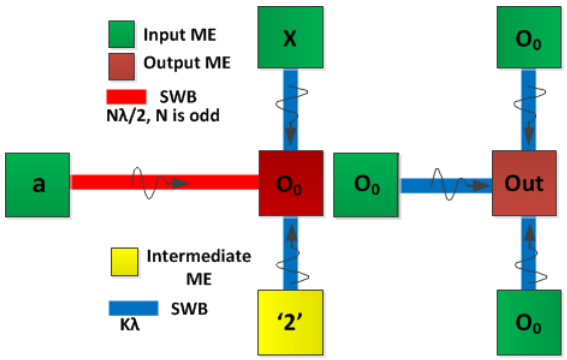

Figure 4.4. SPWF based *upper threshold* operator without amplification (Quaternary logic)

4.4 SPWF operator : Lower threshold

The *lower threshold* operator ${}_{a}^{r-1}x$ defined as

r-1 when x <=a

0 else, where $x \epsilon$ R

It can be realized similar to the upper threshold operator with x and a swapped.

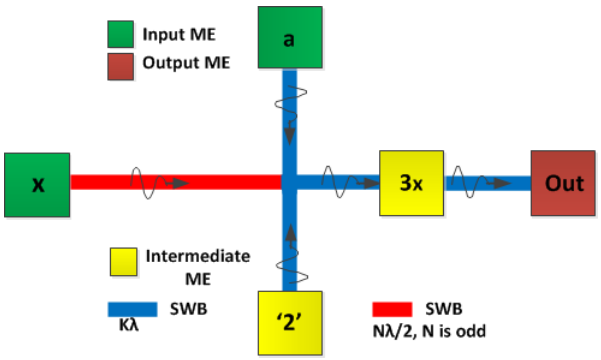

Figure 4.5. SPWF based *lower threshold* operator (Quaternary logic)

4.5 SPWF operator : Window literal

The *window literal* operator ${}^{a}x^{b}$ is defined as

r-1 when a <= x <= b,

0 else where $x < \epsilon$ R.

The *window literal* operator can be built from the *upper* and *lower threshold* operators. Maj$[x_a^{r-1}$, $_b^{r-1}x$, 0] implements the *window literal* operator from the *upper* and *lower threshold* operators.

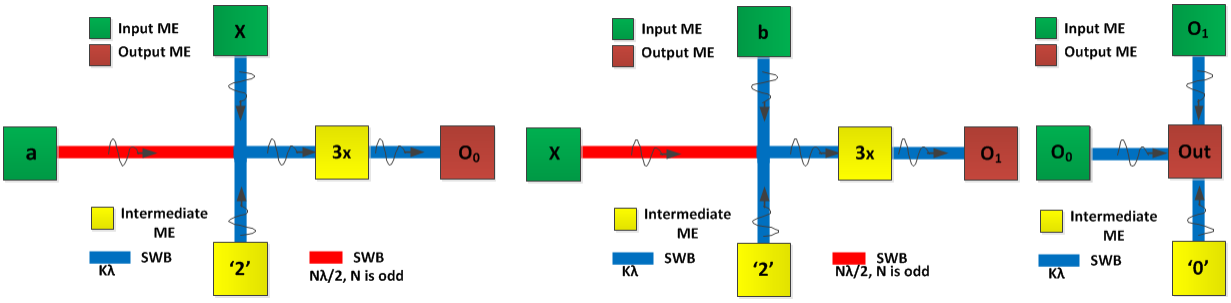

Figure 4.6. SPWF based *window literal* operator (Quaternary logic)

4.6 SPWF operator : Truncated difference

The *truncated difference* operator $x \Xi y$ defined as

$$x - y \text{ if } x > y$$

$$0 \text{ else, where } x, y \, \epsilon \, \mathrm{R}$$

Maj $[x, \bar{y}$, 0] implements the spin wave based *truncated difference* operator.

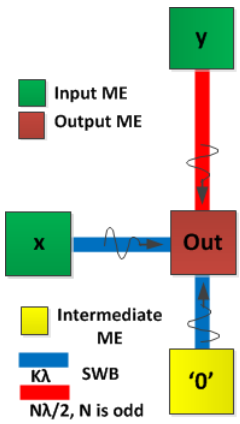

Figure 4.7. SPWF based *truncated difference* operator (Quaternary logic)

4.7 SPWF operator : Min operator

The *min* operator $x.y$ is defined as

$$x \text{ if } x < y$$

$$y \text{ else, where } x, y \; \epsilon \; \mathrm{R}$$

The *min* operator can be implemented using the *truncated difference* operator. $x.y = x\Xi(x\Xi y)$. This can be implemented through Maj [x, inv($x\Xi y$), 0]. Output of the circuit is

$$x - (x - y) \text{ if } x > y \text{ ,}$$

$$x \text{ else}$$

The circuit layout for quaternary logic is shown in Figure 4.8

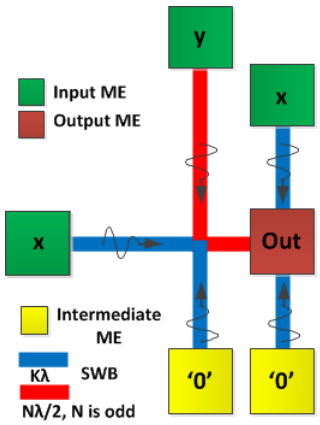

Figure 4.8. SPWF based *min* operator (Quaternary logic)

4.8 SPWF operator : Max operator

The *max* operator $x + y$ is defined as

$$x \text{ if } x > y$$

$$y \text{ else, where } x, y \; \epsilon \; \mathrm{R}$$

The *max* operator is also implemented using the *truncated difference* operator. $x + y = x + (y\Xi x)$. This can be implemented through Maj[x, ($y\Xi x$), r-1]. Output of the circuit is

$$x + (y - x) \text{ if } y > x \ ,$$

$$x \text{ else}$$

Figure 4.9 is the circuit layout for a quaternary logic *max* operator

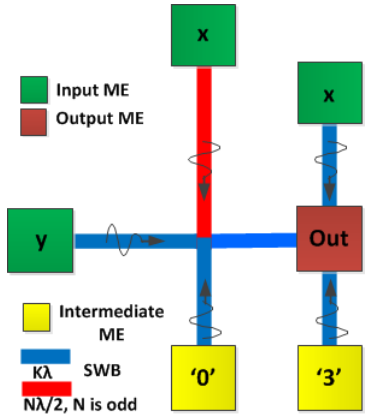

Figure 4.9. SPWF based *max* operator (Quaternary logic)

4.9 Projected Comparisons vs. CMOS 90nm

In this section, the projected results of comparison between the SPWF and CMOS 90nm implementation of the MVL operators are shown. The methodology used for evaluation is also explained.

4.9.1 SPWF methodology

For comparison, the flavours of SPWF circuits with and without amplification is assumed. Also, the comparison is also made for cases with and without I/O delays. This is to make sure that the cases where the input signals are available from previous stages are also covered.

For evaulation, ME cell dimensions used are 100nmx100nm. All the layouts of the ME cell are taken to be circular. The wavelength of the spin wave is assumed to be 100nm. The delay of the circuit is calculated as the sum of ME cell switching delay and propagation delay of the spin wave bus. The group velocity of the spin

waves is assumed to be 10^4 m/s. The switching delay of the ME cell is taken to be 100ps. The propagation of the spin waves does not involve any movement of charge and hence there is no energy consumed for the propagation and the interference of the wave. ME cell switching is presumed to consume 10aJ. Accordingly, the total energy consumed by the circuit depends on the total number of ME cells that are switching. The area, delay and power of the SPWF circuits was calculated with the above assumptions.

For CMOS, the values of power and delay were obtained from [1]. Table 4.1 shows the comparison of threshold operators with SPWF implementation compared with CMOS. Comparisons for max and min operators can be found in Table 4.2 and Table 4.3 respectively.

Threshold operators	Power (μW)	Delay(ps)
CMOS	364	7.47
SPWF (with amplification and I/O ME)	335	0.15
SPWF (with amplification and without I/O ME)	135	0.07
SPWF (without amplification and with I/O ME)	225	0.4
SPWF (without amplification and I/O ME)	25	0

Table 4.1. Projected comparisons of SPWF threshold operators vs. CMOS 90nm

Max operator	Power (μW)	Delay(ps)
CMOS	512	17.22
SPWF (with I/O ME)	225	0.3
SPWF (without I/O ME)	25	0

Table 4.2. Projected comparisons of SPWF max operator vs. CMOS 90nm

Min operator	Power (μW)	Delay(ps)
CMOS	512	17.22
SPWF (with I/O ME)	220	0.3
SPWF (without I/O ME)	20	0

Table 4.3. Projected comparisons of SPWF min operator vs. CMOS 90nm

4.10 Chapter Summary

We have presented and evaluated the SPWF implementation of Multi-Valued Logic(MVL) operators with CMOS in this chapter. Our initial projections for threshold operators show that the SPWF implementations with amplification has 50X power reduction for the same performance. There is a speedup of 1.5X with 19X reduction in power for implementations without amplification. The benefits are larger for the cases without I/O ME cells. For the threshold operators with amplification there is a increased performance of 2.7X along with 106X power reduction. For the implementations without amplification the speed up is 15X without any power consumption.

For the min and max operators, there is a speed up of 2.3X with 57X power reduction for SPWF implementation compared with CMOS. Ignoring the I/O ME cells the increase in performance is 20X and there is no power consumption. Thus, we can see that we have huge power benefits for similar or increased performance compared to CMOS for all the different SPWF scenarios.

In the next chapter, we discuss the functional representation of the multi-valued functions and implementation of multi-valued arithmetic circuits.

CHAPTER 5

MULTI-VALUED ADDERS

In this chapter, we start with showing how to represent and minimize the multi-valued functions using the multi-valued operators. Then implementation of multi-valued adders using spin waves is described.

5.1 Multi-valued function representation

There are three methods to represent the multi-valued functions [3]. They are

1. Sum of Products (SOPs)

2. Multi-valued Networks

3. Multi-valued Decision Diagrams (MDDs)

Sum of Products (SOPs) is similar to the two level sum of product representation of binary functions. A multi-level network of nodes , where each node is a two level multi-valued sum of products constitutes Multi-valued Networks. Multi-valued Decision Diagrams are analogous to binary decision diagrams, except that each node has multi-valued children instead of having just two. For this work, we have chosen the sum of products representation, since it is inherently simple and the most preferred representation in the MVL circuit design works. Another reason is that the minimization techniques using sum of products representation have been extensively studied.

5.1.1 Sum of product representation

A product term P is expressed as P = k.${}^{a1}x_1^{b1}.{}^{a2}x_2^{b2}....{}^{an}x_n^{bn}$, where k is a constant and k ϵ 1, 2, ..., r-1 [7]. P can also be expressed as P = min (k, ${}^{a1}x_1^{b1}.{}^{a2}x_2^{b2}....{}^{an}x_n^{bn}$). Accordingly a product term consists of min operation on a set of literal operators and constants. The sum is the max operation (+) of all the product terms. Hence any arbitrary multi-valued function $f(x)$ can be expressed in SOP form as

$f(x) = P_1 + P_2 + ... + P_i$, where P_i represents the product term

As an example let us take an arbitrary quaternary function $f(x_1, x_2)$ in two variables and try to represent it in sum of products form. The function is chosen so that it will be easy to demonstrate the minimization of multi-valued functions using multi-valued operators. The truth table of the function is shown in Table 5.1

x_1 \ x_2	0	1	2	3
0	1	1	3	2
1	1	1	3	2
2	2	2	3	2
3	0	0	3	2

Table 5.1. Arbitrary function f(x_1, x_2) truth table(Quaternary logic)

The product terms consist of the non zero entries in the truth table. The first product term for $x_1 = 0$ and $x_2 = 0$ is $1.{}^0x_1^0.{}^0x_2^0$. In the same way the other product terms can be written. To simplify the notation, if a=b in a window literal operator ${}^ax^b$ then it is represented as x^a. With the simplified notation,

$f(x_1, x_2) = 1.x_1^0.x_2^0 + 1.x_1^0.x_2^1 + x_1^0.x_2^2 + 2.x_1^0.x_2^3 + 1.x_1^1.x_2^0 + 1.x_1^1.x_2^1 + x_1^1.x_2^2 + 2.x_1^1.x_2^3 + 2.x_1^2.x_2^0 + 2.x_1^2.x_2^1 + x_1^2.x_2^2 + 2.x_1^2.x_2^3 + x_1^3.x_2^2 + 2.x_1^3.x_2^3$

Minimization of the above function can be done using the property of the literal operators [4] [14]. One of the property used for minimization is

Property: If a and b are two constants such that a <= b and a, b ϵ R . Then

$$a.x_1^i + b.x_2^j = a.(x_1^i + x_2^j) + b.x_2^j$$

The minimization technique is similar to K-Map minimization for Boolean logic. We carry out the minimization for each non-zero entry separately. The zero entries are omitted during minimization.

Step 1 : Minimization for '1'

Using the theorem above, entries for 2 and 3 can be made as dont cares. This results in a truth table shown in Table 5.1.

x_1 \ x_2	0	1	2	3
0	1	1	X	X
1	1	1	X	X
2	X	X	X	X
3			X	X

Figure 5.1. Minimization for '1'

After minimization, the result is $1.^0x_1^1$.

Step 2 : Minimization for 2

Similar to step 1, the entries for 3 are dont cares and the resulting truth table is Table 5.3.

x_1 \ x_2	0	1	2	3
0			X	2
1			X	2
2	2	2	X	2
3			X	2

Figure 5.2. Minimization for '2'

After minimization, the result is $2.x_1^2 + 2.x_2^3$.

Step 3 : Minimization for 3

The entries of 1 and 2 are omitted and the truth table is reduced to

x_1 \ x_2	0	1	2	3
0			3	
1			3	
2			3	
3			3	

Figure 5.3. Minimization for '3'

After minimization, the result is x_2^2.

The final minimized SOP form for the function f(x_1, x_2) = $1.^0x_1^1 + 2.x_1^2 + 2.x_2^3 + x_2^2$. This requires 4 window literal operator gates, 3 min gates and 3 max gates. Accordingly, any arbitrary multi-valued function can be implemented using spin wave logic with simple SPWF based min, max and literal operator gates.

5.2 Quaternary half adder

We look at the implementation of quaternary half adder function. The quaternary half adder has two quaternary inputs (A, B). There is one quaternary output (Sum) and a binary (Carry) output. The truth table for the sum and carry outputs are shown in Table 5.2 and Table 5.3 respectively.

A \ B	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

Table 5.2. Sum output truth table(Quaternary logic)

After minimization,

Sum = $1.A^0.B^1 + 1.A^1.B^0 + 1.A^2.B^3 + 1.A^3.B^2 + 2.A^0.B^2 + 2.A^2.B^0 + 2.A^1.B^1 + 1.A^3.B^3 + A^0.B^3 + A^3.B^0 + A^2.B^1 + A^1.B^2$.

B / A	0	1	2	3
0	0	0	0	0
1	0	0	0	1
2	0	0	1	1
3	0	1	1	1

Table 5.3. Carry output truth table(Quaternary logic)

This requires 24 window literal, 20 min and 11 max operators. Similarly,

Carry = $1.^2A^3.^2B^3 + 1.A^3.B^1 + 1.A^1.B^3$

For implementation, 6 window literal, 6 min and 2 max operators are needed. The number of operators required for implementing a Sum of the half adder requires too many operators.

One of the main reason for inefficient implementation is due to the limited number of operators min, max and literal. Symmetric functions can be implemented more efficiently using cyclic operators [17].

5.2.1 Cyclic operator

The cyclic operator also called the mod sum operator $x \oplus y$ is defined as

$(x +_{add} y\ )$ mod r, where $+_{add}$ represents arithmetic addition.

The mod sum operator is similar to the XOR gate of the binary logic. To implement the mod sum operator, we define a new operator- carry operator $+_{carry}$. It is defined as

1 if $x +_{add} y >$ r-1

0 else

Let us see the implementation of the above two operators using spin waves.

5.2.2 SPWF operator : Carry

The carry operator can be implemented as Min (Maj $[x, y, 0]$, 1). The output of the Maj $[x, y, 0]$ circuit is

$x +_{add} y$ - r-1 if $x +_{add} y >$ r-1

0 else

Therefore, we obtain a non zero output only when $x +_{add} y >$ r-1 . Min operation with '1' provides the binary output.

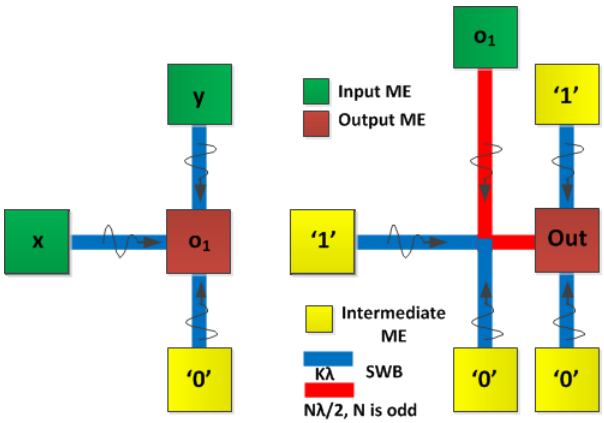

Figure 5.4. Carry operator SPWF implementation (Quaternary logic)

5.2.3 SPWF operator : Mod sum

Maj $[x, y, 0, ({}_{r}^{r-1}(x +_{add} y))$, inv$(x +_{carry} y)]$ implements the mod sum $x \oplus y$ operator. $({}_{r}^{r-1}(x +_{add} y))$ implements a lower threshold operation with $x +_{add} y$ as input. The output of this lower threshold operation is

r-1 , if $x +_{add} y <=$ r-1

0 else

Table 5.4 is a sample truth table of SPWF based mod sum $x \oplus y$ operator.

x	y	'0'	$\overset{r-1}{r}(x +_{add} y)$	$\text{inv}(x +_{carry} y)$	$x \oplus y$
0	0	0	3	3	0
1	1	0	3	3	2
2	2	0	0	2	0
3	3	0	0	2	2

Table 5.4. A sample truth table of SPWF based mod sum operator(Quaternary logic)

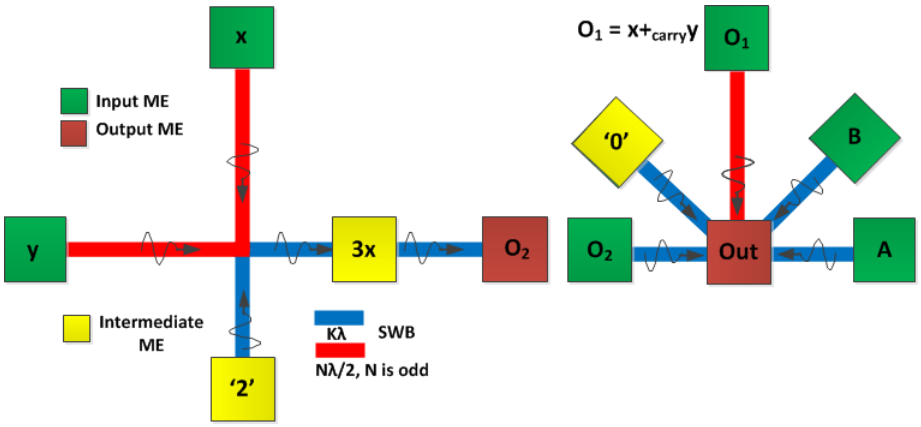

Figure 5.5. Sum operator SPWF implementation (Quaternary logic)

5.3 Multi-Valued full adders

A multi-valued full adder has a binary carry in in addition to the two multi-valued addends. Let us look at how to implement a quaternary full adder.

5.3.1 Quaternary full adder

It consists of two quaternary inputs (A, B) and one binary input (C_{in}). The implementation can be carried out using the half adder sum and carry circuits. The binary carry output can be implemented using two half adder carry circuits and a max gate. Carry output = max ($(A +_{carry} B), (A \oplus B) +_{carry} C_{in})$). In the same way, the quaternary sum output can be implemented using two half adder sum circuits. Sum output = ($(A \oplus B) \oplus C_{in}$)

To implement the full adders even more efficiently, we need carry and mod sum operators with three inputs - two addends and a carry in.

5.3.2 SPWF operator : Carry (3 inputs)

The carry out operator can be implemented using carry operator with three inputs (A $+_{carry}$ B $+_{carry}$ C_{in}) in a single step. Min (Maj [A, B, C_{in}] , 1) implements the carry out operation. The output of the circuit is

1 if A $+_{add}$ B $+_{add}$ C_{in} > r-1 ,

0 else

Figure 5.6 shows the layout for the carry operator implementation with three inputs.

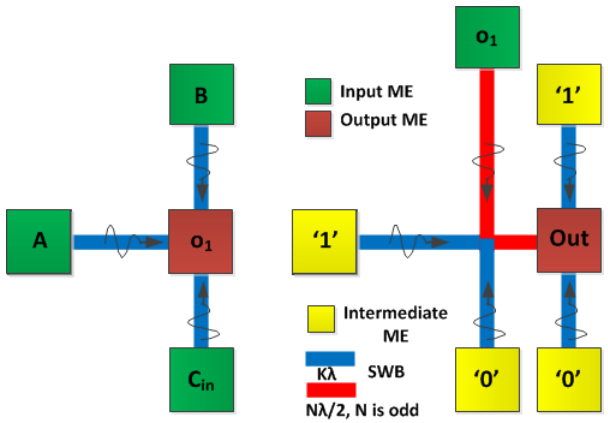

Figure 5.6. SPWF based carry operator (3 inputs) (Quaternary logic)

5.3.3 SPWF operator : Mod sum (3 inputs)

Similar to the carry output of the full adder, the sum output of the full adder can also be realized more efficiently using a mod sum operator with three inputs (A $\oplus$ B $\oplus$ C_{in}). The circuit implementation is like the one used for the two input mod sum operator. It is realized through Maj(A, B, C_{in},$^{r-1}_{r-1}$(A $+_{add}$B $+_{add}$ C_{in}), inv(A $+_{carry}$ B $+_{carry}$ C_{in}))

The output of the circuit is

$$A +_{add} B +_{add} C_{in} \text{ - r if } A +_{add} B +_{add} C_{in} > \text{r-1 ,}$$

$$A +_{add} B +_{add} C_{in} \text{ else}$$

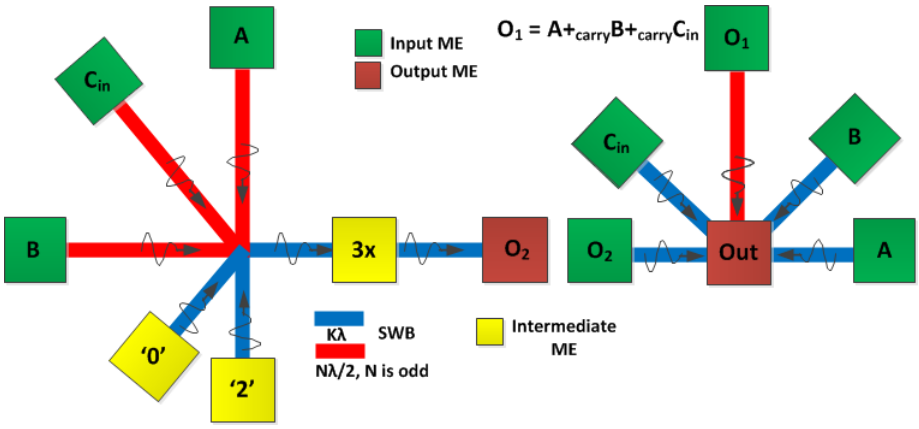

Figure 5.7. SPWF based mod sum operator (3 inputs) (Quaternary logic)

Comparison of the number of ME cells required for the implementation of carry and sum output of the quaternary logic is shown in Table 5.5.

	Min, max and literal	Mod sum (2 inputs)	Mod sum (3 inputs)
Carry	270	26	10
Sum	660	22	13

Table 5.5. Comparison of the ME cells required for sum and carry output realization using various operators (Quaternary logic)

Hence, three input mod sum and carry operators provide the most efficient implementation of SPWF based multi-valued full adders.

5.4 Projected comparisons vs.45nm CMOS

To compare binary and multi-level logic , we implement 4, 8, 16 and 32-bit Full Adder(FA) in binary, quaternary and hexa-decimal SPWF logic using ripple carry style. For evaluating the spin wave based multi-valued full adder implementation, the methodology described for evaluating the multi-valued operators before was used her also. CMOS versions for the adders are synthesized with NCSU 45nm technology using design compiler. For the multi-valued full adders, the implementation is assumed with three input carry and mod sum operators. For the binary SPWF full adder, the comparison is performed with simplified carry and mod sum operators, Carry out = Maj(A, B, C_{in}) and Sum = Maj(A, B, C_{in}, (2)C_{outb}). The I/O delays are ignored for both SPWF and CMOS versions. The graphs for the projected comparisons in terms of delay, area and power between CMOS and SPWF implementations are in Figure 5.8, Figure 5.9 and Figure 5.10.

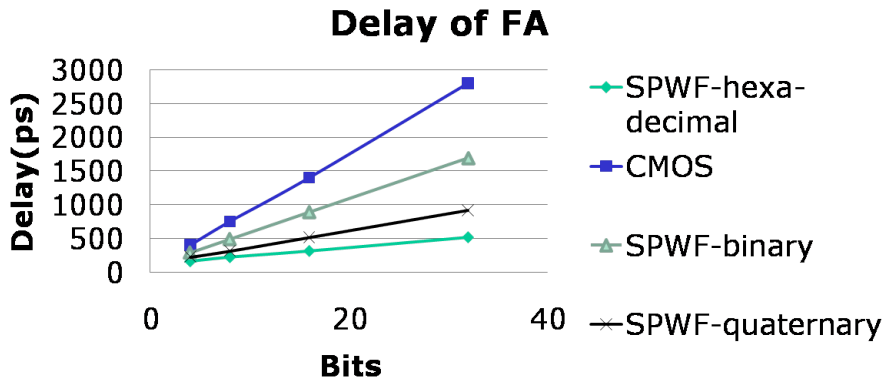

Figure 5.8. Projected comparisons vs. CMOS (45nm) for full adders-Delay

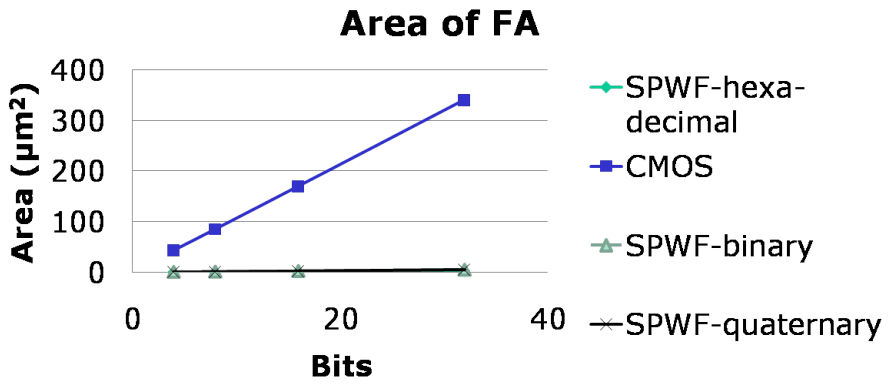

Figure 5.9. Projected comparisons vs. CMOS (45nm) for full adders-Area

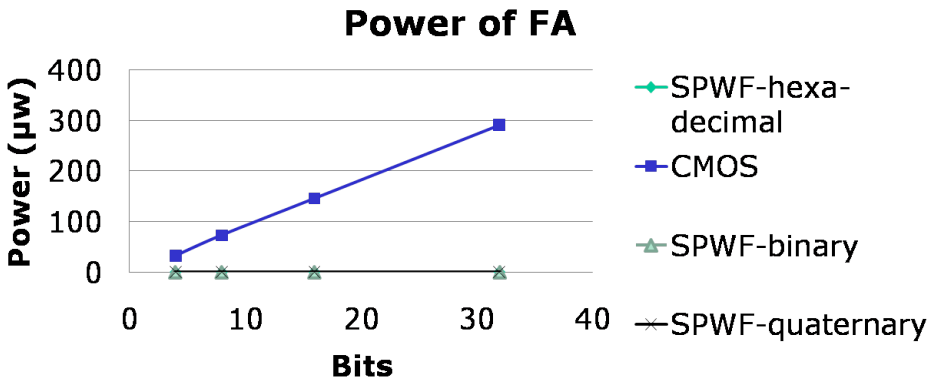

Figure 5.10. Projected comparisons vs. CMOS (45nm) for full adders-Power

5.5 Chapter summary

We can see that the performance of the adder increases with increase in logic level for the SPWF implementation. There is a 5.4X increase in performance of hexa-decimal SPWF FA compared to CMOS for 32 bits. Area is also shown to reduce with logic level increment . We have a 125X area reduction with hexa-decimal SPWF FA compared to CMOS for 32 bits. The power consumption reduces with increase in logic level. Hexa-decimal SPWF full adder consumes 1717X less power compared to CMOS for 32-bits. Thus, in this chapter we have shown how to represent the multi-valued functions using multi-valued operators. We have also introduced new operators for implementing the multi-valued full adders more efficiently. Benchmarking of the full adders with CMOS has also been done in this chapter. The next chapter is about the implementation of multi-valued multipliers.

CHAPTER 6

MULTI-VALUED MULTIPLIERS

In this chapter, we explore the implementation of SPWF based multi-valued multipliers.

For implementing multipliers, parallel multiplication algorithm is selected since it enables faster multiplication. In this algorithm, the generation and accumulation of all the partial products is performed in parallel.

The basic building blocks of multi digit multi-valued multiplier are shown in Figure 6.1

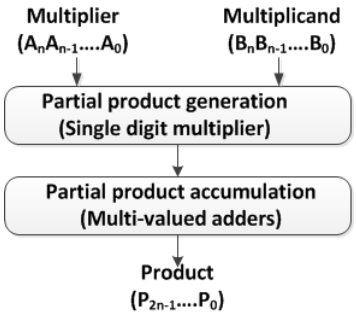

Figure 6.1. Multi digit multi-valued multiplier block diagram

6.1 Multi-valued single digit multiplier

Let us take a look at the implementation of the r-ary single digit multiplier. There is a single digit r-ary multiplicand (B) and a single digit r-ary multiplier (A). Th output product has two digits. The LSB (P_0) output digit is r-ary and the MSB (P_1) output digit is r-1 ary.

6.1.1 Ternary single digit multiplier implementation using min, max and window literal operators

We start by looking at implementation of product function of the ternary multiplier. The truth tables for P_0 and P_1 for ternary logic are shown below

A \ B	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

Table 6.1. Product LSB output truth table(Ternary logic)

A \ B	0	1	2
0	0	0	0
1	0	0	0
2	0	0	1

Table 6.2. Product MSB output truth table(Ternary logic)

After minimization, $P_0 = 1.^1A^2.^1B^2 + A^1.B^2 + A^2.B^1$. This requires 6 window literal, 4 min and 2 max operators. Similarly, $P_1 = 1.A^2.B^2$ which requires 2 window literal and min operators.

6.1.2 Quaternary single digit multiplier implementation using min, max and window literal operators

Next, we look at the implementation of product function for Quaternary logic. Table 6.3 and Table 6.4 represent the product output function for the quaternary logic

After minimization, $P_0 = 1.A^1.B^1 + 1.A^3.B^3 + 2.A^1.B^2 + 2.A^2.B^1 + 2.A^2.B^3 + 2.A^3.B^1 + A^1.B^3 + A^3.B^1$. This requires 16 window literal, 14 min and 7 max operators.

B / A	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	0	2
3	0	3	2	1

Table 6.3. Product LSB output truth table(Quaternary logic)

B / A	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	0	1	1
3	0	0	1	2

Table 6.4. Product MSB output truth table(Quaternary logic)

Similarly, $P_1 = 1.^2A^3.^2B^3 + 2.A^3.B^3$, which requires 4 window literal, 4 min and 2 max operators.

The inference from the implementation of the product function for ternary and quaternary logic is that the number of operations required increases rapidly with increase in radix. Figure 6.2 shows this explosion in the number of operations with increase in radix. Thus implementing the single digit multiplier using only window literal, min and max operators is not efficient.

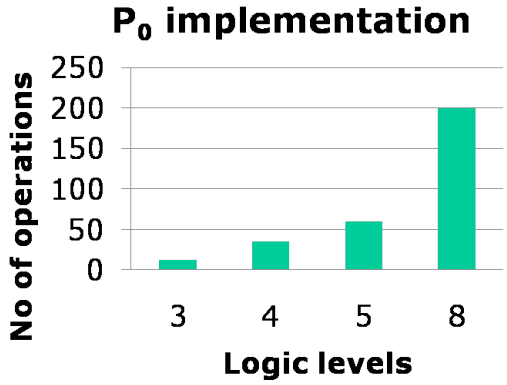

Figure 6.2. Product LSB output implementation requirements for various radix

6.1.3 Multi-valued single digit multipliers using multi-valued adders and multiplexers

One of the alternative method is to compute all the multiples of multiplicand using multi-valued adders and select them using multi-valued multiplexers. For example, we generate 0B, 1B and 2B for ternary logic. Similarly, we generate 0B, 1B, 2B and 3B for quaternary logic. This requires k-1 stages of adders, where 2^k= r if r is even and 2^k= r+1 if r is odd. Then the correct multiple of multiplicand is selected by using the value of each multiplier digit. The multiplexer can be implemented by using r window literal, r min and r-1 max operators. The block diagram of this new scheme is shown in the Figure 6.3

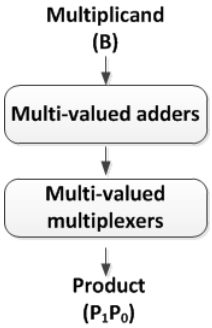

Figure 6.3. Single digit multiplier using multi-valued adders

Table 6.5 shows the comparison of the implementation requirements of a single digit quaternary multiplier for the two schemes.

Quaternary multiplier	Number of ME cells
Min, max and literal based	448
Multi-valued adders based	144

Table 6.5. Comparison of the ME cells required for single digit quaternary multiplier

Thus using the multi-valued adders for generation of partial products provide almost 3X reduction in number of ME cells compared to the sum of product implementation of partial product generation.

6.2 Multi-valued multi digit multipliers

Now using these single digit r-ary multipliers, we can implement n-digit r-ary multipliers. Traditionally, in binary logic the modified booth algorithm is used for faster multiplication. This is due to the fact that the mutliples of multiplicand required for modified booth algorithm can be obtained by just shifting and complementing. Thus fewer multiples of multiplicand are required using modified booth recoding compared to normal multiplication. Let us look at the modified booth encoding for higher radix ($r > 2$) and analyze if we get the same benefits. Table 6.6 shows the modified booth encoding (Y) for ternary logic for two multiplier(A) digits with one reference digit. The multiples of multiplicand (B) required are 0B, ±1B, ±2B, ±3B, 4B, 5B and 6B which is almost same as normal multiplication. Additional stages of adders are required for implementing these multiples of multiplicands as they cannot be obtained by just shifting and complementing. The same observation is made for modified booth encoding for quaternary logic. The encoding table for the same is shown in Table 6.7 and Table 6.8. The multiples of multiplicand (B) for quaternary logic required are 0B, ±1B, ±2B, ±3B, ±4B, 5B, 6B, 7B, 8B, 9B, 10B, 11B and 12B. Thus for higher

radix ($r > 2$), the multiples of multiplicand required for modified booth recoding are almost same as normal multiplication and require additional stages of adders for generation. Hence, we find that there is no performance improvement through recoding for multi-valued multi digit multiplier.

The partial product generation will consist of the two main blocks which consist of the preparation of partial products using the multi-valued adders and the selection of the correct partial product using the multiplexer. The partial product preparation consists of the multi-valued adders to generate all the multiples (0 to r-1) of the multiplicand. It consists of k -1 stage n-digit half adders, where 2^k= r if r is even and 2^k= r+1 if r is odd. The partial product selection has n r:1 multiplexers to select the partial product of the n digits of the multiplier. Wallace tree made up of carry save adders, which are implemented using (3,2) and (2,2) counters is used for the partial product array reduction. Final addition is performed to produce the final product ($P_{2n-1}....P_0$). Figure 6.4 shows the implementation of a 4-digit quaternary multiplier

A_i	A_{i-1}	A_{i-2}	Y_i	Y_{i-1}
0	0	0	0	0
0	0	1	0	0
0	0	2	0	1
0	1	0	0	1
0	1	1	0	1
0	1	2	0	2
0	2	0	0	2
0	2	1	0	2
0	2	2	1	0
1	0	0	1	0
1	0	1	1	0
1	0	2	1	1
1	1	0	1	1
1	1	1	1	1
1	1	2	1	2
1	2	0	1	2
1	2	1	1	2
1	2	2	2	0
2	0	0	$\bar{1}$	0
2	0	1	$\bar{1}$	0
2	0	2	0	$\bar{2}$
2	1	0	0	$\bar{2}$
2	1	1	0	$\bar{2}$
2	1	2	0	$\bar{1}$
2	2	0	0	$\bar{1}$
2	2	1	0	$\bar{1}$
2	2	2	0	0

Table 6.6. Modifed booth encoding for ternary logic

A_i	A_{i-1}	A_{i-2}	Y_i	Y_{i-1}
0	0	0	0	0
0	0	1	0	0
0	0	2	0	0
0	0	3	0	1
0	1	0	0	1
0	1	1	0	1
0	1	2	0	1
0	1	3	0	2
0	2	0	0	2
0	2	1	0	2
0	2	2	0	2
0	2	3	0	3
0	3	0	0	3
0	3	1	0	3
0	3	2	0	3
0	3	3	1	0
1	0	0	1	0
1	0	1	1	0
1	0	2	1	0
1	0	3	1	1
1	1	0	1	1
1	1	1	1	1
1	1	2	1	1
1	1	3	1	2
1	2	0	1	2
1	2	1	1	2
1	2	2	1	2
1	2	3	1	3
1	3	0	1	3
1	3	1	1	3
1	3	2	1	3
1	3	3	2	0

Table 6.7. Modifed booth encoding for quaternary logic

A_i	A_{i-1}	A_{i-2}	Y_i	Y_{i-1}
2	0	0	2	0
2	0	1	2	0
2	0	2	2	0
2	0	3	2	1
2	1	0	2	1
2	1	1	2	1
2	1	2	2	1
2	1	3	2	2
2	2	0	2	2
2	2	1	2	2
2	2	2	2	2
2	2	3	2	3
2	3	0	2	3
2	3	1	2	3
2	3	2	2	3
2	3	3	3	0
3	0	0	$\bar{1}$	0
3	0	1	$\bar{1}$	0
3	0	2	$\bar{1}$	0
3	0	3	0	$\bar{3}$
3	1	0	0	$\bar{3}$
3	1	1	0	$\bar{3}$
3	1	2	0	$\bar{3}$
3	1	3	0	$\bar{2}$
3	2	0	0	$\bar{2}$
3	2	1	0	$\bar{2}$
3	2	2	0	$\bar{2}$
3	2	3	0	$\bar{1}$
3	3	0	0	$\bar{1}$
3	3	1	0	$\bar{1}$
3	3	2	0	$\bar{1}$
3	3	3	0	0

Table 6.8. Modifed booth encoding for quaternary logic(continued)

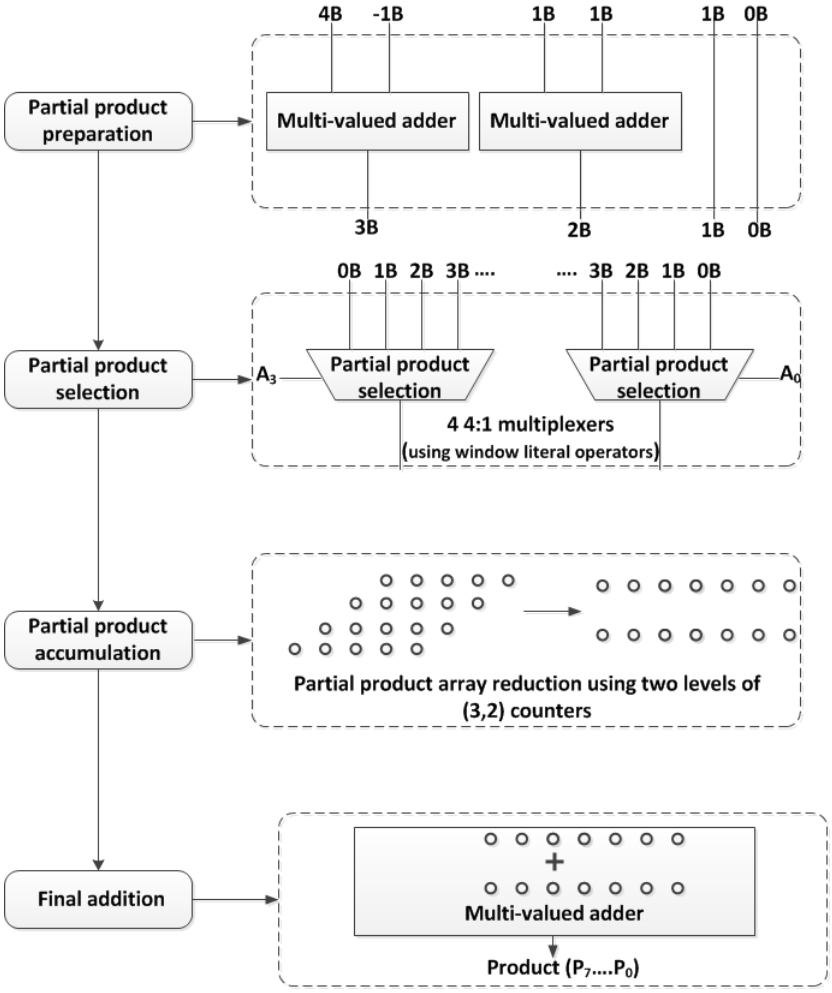

Figure 6.4. 4 digit quaternary multiplier

6.3 Projected comparisons vs.45nm CMOS

To compare binary and multi-level logic , we implement 4, 8, and 16-bit multiplier in binary, quaternary and hexa-decimal SPWF logic . For evaluating the spin wave based multi-valued multiplier implementation, the methodology described for evaluating the multi-valued operators before was used her also. CMOS versions for the multipliers are synthesized with NCSU 45nm technology using design compiler. In SPWF multipliers, multi-valued adders are used for generation of partial products Threshold operator is assumed to use the amplification ME cell. The I/O delays are ignored for both SPWF and CMOS versions. The graphs for the projected comparisons in terms of delay, area and power between CMOS and SPWF implementations are shown in Figure 6.5, Figure 6.6 and Figure 6.7 respectively.

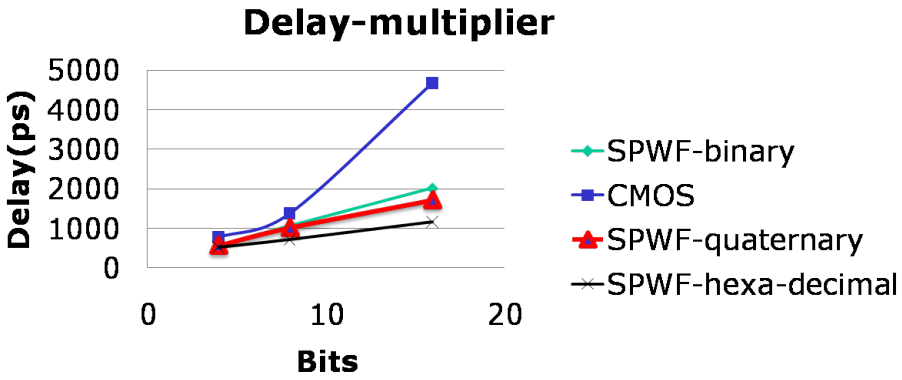

Figure 6.5. Projected comparisons vs. CMOS (45nm) for multipliers-Delay

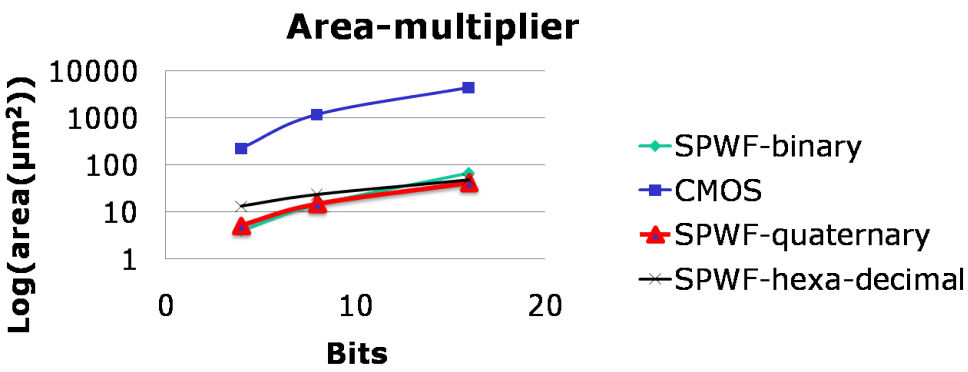

Figure 6.6. Projected comparisons vs. CMOS (45nm) for multipliers-Area

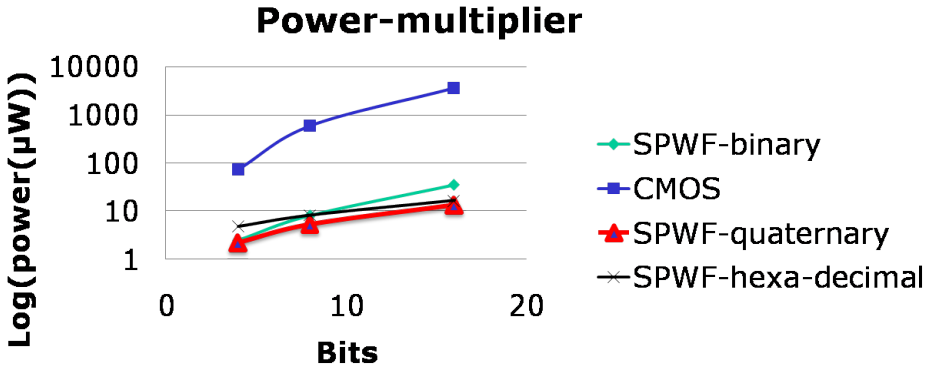

Figure 6.7. Projected comparisons vs. CMOS (45nm) for multipliers-Power

6.4 Chapter summary

We can see that the performance of the multiplier increases with increase in logic level for the SPWF implementation. There is a 4X increase in performance of hexa-decimal SPWF multiplier compared to CMOS for 16 bits. For 4 and 8 bit multiplier, area increases with logic level increment . For 16-bit SPWF multiplier, quaternary has the least area overhead followed by hexa-decimal logic and then binary. We have a 102X improvement for quaternary SPWF multiplier compared to CMOS for 16 bits. For higher order bits (>4), quaternary logic is the most power efficient . There is a 268X improvement for quaternary SPWF multiplier compared to CMOS for 16 bits. Thus, in this chapter we have shown how to implement SPWF based multi-valued multipliers efficiently and also the benefits over CMOS implementation has also been calculated. In the next chapter we discuss about how to implement the I/O logic for converting from binary to r-ary logic using SPWFs.

CHAPTER 7

I/O LOGIC

In this chapter, we look at the implementation of I/O circuits. The I/O logic converts binary logic to r-ary logic and vice-versa. Thus it acts as an interface with the binary electrical domain. This enables a smooth integration with binary CMOS circuits.

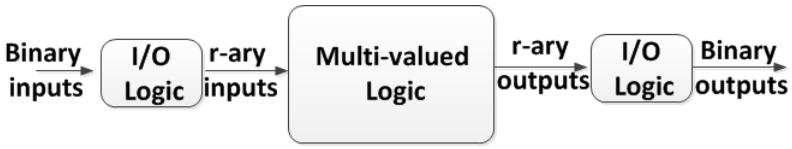

Figure 7.1. Block digaram of multi-valued logic implementation with i/o logic interface

7.1 Binary to r-ary conversion

We start with finding a generic framework for the conversion from binary to r-ary logic. Our approach is to to implement the binary to quaternary and binary to octonary conversion circuits and infer the generic framework from them.

7.1.1 Binary to quaternary conversion

Let us first explore the binary to quaternary conversion. In this conversion, for every two binary inputs (A_1A_0), there is a corresponding quaternary output (Y). This can implemented by a simple weighted majority of the binary inputs. Table 7.1 and Table 7.2 represent the binary and quaternary logic representation in spin waves

respectively. Figure 7.2 shows the SPWF implementation of the binary to quaternary conversion circuit

Logic states	Spin wave representation
0	-A
1	A

Table 7.1. Binary logic

Logic states	Spin wave representation
0	-3A
1	-A
2	A
3	3A

Table 7.2. Quaternary logic

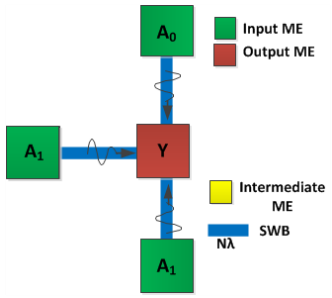

Figure 7.2. Binary- quaternary conversion circuit

7.1.2 Binary to octonary conversion

Similarly, we can implement binary to octonary conversion circuit using weighted majority. Here we convert three binary inputs ($A_2A_1A_0$) to one octonary output (Y) Octonary logic is represented by Table 7.3. Figure 7.3 shows the SPWF implemen-

Logic states	Spin wave representation
0	-7A
1	-5A
2	-3A
3	-A
4	A
5	3A
6	5A
7	7A

Table 7.3. Octonary logic

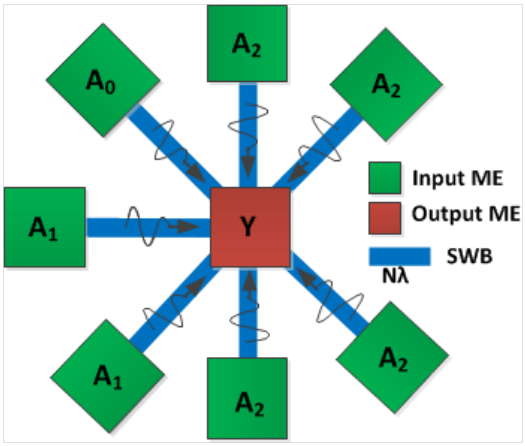

Figure 7.3. Binary- octonary conversion circuit

tation of the binary to octonary conversion circuit. From the above circuits, we can generalise the implementation of binary to r-ary conversion circuit. We need to group d (2^d= r) binary inputs ($A_{d-1}....A_1A_0$) We have one r-ary output (Y) for every group, where Y = Maj($(2^0)A_0, (2^1)A_1$, ...$(2^{d-1})A_{d-1}$). Thus we can convert binary inputs in to multi valued inputs using simple majority logic.

7.2 r-ary to binary conversion

Next we look at how to convert back from r-ary to binary logic. We follow the same approach as for binary to r-ary conversion by implementing the quaternary to binary and octonary to binary conversion circuits and infer the generic framework from them.

7.2.1 Quaternary to binary conversion

We start with converting quaternary to binary logic. We have single quaternary input (A) and two binary outputs (O_1O_0). The binary MSB (O_1) output is 1 only for quaternary input states 2 and 3 . From the encoding for quaternary logic, we see that logic states 2 and 3 have positive phases and 0 and 1 have negative phases. Hence the output of a phase dependent ME cell with input as A would provide the MSB (O_1) . The LSB (O_0) can be generated by subtracting the weighted MSB from the quaternary input (A). Figure 7.4 and Figure 7.5 shows the SPWF implementation of the quaternary to binary conversion circuit

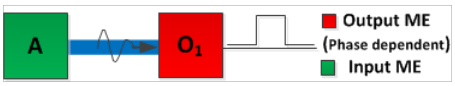

Figure 7.4. Quaternary - Binary conversion circuit (MSB)

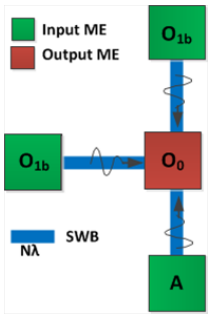

Figure 7.5. Quaternary - Binary conversion circuit (LSB)

7.2.2 Octonary to binary conversion

Using the above approach, we can realize the octonary to binary conversion circuit. Here there is a single Octonary input (A) and three binary outputs ($O_2O_1O_0$)). The MSB (O_2) can be generated from the phase dependent ME cell with A as the input. Output (O_1) is generated from a phase dependent ME cell, whose input is the difference of octonray input and weighted MSB. The LSB output (O_0) can be generated by subtracting the weighted higher output bits from the octonary input ($O_0 = Maj(Maj(A, (4)\bar{O}_2), (2)\bar{O}_1)$. Figure 7.6, Figure 7.7 and Figure 7.8 show the SPWF implementation for the octonary to binary conversion circuit.

Figure 7.6. Octonary - Binary conversion circuit (MSB)

Thus we can generalise the implementation of r-ary to binary conversion circuit. For every single r-ary input (A), there are d(2^d = r) binary outputs ($O_{d-1}....O_1O_0$). The MSB (O_{d-1}) is the output of phase dependent output ME cell with input as A. The remaining output bits (O_{d-i} (where 1<i)) can be generated as the output of phase dependent output ME cell with input as Maj(A, $(2d-1)\bar{O}_{d-1}$, ...$(2d-i+1)\bar{O}_{d-i+1}$). Thus the I/O logic can be implemented with simple weighted majority.

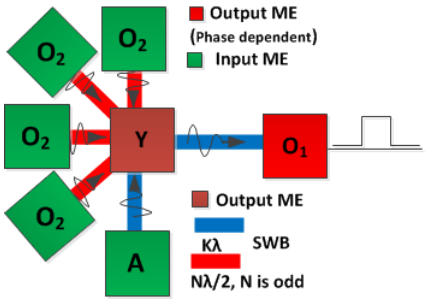

Figure 7.7. Octonary - Binary conversion circuit (O_1)

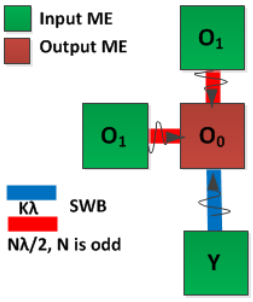

Figure 7.8. Octonary - Binary conversion circuit (LSB)

CHAPTER 8

CONCLUSIONS

We have introduced a truly multi-state device using spin waves. The implementation of basic operators of multi-valued logic using spin wave functions has been demonstrated. The benefits of the SPWF implementation of these operators over implementation in CMOS have also been presented. For efficient implementation of arithmetic circuits like adders and multipliers new operators have been proposed and implemented. Benchmarking with CMOS for these arithmetic circuits was also done and our initial evaluation for 32-bits, show a 5X increase in performance, 125X area improvement and 1717X power reduction for hexa-decimal spin wave based adders compared to binary CMOS. Similarly, there is a 4X increase in performance of hexa-decimal SPWF multiplier compared to CMOS for 16 bits. From our implementation of multi-valued multipliers, we infer that increase in logic level after a certain limit (quaternary) does not provide area and power benefit. To ensure easier integration with CMOS, I/O circuits for smooth interface between binary CMOS and multi-valued SPWF logic have also been developed. Thus, we have demonstrated that by using a truly multi-valued device, we can build high speed arithmetic circuits of future for processing the huge data sets.

BIBLIOGRAPHY

[1] Abdolahzadegan, Sh, Keshavarzian, Peiman, and Navi, Keivan. Mvl current mode circuit design through carbon nanotube technology. *European Journal of Scientific Research 42*, 1 (2010), 152–163.

[2] Allen, C.M., and Givone, D.D. A minimization technique for multiple-valued logic systems. *Computers, IEEE Transactions on C-17*, 2 (feb. 1968), 182 – 184.

[3] Brayton, R.K., and Khatri, S.P. Multi-valued logic synthesis. In *VLSI Design, 1999. Proceedings. Twelfth International Conference On* (jan 1999), pp. 196 – 205.

[4] Hassoun, S., and Sasao, T. *Logic Synthesis and Verification*. The Springer International Series in Engineering and Computer Science Series. Kluwer Academic Publ., 2002.

[5] Hemati, S. High Performance MOS Current Mode Logic Circuits. Ph.D. candidate presentation, Carleton university.

[6] Hurst, S.L. Multiple-valued logic its status and its future. *IEEE Transactions on Computers 33*, 12 (1984), 1160–1179.

[7] Jain, A.K., Bolton, R.J., and Abd-El-Barr, M.H. Cmos multiple-valued logic design. ii. function realization. *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on 40*, 8 (aug 1993), 515 –522.

[8] Khitun, A., Bao, Mingqiang, and Wang, K.L. Spin wave magnetic nanofabric: A new approach to spin-based logic circuitry. *Magnetics, IEEE Transactions on 44*, 9 (sept. 2008), 2141 –2152.

[9] Miller, D.M., and Thornton, M.A. *Multiple Valued Logic: Concepts and Representations*. Synthesis Lectures on Digital Circuits and Systems Series. Morgan & Claypool, 2008.

[10] Navi, K., Kazeminejad, A., and Etiemble, D. Performance of cmos current mode full adders. In *Multiple-Valued Logic, 1994. Proceedings., Twenty-Fourth International Symposium on* (may 1994), pp. 27 –34.

[11] Onneweer, Siep P, and Kerkhoff, Hans G. High-radix current-mode cmos circuits based on the truncated-difference operator. In *Proc. 17th ISMVL* (1987), pp. 188–195.

[12] Post, Emil L. Introduction to a general theory of elementary propositions. *American Journal of Mathematics 43*, 3 (1921), pp. 163–185.

[13] ReC-SDSW, (Research Center for Spin Dynamics and Spin-Wave Devices). Collective motion of coupled spins (spin waves). `http://sdsw.snu.ac.kr`, 2012.

[14] Sarica, F. *Current Mode CMOS sequential multiple-valued logic circuits*. PhD thesis, Bogazici University, 2012.

[15] Shabadi, Prasad, Rajapandian, Sankara Narayanan, Khasanvis, Santosh, and Moritz, Csaba Andras. Design of spin wave functions-based logic circuits. *SPIN 02*, 03 (2012), 1240006.

[16] Temel, T., and Morgul, A. Implementation of multi-valued logic, simultaneous literal operations with full cmos current-mode threshold circuits. *Electronics Letters 38*, 4 (feb 2002), 160 –161.

[17] Temel, Turgay, and Morgul, Avni. Implementation of multi-valued logic gates using full current-mode cmos circuits. *Analog Integrated Circuits and Signal Processing 39* (2004), 191–204.