Energy-efficient DSP System Design

based on the Redundant Binary Number System

by

Rupa Mahadevan

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved July 2011 by the
Graduate Supervisory Committee:

Chaitali Chakrabarti, Chair
Sayfe Kiaei
Yu Cao

ARIZONA STATE UNIVERSITY

August 2011

ABSTRACT

Redundant Binary (RBR) number representations have been extensively used in the past for high-throughput Digital Signal Processing (DSP) systems. Data-path components based on this number system have smaller critical path delay but larger area compared to conventional two's complement systems. This work explores the use of RBR number representation for implementing high-throughput DSP systems that are also energy-efficient.

Data-path components such as adders and multipliers are evaluated with respect to critical path delay, energy and Energy-Delay Product (EDP). A new design for a RBR adder with very good EDP performance has been proposed. The corresponding RBR parallel adder has a much lower critical path delay and EDP compared to two's complement carry select and carry look-ahead adder implementations. Next, several RBR multiplier architectures are investigated and their performance compared to two's complement systems. These include two new multiplier architectures: a purely RBR multiplier where both the operands are in RBR form, and a hybrid multiplier where the multiplicand is in RBR form and the other operand is represented in conventional two's complement form. Both the RBR and hybrid designs are demonstrated to have better EDP performance compared to conventional two's complement multipliers. The hybrid multiplier is also shown to have a superior EDP performance compared to the RBR multiplier, with much lower implementation area. Analysis on the effect of bit-precision is also performed, and it is shown that the performance gain of RBR systems improves for higher bit precision.

Next, in order to demonstrate the efficacy of the RBR representation at the system-level, the performance of RBR and hybrid implementations of some common DSP kernels such as Discrete Cosine Transform, edge detection using Sobel operator, complex multiplication, Lifting-based Discrete Wavelet Transform (9, 7) filter, and FIR filter, is compared with two's complement systems. It is shown that for relatively large computation modules, the RBR to two's complement conversion overhead gets amortized. In case of systems with high complexity, for iso-throughput, both the hybrid and RBR implementations are demonstrated to be superior with lower average energy consumption. For low complexity systems,

the conversion overhead is significant, and overpowers the EDP performance gain obtained from the RBR computation operation.

To my parents - for they are the pillars that hold me up...

... I am what I am because they are my eternal inspiration.

To my guardian angels - my sister and brother-in-law...

... whose love and enduring protection I wholeheartedly cherish.

ACKNOWLEDGEMENTS

I owe my sincerest gratitude to my advisor, Dr. Chaitali Chakrabarti, for her constant direction and thoughtful encouragement, right since the initial days of my research, which has only grown from then to now. There are some who teach, and others who impart. I honestly believe that her tutoring has made me a better academic professional. I more than appreciate our extended discussions and interactions, and her patience and tolerance towards my imperfections. Her ardent interest and guidance have been indispensable in the successful completion of my research.

My sincere thanks are also due to my committee members Dr. Sayfe Kiaei and Dr. Yu Cao for their kind encouragement.

I am deeply grateful to my mentor, Yunus Emre, for the copious discussions and brainstorming sessions we have had throughout the course of my research. I owe the clarity in my signal processing concepts to Yunus. I am equally grateful to my colleagues Chi-Li Yu, Chengen Yang, and Ming Yang, for their constant motivation and patience towards my questions.

I am profoundly indebted to my friend, Nikhil Ghadge, for teaching me the importance and power of automation. Without his help and motivation, the timely completion of this thesis would have been impossible. I also thank my friend, Hardik Mehta, for not only helping me with the documentation process, but also for his unconditional support.

I am also extremely thankful to my room-mates, Vaidehi and Shreya, for putting up with me these past two years, and all my friends, whose encouragement I greatly appreciate.

I would be horribly remiss if I do not mention the faculty members and the administrative staff, here at Arizona State University, for making this journey of my master's degree enormously smooth-sailing.

TABLE OF CONTENTS

Chapter Page

Chapter Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Chapter 1

INTRODUCTION

Contemporary Digital Signal Processing (DSP) systems are characterized by high-throughput, energy-intensive computations. Examples of such systems include image and video codecs. Many of these computation-intensive systems are on portable platforms, where energy constraints pose a rigid limitation to the number of computations that can be performed given the battery restrictions. Thus, for high-throughput, energy-efficient DSP systems, it is desirable to have low delay, energy and the energy-delay product.

The performance of a DSP system is dictated by the performance of datapath components that are used to implement the constituent kernels. This is because most DSP kernels can be reduced to a mix of add/shift/multiply operations. This work looks at optimizing the speed and energy performance of data-path components in a for DSP system.

1.1 Redundant Number Systems

Conventional DSP data-path components are based on two's complement binary arithmetic. They are popular because of legacy architectures and ease of implementation. However, two's complement adders and multipliers encompass carry propagation chains, which limit the speed performance, especially for large operand bit-widths. Even the best two's complement adders and multiplier designs have a critical path delay that is proportional to $O(logn)$, where n is the bit-width. For high-throughput DSP systems, these two's complement adders and multipliers are heavily pipelined so that high frequency of operation may be achieved. This, in turn, increases the system latency and area.

In this thesis, we focus on design of data-path components operating in the redundant number system. These systems were first conceived in order to reduce or eliminate carry propagation chains in arithmetic circuits. They are characterized by carry-propagation-free addition; addition can be performed in constant time, independent of the operand bit-width. This feature of fast, parallel addition was also utilized in multipliers,

to yield fast multiplication times. While two's complement systems are characterized by unique number representations, redundant number systems have multiple, redundant, representations for the same number. This allows the addition rules to be manipulated, so that fast, parallel, carry-propagation-free addition can be accomplished. However, since multiple representations are possible, more number of bits are required to represent a single two's complement bit (called a digit in the redundant system). This translates to processing more number of bits at a single digit position, and so, the gate complexity for single-digit redundant system adders is higher than two's complement single-bit adders. Also, since majority of the processing environment is in two's complement, any computation block that uses redundant representations is required to convert to and from two's complement representation, leading to additional circuit overhead.

Redundant number systems were very popular, a couple of decades ago, for high-throughput applications. This is because the carry-propagation-free addition allowed Most-Significant-Digit (MSD) first computation [7, 9, 15, 22, 21, 43, 49].Conversion from and to two's complement representation were performed only when the operands are read from or written to memory. Even Intel introduced a processor architecture that used redundant intermediate forms to improve the instruction pipelines [11]. However, due to the circuit, and conversion overhead involved, these number systems are not very popular today.

1.2 Contributions

In this work, redundant number system arithmetic is revisited, and its applicability to the design of energy-efficient systems is investigated. The idea is that since computation units that use redundant number representation are high speed, in order to obtain the same speed of operation as two's complement systems, the supply voltage of the redundant number representation units can be scaled. Since energy is proportional to the square of the supply voltage, this reduces the average energy consumption while maintaining the throughput.

Three performance metrics are considered: delay, energy and energy-delay product (EDP). We include energy-delay product since the deisgn objective is to have the best

speed perfromance, with the least possible energy consumption. First, a new radix-2 Redundant Binary (RBR) number representation based adder is proposed and its performance is compared with an existing RBR adder design as well as two's complement Carry -Select Adder (CSA) and carry look-ahead Adder (CLA). The proposed design is shown to have reduced critical path delay, and comparable EDP performance with respect to the reference RBR adder design. The proposed RBR parallel adder has a superior EDP performance with respect to the two's complement CSA and CLA adders. Since the RBR adder delay is independent of bit-width, the EDP performance gain of RBR adders increases with increasing bit-width.

Second, a new hybrid digit-parallel multiplier design that accepts the multiplicand operand in RBR form and the second operand in two's complement form is proposed and its performance evaluated. A novel architecture for a RBR digit-parallel multiplier that accepts both operands in RBR form that is based on the new hybrid multiplication architecture is also proposed. The performance of this new RBR multiplier is compared with existing RBR tree multiplier architectures. This new design is both energy and area-efficient compared to the RBR tree multiplier designs, with and a 22% relative EDP performance gain, as well as 40% relative area efficiency.

Extensive performance evaluations of the new multiplier architectures are performed in comparison to two's complement multipliers. The EDP performance of both the proposed multiplication architectures is evaluated with respect to that of existing two's complement Wallace tree multiplier designs. Both new multiplication architectures have superior EDP performance over the two's complement multiplier. For iso-throughput comparison, with voltage scaling, the new architectures are found to have lower average energy consumption. The RBR multiplier exhibits 14% EDP performance gain, while the hybrid multiplier shows a substantial 42% EDP performance gain over conventional two's complement multiplier, for iso-throughput. Analysis on the implementation area comparisons for all multiplier architectures is also performed, and it is shown that while the RBR implementation has a 25% area overhead, the hybrid multiplier exhibits comparable area with

respect to the two's complement system.

Next, popular DSP kernels such as edge detection using Sobel operator, Discrete Cosine Transform, complex multiplication, lifting-based Discrete Wavelet Transform (9, 7) filter, and FIR filter design are built using the proposed RBR adder, and the proposed RBR and hybrid multipliers. Performance evaluations are performed in comparison to two's complement modules to demonstrate the efficacy of RBR representation for implementing DSP systems. Conversion overhead, which is inevitable for RBR systems, is analyzed for all the RBR implementations of the DSP kernels. For low complexity systems, the RBR conversion presents a significant overhead and the EDP performance of two's complement systems is found to be better. As the complexity of the computation system increases, the conversion overhead is shown to get amortized over the computation operation, and the RBR systems are found to have superior EDP performance over two's complement systems. In fact, for high complexity DSP kernels, substantial relative EDP performance gain is observed. For iso-throughput, the hybrid implementations achieve an impressive 37% average EDP performance gain over two's complement systems.

1.3 Thesis Organization

The layout of this work is as follows. Chapter 2 presents an in-depth discussion on redundant number representations. The simulation environment and the Verilog-based model used for performance evaluations in this work is also introduced here. In Chapter 3, the addition algorithm and design of the proposed new RBR adder design is described in detail with all supporting simulation results. Chapter 4 presents EDP performance evaluations of RBR and two's complement multipliers. The architectures of the proposed new, digit-parallel, purely RBR and hybrid multiplier is also introduced here. Chapter 5 addresses the issue of overhead in RBR systems and considers five popular DSP applications to compare the performance of the two's complement and RBR implementations. Concluding comments, along with pointers to further research in this area are provided in Chapter 6.

Chapter 2

BACKGROUND

This chapter elaborates on some of the representations that have been researched in the past for high-performance computing. The focus here is on the redundant number representation, which forms the basis of this work.

2.1 Two's Complement Representation

In a conventional number system with radix-r, the digits can assume exactly r values, from 0 to $r-1$. For instance, the binary (radix-2) system has the digit-set $\{0,1\}$, while the radix-4 system has the digit-set $\{0,1,2,3\}$. In the weighted positional system in radix r, a number X with n digits, written as $x_{n-1}x_{n-2}x_1x_0$, has the value,

$$X = \sum_{i=0}^{n-1} x_i r^i$$

Each digit position i has an assigned weight r^i, and the digits are multiplied by this weight to calculate the numeric value of the number. Thus, each number has a unique digit representation in this system.

In the two's complement binary representation, the range of numbers is from $-2^{(n-1)}$ to $2^{(n-1)}-1$ [35]. Although two's complement systems usually have relatively low complexity, in the worst-case, there is a carry propagation from the least significant to the most significant position. Even for the most efficient two's complement adder designs, the delay of an n-bit adder is proportional to $O(logn)$. Thus, these systems tend to have large latency, and additional techniques have to be incorporated before they can be used to build high throughput systems.

2.2 Redundant Number Representation

In order to speed-up computation and improve the performance of the arithmetic units, redundant number systems were developed. Arithmetic circuits built in this number system had reduced carry propagation systems. In contrast to the conventional number system

that consists of only unique number representations, the redundant number system permits multiple (redundant) representations of the same number.These multiple representations allow the addition rules to be manipulated, resulting in fast, parallel addition. For instance, in the addition intermediate representation of the incoming operand sum $(x_i + y_i)$ is chosen, so that the final-stage addition will not present a carry to the higher-level digit position.

Implementation of redundant arithmetic algorithms can be broadly classified into the following categories, [12]: (1) conventional binary logic encoding, wherein the multi-valued redundant digits are encoded using two or more binary bits; (2) current-mode circuits, which use non-binary digital current signals to represent multi-valued redundant digits; (3) heterostructure and quantum electronic circuits, which rely on device-level innovations to incorporate redundancy. The focus of this work is entirely on implementation of redundant arithmetic algorithms using conventional binary logic circuits, and addressing its design challenges. The following sections elucidate the fundamental concepts of the redundant number representation.

2.2.1 Signed Digit Representation

The signed-digit number system [3] was originally proposed by Avizienis, with the purpose of implementing parallel addition where carry propagation is eliminated altogether. The algebraic value of a signed-digit number is given by

$$P = \sum_{i=-n}^{m} p_i r^{-i}$$

where r is the radix, $r > 0$ and p_i are the digits. In a redundant number representation, the digit-set is comprised of more than r values, unlike the conventional number representation, where it is restricted to exactly r values. The values of the radix and the digits, p_i, are chosen to satisfy the condition of unique representation for the value $P = 0$. The sign of a redundant number is the sign of the most-significant non-zero digit. Also, the signed-digit representation of P is simply obtained by changing the sign of every non-zero digit in the representation of P.

The parallel addition algorithm of the signed-digit representation is performed in two steps, and is shown pictorially in Figure 2.1.

$$x_i + y_i = rc_i + w_i$$

$$s_i = w_i + c_{i-1}$$

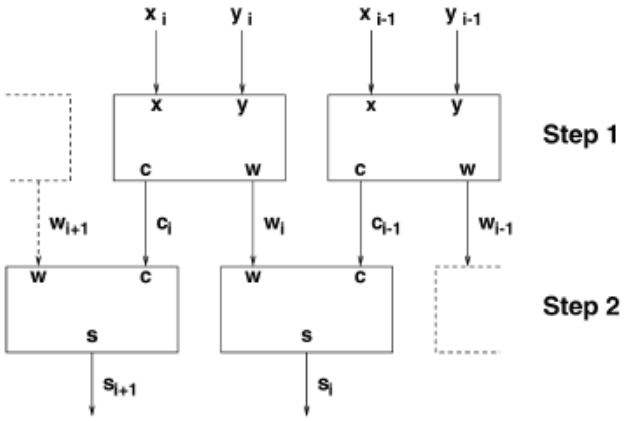


Figure 2.1: Signed Digit Representation. Totally parallel addition

The addition of two digits x_i and y_i is totally parallel only if: (1) the final sum digit is a function of only the operand digits x_i and y_i, and the interim carry digit, $c_{(i-1)}$ from the adjacent digit position; (2) the interim carry digit to the next position, c_i, is a function of only the operand digits, x_i and y_i. From the above definition of the two-step totally parallel addition algorithm, the required and allowed digit values for each of the variables can be derived. The upper bound for the magnitude of the interim sum is $|w_i| \leq r-2$ and the smallest sufficient set of values for the carry digit is $c_i = \{-1, 0, 1\}$.

Importantly, from the derivation, it is apparent that Avizienis signed-digit representation is valid only for radix values $r > 2$. This is because, for radix $r = 2$, the allowed digit-set for the interim sum is_i is only $\{0\}$! Although Avizienis signed-digit representation excludes radix-2 representation, one possible way to make it work for radix-2 is by allowing the carries to ripple two positions, making it a three-step addition process [19].

2.2.2 *Generalized Signed Digit Representation*

Parhami investigated digit-level redundant representations, and introduced a framework for generalized signed digit (GSD) number systems [33, 34, 35]. Avizienis signed-digit representation is included in this framework, and is called the Ordinary Signed-Digit number system (OSD). The GSD number system unified the OSD and the Binary Signed-Digit (BSD) number systems, and included other practical redundant number representations such as stored-carry and stored-borrow as special cases.

Parhamis GSD number system is a weighted positional system with digit-set $\{-\alpha, -\alpha+1, ..., \beta-1, \beta\}$, where $\alpha \geq 0$, and $\beta \geq 0$, $\alpha+\beta+1 > r$, where r is the radix of the number representation. The excluded case $\alpha+\beta+1 = r$ results in a non-redundant number systems, which covers the conventional radix-r system with $\alpha = 0$, $\beta = r-1$ as a special case. GSD number systems cover the following number systems as special cases.

(1) Binary Stored-Carry (BSC): $r = 2$, $\alpha = 0$, $\beta = 2$.

(2) radix-r Stored-Carry (SC): $\alpha = 0$, $\beta = r$.

(3) Binary Stored-Borrow (BSB or BSD): $r = 2$, $\alpha = \beta = 1$.

(4) radix-r Stored-Borrow (SB): $\alpha = 1$, $\beta = r-1$.

(5) Binary Stored-Carry-or-Borrow (BSCB): $r = 2$, $\alpha = 1$, $\beta = 2$.

(6) radix-r Stored-Carry-or-Borrow (SCB): $\alpha = 1$, $\beta = r$.

(7) Minimally redundant symmetric signed-digit: $2\alpha = 2\beta = r \geq 4$.

(8) Ordinary Signed-Digit (OSD): $r \geq 3$, $1/2r \leq \alpha = \beta \leq r$.
Minimally redundant: $\alpha = \beta = \lfloor 1/2r \rfloor + 1$.
Maximally redundant: $\alpha = \beta = r-1$.

For a symmetric digit set, α *equals* β. A maximally redundant radix-4 OSD number representation has the digit-set $\{-3,-2,-1,0,1,2,3\}$, while the minimally redundant radix-4 OSD representation has the digit-set $\{-2,-1,0,1,2\}$.

The BSC number representation, which is a special case of radix-r stored-carry representation, has a digit-set $\{0,1,2\}$. A BSC number can be added to a conventional binary number to give a BSC result, using a set of full adders without carry propagation, and has been used before in a few implementations [50].

SB number representation uses the digit-set $\{-1,0,1,,r-1\}$. For radix r=2, this system is called the Binary Stored-Borrow (BSB) or the Binary Signed-Digit (BSD) number system, and has the digit-set $\{-1,0,1\}$. BSD numbers have been used for representing intermediate temporary values in high-speed multiplication and division algorithms such as Booths recoding algorithm for multiplication [12]. Two BSD numbers can be added by limited carry propogation. Examples of BSD implementations are given in [16, 29, 47].

2.2.3 Choice of Radix in the Redundant Representation

As the radix r increases (and hence the number of bits required to represent a number in conventional number system increases), the number of extra bits required for the redundant representation relative to the number of bits for the conventional representation decreases. Thus, as r increases, the representation overhead for redundant systems decreases as shown in Table 2.1. The same cannot be said, however, for the implementation complexity.

Radix-2 and radix-4 are the most commonly chosen representations for redundant representations. For instance the Paste-up system, introduced by Irwin and Owens in [22] uses radix-4, which they claim has fewer interconnects and simpler logic. Their rationale for choosing radix-4 is that relatively low latency radix-2 systems are difficult to design [19, 22]] and the gate complexity and slow processing speed of radix-8 precludes its usage, although it can process more data per cycle. In [19], Irwin illustrates that radix-2 redundant binary addition can be performed by limiting the carry chain ripple to two digit positions by a three-step addition process.

Table 2.1: Representation Overhead in Redundant Systems

Base	2	3	4	8	10	16	32
No. of bits in redundent representation	2	3	3	4	5	5	6
No. of bits in conventional representation	1	2	2	3	4	4	5
Overhead	100%	50%	50%	33%	25%	25%	20%

In iterative or serial implementations of arithmetic operations, higher-order radices can be employed to reduce the number of iteration cycles [36]. Radix-4 representation is advantageous for such systems, since it halves the number of iteration cycles in comparison to radix-2. However, the basic components in a radix-4 system are far more complex. In this work, we used the radix-2 representation for the adder and multiplier implementations.

2.2.4 Redundant Binary Representation

In redundant binary logic, each redundant digit is encoded by two or more bits. For example, a radix-2 redundant digit can be encoded using two bits [12]. Carry-save is one such type of redundant representation, having the digit-set $\{0,1,2\}$. There are two well-known encodings [28] for $\{-1,0,1\}$, which are *sign-mag*, where the two-bits represent a magnitude and a sign, respectively, and *borrow-save*, wherein one bit is positive and the other negative [12]. We choose the borrow-save representation for representing the radix-2 digits in our proposed designs.

2.2.5 Conversion Issues

Since legacy architectures use two's complement arithmetic, any system using redundant number representation kernels must address the issue of converting to/from the redundant digit system. Redundant systems have an overhead, not only in terms of the additional number of bits used for the data representation, but also in terms of the input and output conversion as shown in Figure 2.2. Conversion to the redundant-binary representation is trivial, and is comprised of only expanding the operand bit-width. Only the MSB-bit re-

quires little manipulation to take care of negative numbers.

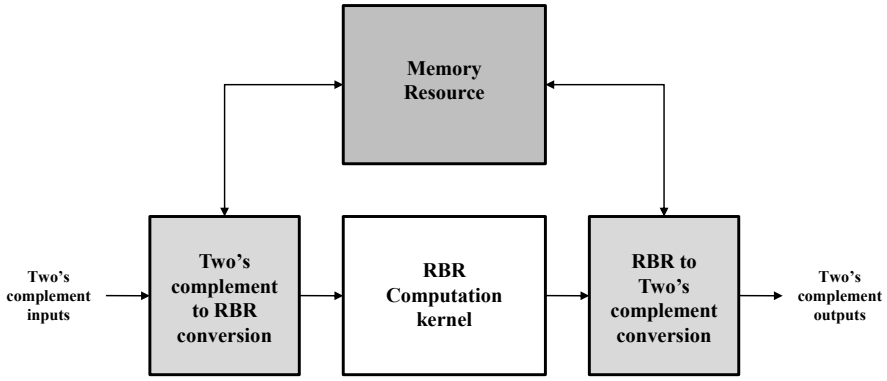


Figure 2.2: Overhead in a typical RBR System

The real overhead for RBR systems lies in the RBR-to-two's complement conversion, which is typically implemented by converting the RBR result to two unsigned two's complement numbers R_+ and R_-, obtained by mapping only the '1' digit and '-1' digit positions respectively, and then finding their difference, $R_+ - R_-$. This RBR-to-two's complement conversion unit is typically implemented by a fast two's complement parallel adder, the delay and power consumption of which increases with increasing bit precision.

In addition to communication with other computation units, any data-path kernel also communicates with the system memory. If a RBR kernel is used to fetch and store RBR operands from and to memory, then the memory requirement of the system automatically doubles because of the representation overhead. Doubling the system memory leads to almost double the average energy consumption. Hence, to avoid increasing the memory capacity and subsequent energy consumption, converting the RBR numbers into two's complement and storing them in two's complement form is advisable.

2.3 Performance Metrics: Delay, Average Energy Consumption, and Energy-Delay Product (EDP)

We compare the competing designs with respect to critical path delay, average energy consumption, EDP and implementation area. The power consumption of a design governs the

amount of energy consumed per operation, and how much heat the circuit dissipates. It is an important design metric, because, it directly affects the system requirements such as power supply ratings, battery specifications, packaging and cooling requirements. In addition, increased on-chip power density is extremely detrimental to reliability, as it makes the chip more easily prone to 'thermal runaway'.

The power consumption of a system is given by $P = V_{DD}.I_{total}$, where I_{total} is comprised of I_{static} and $I_{dynamic}$.I_{static} is the total leakage current across the integrated circuit, while $I_{dynamic}$ is the total switching power, due to the charging and discharging of the circuit capacitances, that is, $I_{dynamic} = \alpha C_L V_{DD}^2 f$, where α is the activity factor, which indicates how often the circuit switches, C_L is the effective load capacitance that is switched, V_{DD} is the supply voltage, and f is the frequency of operation. The dynamic power has a quadratic dependence on the supply voltage V_{DD}. Thus, reducing the supply voltage, also known as voltage scaling, will automatically lead to quadratic reduction in power. While this is true, it does come with a penalty in terms of speed performance. This is because, with reduction in supply voltage, the current drive reduces, and hence, the signal slopes suffer, thus affecting transition and propagation delay times, and ultimately the frequency of operation. Thus, propagation delay and power consumption of a gate are interrelated.

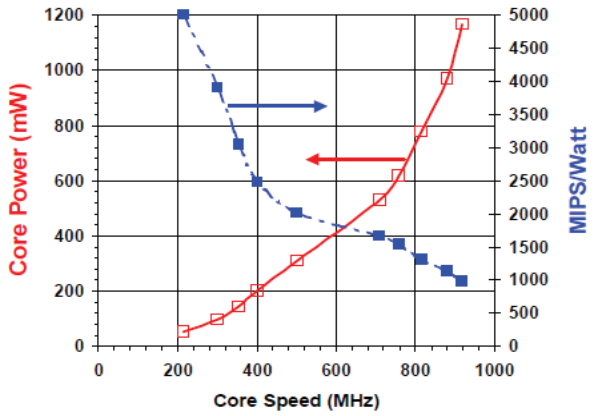


Figure 2.3: Trade-off between supply voltage and frequency of operation

The Power-Delay Product (PDP), is a quality measure that determines the energy

consumption of a gate per switching event, that is, charging or discharging of the load capacitance. Faster the energy transfer, the faster the gate, however, higher the power consumption. The power-delay product, or the average energy consumption, however, does not give any information regarding the speed of operation. In contrast, the EDP unifies the two quality metrics, propagation delay and energy consumption, and allows designers to trade-off one with the other. Figure 2.3 depicts the trade-off between supply voltage and frequency of operation of a processor. In this work, we use delay,energy, EDP as performance metrics while evaluating data-path components based on two's complement and RBR representation systems.

2.4 Simulation Environment

Figure 2.4 is a pictorial representation of the simulation environment. In order to model the delay, energy consumption and EDP characteristics of the various arithmetic units, a gate library consisting of the commonly used logic gates, including multiplexers, and registers, is characterized using HSPICE. The circuit-level implementation of these gates and their HSPICE characterization was performed using the PTM 45 nm process technology models [18] (fanout-of-4 (FO4) inverter delay is 40 ps for this technology). The propagation delay and average switching energy consumption for both rising and falling transitions is estimated for each gate for different load conditions such as FO1, FO2, and FO4. The gate libraries were also characterized for supply voltages ranging from 0.6 V to 1 V (nominal) in steps of 0.1 V. The average energy consumption in each case is determined by integrating the total switching power consumption of the gate over the transition time.

Verilog HDL structural constructs were used to simulate all the circuits with the delay and energy parameters referenced from the characterized gate library. To elaborate, a gate-level model of the arithmetic unit was developed in Verilog, such that every time the input to any gate switches, its output takes its logical value after a circuit propagation delay as determined by the HSPICE parameter library. The average energy consumption of each system is approximated by calculating the number of rising and falling transitions of each gate, and replicating the average switching energy consumption of a single transition

as obtained from the HSPICE parameter library.

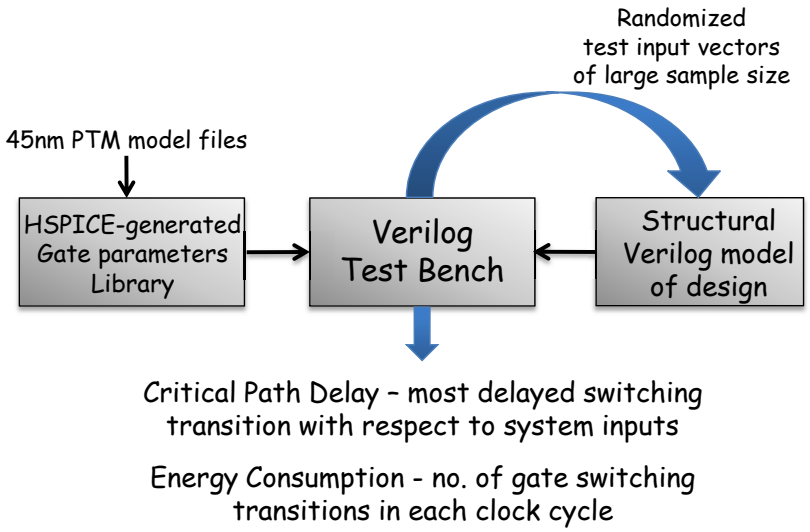


Figure 2.4: Pictorial representation of simulation environment

The simulation setup involved applying large number of randomized input vectors (both operands) to the circuits, for instance 32,767 in case of the 16-bit adders and 16-bit multipliers. The critical path delay was obtained across the entire range of input vectors. The signal path delay calculations were performed by detecting signal transitions at each output and by recognizing the most delayed output signal transition with respect to the system inputs, as the critical path. The energy consumption was estimated by averaging the energy values for all input combinations. The system energy values for each input combination are calculated by counting the number of rising and falling signal transitions. Each rising or falling transition of any signal in the system is linked to the HSPICE-generated gate library to obtain the average rising/falling transition energies for each gate. The cumulative transition energies of all the switching gates in the system for each input operand combination is the system average energy consumption for that input combination.

Chapter 3

REDUNDANT BINARY ADDER ARCHITECTURES

Adders based on RBR representations have a fixed delay, independent of the bit-width of the operands, which makes them highly efficient for large operand-width computations. Such adders have been extensively researched in the past. Apart from speeding up addition, they also enable fast multiplication.

Since the basic redundant binary adder cell processes more bits than a normal binary full adder cell, these units are quite gate-intensive. To the best of our knowledge, these structures were seldom researched for low energy designs. In this work, we revisit RBR adder designs and evaluate their effectiveness in terms of delay, energy and EDP. A new radix-2 RBR adder with novel addition rules is proposed, and its performance compared with a previously reported RBR adder [29]. The RBR adders are also compared with the two's complement adders in order to demonstrate their superiority in terms of EDP.

3.1 Existing Radix-2 RBR Adder Designs

A lot of the prior research on RBR systems focused on finding the best representation that resulted in a compact adder cell. Many algorithms and circuits have been reported for the redundant binary adder [5, 16, 29, 39, 8, 25, 42, 48, 46]. Most of the early radix-2 RBR adders, based on the digit-set $\{-1,0,1\}$, used the three-level scheme, where the sum digit is a function of the digits in three adjacent digital positions. Figure 3.1 shows the high-level representation of such a 1-digit RBR adder [19]. Stage I produces a transfer carry tc_j; Stage II generates the interim carry $ic_{(j+1)}$ and the interim sum is_j. The interim carry and interim sum digit sets are chosen such that they can never be simultaneously 1 and -1, thus eliminating a possible carry condition. Stage III performs addition of the interim sum and carry to generate the final RBR sum digit s_j. Variations of this scheme were also researched [5]. However, as seen, the sum digit depends on the operands from three adjacent positions, leading to relatively slow parallel addition.

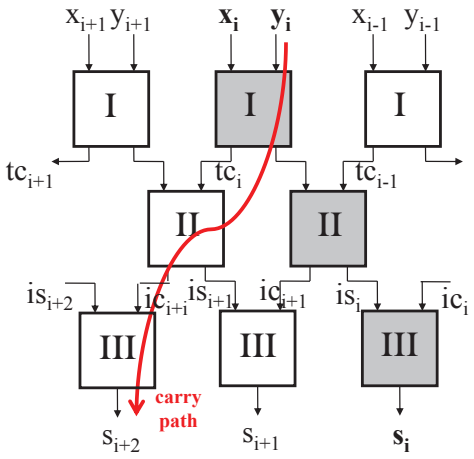


Figure 3.1: Three level redundant binary addition scheme

New addition schemes were later devised to shorter addition time by making the sum digit dependent on only two adjacent operand positions [29, 8, 25, 42, 48]. In [25], Kuninobu et al. presented a new logic circuit for the RBR adder, which is smaller and faster. In [42], a transmission gate based RBR adder was designed, which resulted in a compact and efficient basic building block for the technology node considered, which was $0.8\mu m$. However, these transmission gate RBA blocks may not be viable for contemporary deep sub-micron processes.

In [29], a fast redundant-binary multiplication scheme based on a high speed RBR adder design was proposed. The adder was not only fast but had a more compact circuit realization compared to normal binary 4:2 compressors, as well as other RBR adder designs such as those in [25] and [48]. This is shown in Figure 3.2. Because of the relatively superior RBR adder design in [29], this scheme is chosen as a benchmark for comparing with the proposed adder design.

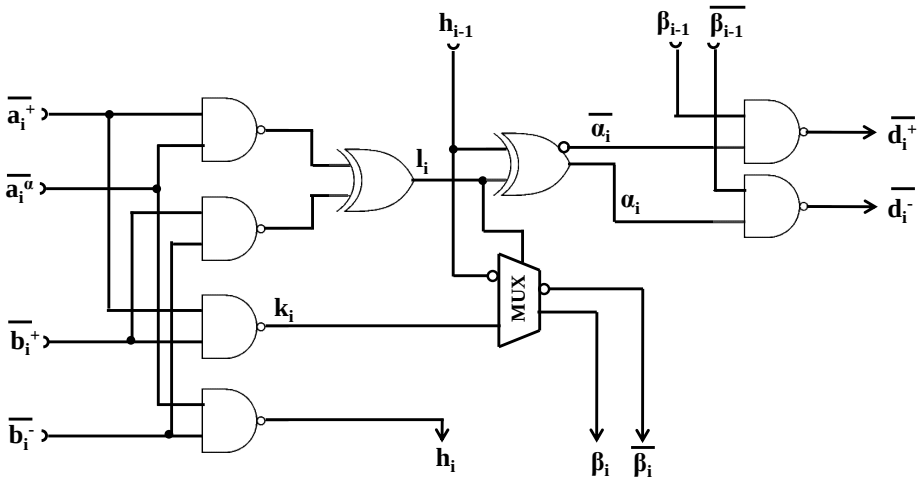


Figure 3.2: Redundant binary adder from [29]

Table 3.1: Bit-level digit representation of the borrow-save encoding scheme

x_i	x_i^-	x_i^+
0	0	0
1	0	1
-1	1	0

3.2 Proposed Radix-2 Redundant Binary Adder Design

3.2.1 Encoding Scheme

The proposed addition algorithm is based on a radix-2 representation, with the digit-set $\{-1,0,1\}$, encoded using two bits. In this representation, if x_j is a redundant binary digit denoted using two bits (x_j^-, x_j^+), then the algebraic value of the digit x_j can be calculated as $x_j = x_j^+ - x_j^-$, where $x_j \in [-1,0,1]$, and $\{x_j^+, x_j^-\} \in [0,1]$. The assumed representation leads to a bit-level encoding as shown in Table 3.1. Note that the (1,1) combination is unused, since the encoding shown in Table 3.1 was found to give lower implementation overhead.

3.2.2 Addition Algorithm

Most redundant binary radix-2 adder designs express the addition of two redundant binary operands x_j and y_j as a combination of an intermediate sum, is_j, and an intermediate carry, ic_{j+1}. Thus the sum of x_j and y_j is represented by $x_j + y_j = 2ic_{j+1} + is_j$ [19]. The intermediate sum and carry digits are chosen such that there is no carry signal propagation into the next digit position. This further entails that the digits is_j and ic_j can never be both '1' or '-1'.

The proposed addition algorithm uses a similar approach. In the proposed algorithm, a transfer carry of '1' from the previous digit position, automatically implies a carry of '-1' into the next digit position. The transfer carry into the j^{th} digit position indicates if either of the previous digit position operands is negative, as shown below.

(i) $tc_j = 0$, for positive previous operands, indicating a possible carry of '+1' from the previous digit position.

(ii) $tc_j = 1$, for negative previous operands, indicating a possible carry of '-1' from the previous digit position.

The intermediate sum bit is_j is a single bit (0 or 1) while the intermediate carry ic_{j+1} is represented as $ic_{j+1} = (ic^-_{j+1}, ic^+_{j+1}) = ic^+_{j+1} - ic^-_{j+1}$. The bit $ic^+_{j+1} = 1$ indicates $ic_{j+1} = 1$ while ic^-_{j+1} implies $ic_{j+1} = -1$. The j^{th} intermediate carry ic_j comes from the adjacent digit position, and is a function of the operands at that digit position, as well as the transfer carry. The final sum output at the j^{th} digit position, s_j, is a function of the bits is_j, ic^+_j, and ic^-_j, as well as the transfer carry from the previous digit position tc_{j-1}, thus giving totally parallel addition.

The proposed radix-2 addition algorithm can be summarized as under:

$$x_j + y_j = +2(ic_{j+1}) - is_j, \text{ for } tc_{j-1} = 0 \qquad (1)$$

$$x_j + y_j = -2(ic_{j+1}) + is_j, \text{ for } tc_{j-1} = 1 \qquad (2)$$

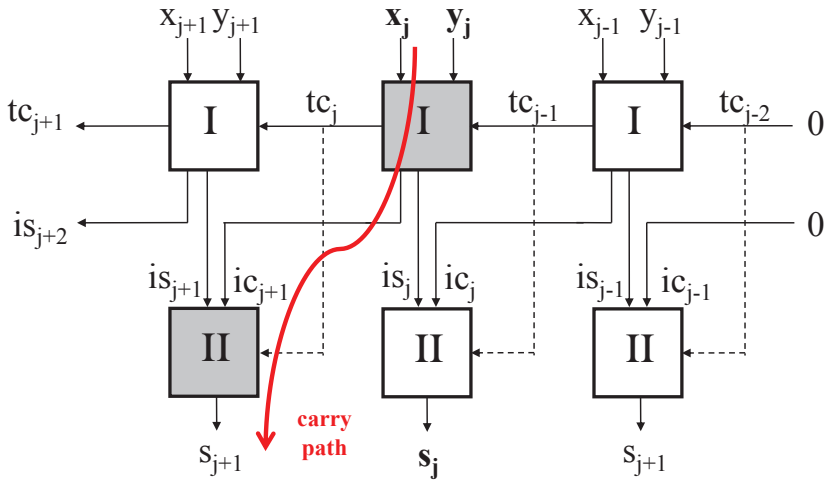


Figure 3.3: High-level block diagram of the proposed RBR addition algorithm

The above two equations essentially imply that for $tc_{j-1} = 0$, corresponding to a possibility of $ic_j = 1$ $(x_j + y_j = 2)$ from the previous digit position, we choose the sign for is_j as '-', while for $tc_{j-1} = 1$, corresponding to a possibility of $ic_j = -1$ $(x_j + y_j = -2)$ from the previous digit position, we choose the sign for is_j as '+'. Similar to the signed-digit characteristics, zero has a unique digit representation of $is_j = 0$ and $ic_{j+1} = 0$. The effective intermediate carry in turn depends on the bits ic_j^+ and ic_j^-. The bit $ic_j^+ = 1$ indicates a positive intermediate carry into the j^{th} position, while $ic_j^- = tc_{j-1} = 1$ directly implies a negative intermediate carry into the j^{th} position.

The proposed algorithm essentially implements the following steps:

(1) Determine the j^{th} intermediate sum bit is_j based on the j^{th} digit operands.

(2) Determine the $(j+1)^{th}$ intermediate carry bits, ic_{j+1}, based on the j^{th} digit operands and tc_{j-1}.

(a) If $tc_{j-1} = 0$, depending on the j^{th} digit operands, the bit ic_{j+1}^+ has a possibilty of being '1'.

(b) If $tc_{j-1} = 1$, the bit $ic^-_{j+1} = tc_{j-1} = 1$.

(c) If both $ic^+_{j+1} = ic^-_{j+1} = 1$, the effective carry into the (j+1) digit position is '0'.

(3) Next, assign the bits is_j, ic^+_j, and ic^-_j appropriately to the bits of the pre-sum output $a_j = (a^-_j, a^+_j)$ based on the transfer carry tc_{j-1}.

(a) For $tc_{j-1} = 0$, that is, non-negative adjacent operands, the effective carry into the j^{th} digit position can never be '-1', and so is_j is assigned to the negative pre-sum bit a^-_j, while the positive pre-sum bit $a^+_j = ic^+_j$.

(b) For $tc_{j-1} = 1$, the effective carry into the j^{th} digit position can never be +1 and so is_j is assigned to the positive pre-sum bit a^+_j, while $a^-_j = ic^-_j$.

(4) Generate the final sum output (s^-_j, s^+_j) by converting any '11' bit combination obtained at the pre-sum output as a result of the step (3) concatenation to '00', since it is outside the assumed encoding scheme.

Figure 3.3 depicts the high-level block diagram of the proposed RBR addition scheme. Block I receives the input operands x_j, y_j as well as the transfer carry from the previous digit position tc_{j-1}, and determine the intermediate carry and sum bits. Block II receives the j^{th} intermediate carry and sum bits, ic_j and is_j, and concatenates them, depending on tc_{j-1} to generate the final sum bits $s_j = (s^-_j, s^+_j)$. Depending on c_{j-1}, it forces a bit '1 and '0 respectively at a^-_j, and b^+_j of the interim sum respectively, or passes them through as is. The final operation is the elimination of the '11 combination to generate s_j.

The truth table for this addition algorithm is given in Table 3.2. The possible combinations for is_j and ic_{j+1} are given for $tc_{j-1} = 0$, and $tc_{j-1} = 1$. As seen from the truth table, for every input operand combination, the is_j and ic_{j+1} bits are never both assigned to the pre-sum bits a^-_j and a^+_j, and the concatenation operation leads to totally parallel addition.

As seen from the truth table, the positive intermediate carry bit ic^+_{j+1}, should be '1 for $is_j = 1$, and $tc_{j-1} = 0$, as well as, $x_j + y_j = 2$. For ease of circuit implementation, the

Table 3.2: Intermediate carry and sum digit possibilities for proposed addition algorithm

$x_j + y_j$	tc_{j-1}	ic^*_{j+1}	is^*_j
0	0	$(0,0^+)$	0^-
	1	$(0^-,0)$	0^+
1	0	$(0,1^+)$	1^-
	1	$(0^-,0)$	1
-1	0	$(0,0^+)$	1^-
	1	$(1^-,0)$	1^+
2	0 or 1	$(0,1^+)$	0^-
-2	0 or 1	$(1^-,0)$	0^+

*The superscript '+' indicates the bit is assigned to the positive pre-sum bit a_j^+, while '-' indicates the bit is assigned to the negative pre-sum bit a_j^-

Table 3.3: Variable values for $x_j + y_j = [-2,0,2]$

x_j^-	x_j^+	y_j^-	y_j^+	$x_j + y_j$	ic_{j+1}^+	is_j	tc_j	c_j
0	1	0	1	2	1	0	0	0
0	1	1	0	0	1	0	1	0
1	0	0	1	0	1	0	1	0
1	0	1	0	-2	1	0	0	0

algorithm is modified slightly in that, in addition to the above condition, ic_{j+1}^+, is also '1 for the input operand combinations $x_j + y_j = [-2,0,2]$. These input operand combinations are detected using the variable p_j, that is, $p_j = 1$, for $x_j + y_j = [-2,0,2]$. Table 3.3 indicates the values of the various bits for these operand combinations. This modification helps achieve the following simplification:

(1) For $tc_{j-1} = 0$, assign ic_{j+1}^+ to a_j^-, is_j to a_j^+

(2) For $tc_{j-1} = 1$, assign $\overline{ic_{j+1}^+}$ to a_j^+, is_j to a_j^-

As seen from the table, ic_{j+1}^+ = 1 for an additional input operand combination $x_j + y_j = -2$. For this operand combination, the effective carry ic_{j+1} should be '-1'. This is accomplished by forcing a bit '1' at the a_j^- bit position, after the pre-sum bits (a_j^-, a_j^+) are assigned. The input operand combination $x_j + y_j = -2$ is detected using the variable c_j.

3.2.3 Illustration of addition algorithm

The proposed addition algorithm is further explained with the aid of the following example. Consider the addition of two redundant binary numbers whose numeric values are equal to - 42 and - 7 respectively. The digit-level representation of these numbers is as shown in Figure 3.4 below. For the least significant digit position ($j = 0$), the bits tc_{j-1}, and ic_j^+ from the previous digit position, are assumed to be zero. The intermediate sum bit is_j for each digit position is found based on the input operand bits at the corresponding digit position. The intermediate sum and carry bits generated at each digit position are grouped together as shown in Figure 3.4. The intermediate carry bit ic_{j+1}^+ is asserted if the j^{th} bit position operands are non-negative, for $is_j = 1$, and also if $p_j = 1$, when $is_j = 0$.

The pre-sum bits (a_j^-, a_j^+) take on bit values derived from is_j or ic_j^+, at each digit position depending on the value of tc_{j-1}. The final sum bits are simply (a_j^-, a_j^+), except for the case of $(a_j^-, a_j^+) = (1,1)$, which is converted to '00' at the final sum output. However, if input operands in the adjacent digit position are such that $x_{j-1} + y_{j-1} = -2$ (as shown highlighted in the example at the $j = 4$ digit position), that is, for $c_{j-1} = 1$, a '1' must be forced at the a_j^- position, because this indicates a carry of '-1' from the previous bit position. In the digit-position $j = 5$, for which $c_{j-1} = 1$, forcing a '1' at the a_j^- position makes the effective $(a_j^-, a_j^+) = (1,1)$, which is then converted to $s_j = (0,0)$. As seen from the final output, the sum is -49 as expected.

3.2.4 Gate-level Implementation

The gate-level implementation of the proposed algorithm is derived below. The transfer carry into the next bit position indicates if either of the present operands is negative, i.e., $tc_j = 1$ if either $x_j = (1,0)$ or $y_j = (1,0)$. This yields the following equation for tc_i.

$$tc_j = x_j^- + y_j^-$$

As seen from the truth table, $s_j = 1$ for an odd number of ones in the input operand bits.

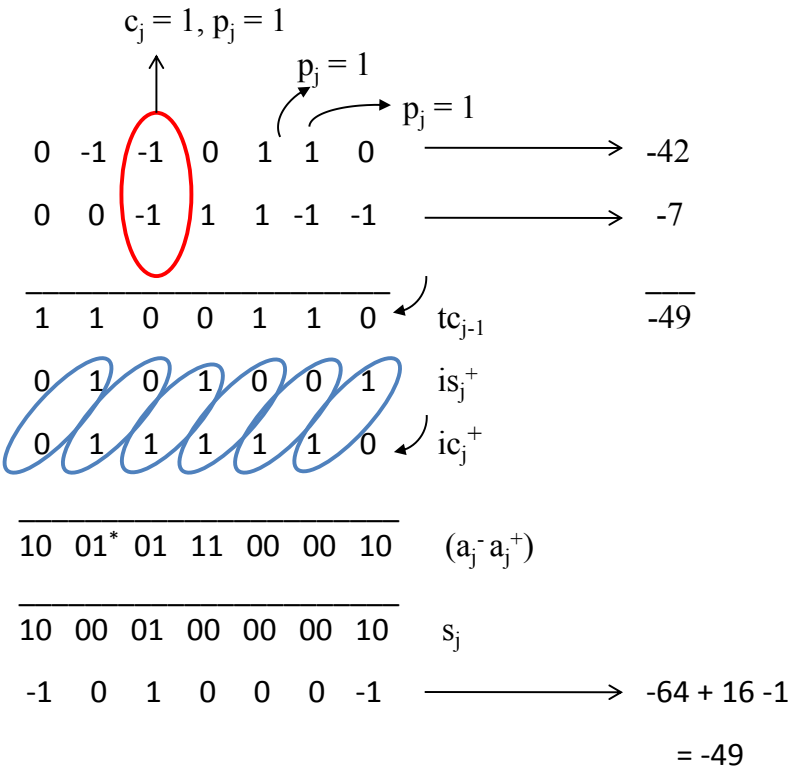


Figure 3.4: Addition Example

$$is_j = \overline{(x_j^- + x_j^+)} \oplus \overline{(y_j^- + y_j^+)}$$

Also, $ic_{j+1} = +1$, for $is_j = 1$, and $tc_{j-1} = 0$. Additionally, $ic_{j+1} = +1$ for $x_i + y_i = 2$. The bit p_j is asserted for $x_i + y_i = 2$. This, and the assumed rules from (1) and (2) help determine the logic expression for ic_{j+1}^+ as:

$$ic_{j+1}^+ = is_j . \overline{(tc_{j-1})} + p_j$$
$$\text{where, } \overline{p}_j = \overline{(x_j^- + x_j^+)} + \overline{(y_j^- + y_j^+)}$$

The pre-sum bits (a_j^-, a_j^+) are determined by the following concatenation operation depending on is_j and the previous digit position intermediate carry, ic_j^+, as controlled by tc_{j-1}.

$$a_j^- = tc_{j-1}.\overline{(ic_j^+)} + \overline{tc_{j-1}}.(is_j)$$
$$a_j^+ = \overline{tc_{j-1}}.(ic_j^+) + tc_{j-1}(is_j)$$

The pre-sum bits (a_j^-, a_j^+) may result in (1,1), which is outside the assumed representation, and hence must be converted to (0,0). This can be accomplished by the following equation:

$$s_i^+ = \overline{a_j^-}.a_j^+$$
$$s_i^- = a_j^-.\overline{a_j^+}$$

It follows that, in addition to the above-mentioned conditions, ic_{j+1}^+ is '1' for the $\{(1,0)+(0,1)\}$, $\{(0,1)+(1,0)\}$, and $\{(1,0)+(1,0)\}$ input operand combinations as well. For the first two cases, since $tc_j = 1$, the effective carry is zero, as desired. However, for the input operands $\{(1,0)+(1,0)\}$, the negative pre-sum bit a_j^- must be forced to '1' and the positive pre-sum bit a_j^+ to '1', as explained previously. This is incorporated in the conversion function itself, as shown below. The bit c_j is asserted for the (1,0)+ (1,0) input operand combination.

$$s_i^+ = \overline{a_j^-}.a_j^+.\overline{c_{j-1}}$$
$$s_i^- = (a_j^- + c_{j-1}).\overline{a_j^+}$$
$$\text{where, } c_{j-1} = x_{j-1}^-.y_{j-1}^-$$

These equations are summarized in Figure 3.5. The complete gate-level implementation for a one-digit adder is as shown in Figure 3.6.

3.3 Simulation Results

3.3.1 One-digit Redundant Binary Adder Performance Comparisons

In order to evaluate the performance of the proposed RBR adder, it is compared with the reference RBR adder [29], RBA_ref. Both the adder designs are implemented and simulated using HSPICE, as well as the Verilog-based model.

$$tc_j = {x_j}^- + {y_j}^-$$

$$is_j = \left(\overline{{x_j}^- + {x_j}^+}\right) \oplus \left(\overline{{y_j}^- + {y_j}^+}\right)$$

$$ic_{j+1}^+ = is_j . \overline{tc_{j-1}} + p_j$$

where, $\overline{p_j} = \left(\overline{{x_j}^- + {x_j}^+}\right) + \left(\overline{{y_j}^- + {y_j}^+}\right)$

$${a_j}^- = tc_{j-1} . \left(\overline{ic_j^+}\right) + \overline{tc_{j-1}} . (is_j)$$

$${a_j}^+ = \overline{tc_{j-1}} . \left(ic_j^+\right) + tc_{j-1} . (is_j)$$

$${s_i}^+ = \overline{{a_j}^-} . a_j^+ . \overline{c_{j-1}}$$

$${s_i}^- = (a_j^- + c_{j-1}) . \overline{{a_j}^+}$$

where, $c_{j-1} = {x_{j-1}}^- . {y_{j-1}}^-$

Figure 3.5: Summary of equations for proposed RBR adder

In order to test the adders in a realistic environment using HSPICE, two-digit RBR adders are constructed, with the carry outputs of one adder driving the inputs of the other adder. The adders are characterized in terms of critical path delay, average energy consumption, and EDP. The adders are tested across a majority of input operands, and realistic carry input combinations to yield the results as shown in Figure 3.7.

As seen from Figure 3.7a, the proposed design has a lower critical path delay compared to the reference circuit, higher average energy consumption, and comparable energy-delay product. As seen from the delay plots, at lower supply voltages, the reference design has a longer delay due to signal slope degradation for the chain of transmission gates. In the proposed design, the transmission gates are placed always between logic gates.

A similar characterization using Verilog yields the plots shown in Figure 3.8.We see that, the delay and energy trends are similar to those obtained from HSPICE, though the absolute numbers are slightly different. In both cases, the proposed design has lower critical path delay and slightly higher average energy consumption. In the rest of the chapter, we will present results using Verilog-based characterization, owing to its easy portability for high complexity designs.

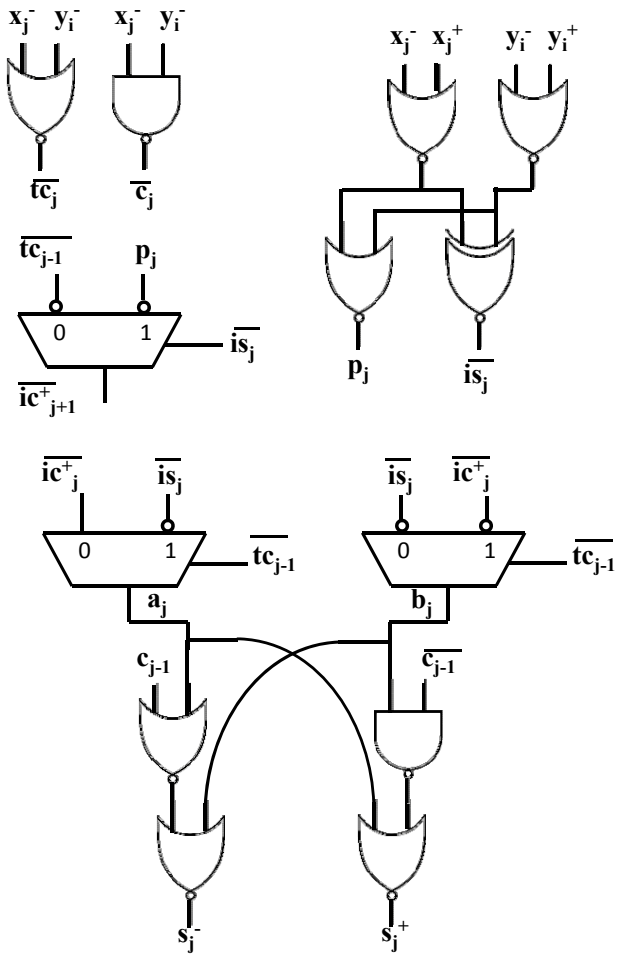
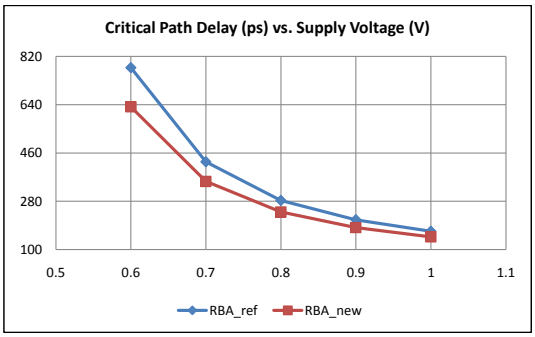


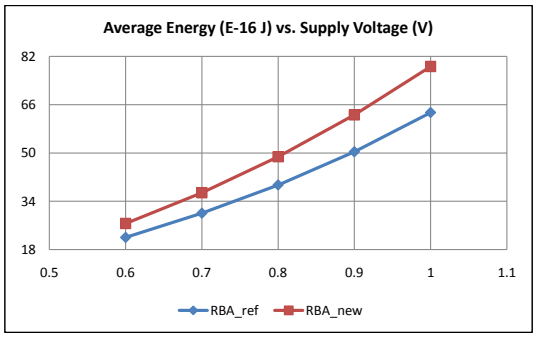
Figure 3.6: Gate-level implementation for proposed 1-digit radix-2 RBR adder (RBA_new)

3.3.2 16-digit Redundant Binary Adder Performance Comparisons

In order to demonstrate the energy-efficacy of RBR systems, the performance of two's complement and RBR adder architectures are compared in terms of delay, average energy consumption and EDP. We consider 16-bit two's complement Carry Select (CSA), and Carry Look Ahead (CLA) adders for the two's complement case, and, 16-digit redundant binary adder based on RBA_ref and the proposed parallel adder design, RBA_new, for the RBR case. The simulation results for delay, average energy and EDP are as shown in Figures 3.9a, 3.9b and 3.9c, respectively.

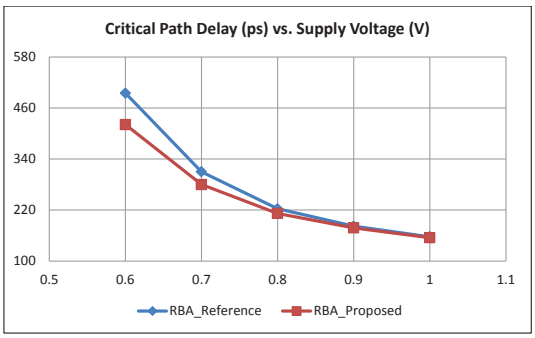


(a) Critical Path Delay

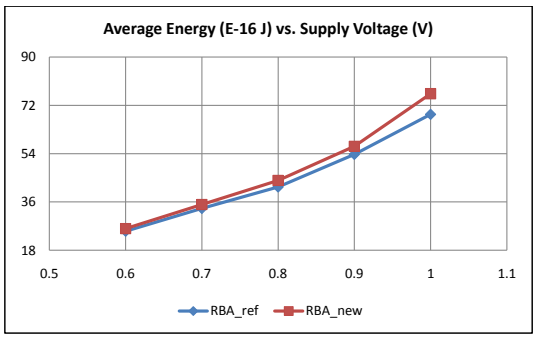


(b) Average Energy Consumption

Figure 3.7: HSPICE characterization: 1-digit RBR Adder Performance Comparisons



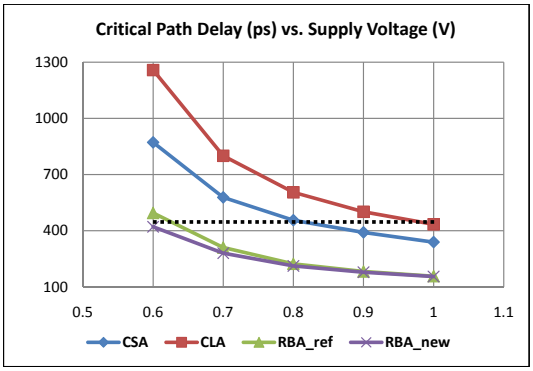
(a) Critical Path Delay



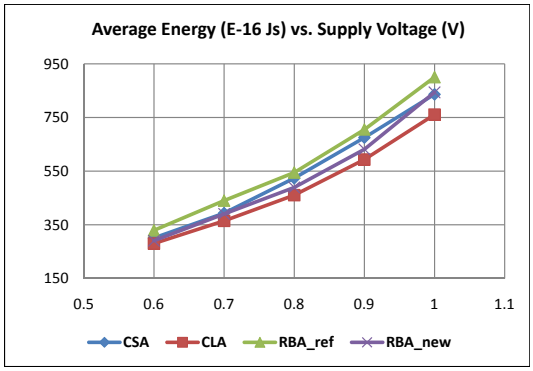
(b) Average Energy Consumption

Figure 3.8: Verilog-based characterization: 1-digit RBR Adder Performance Comparisons

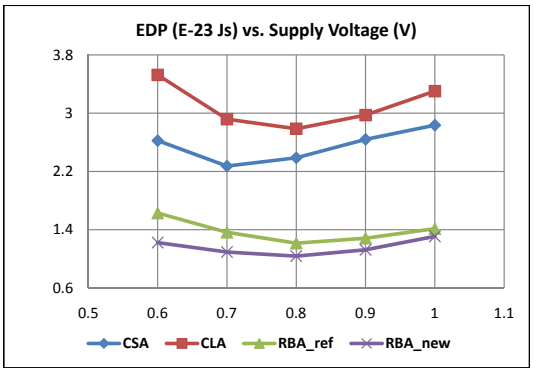
We see that the RBR parallel adders have a very short critical path delay when compared to the two's complement CSA and CLA adders. The RBR adder designs, RBA_ref and RBA_new, have comparable critical path delay. In terms of the average energy consumption, the CLA has significantly lower energy compared to other adders. The RBR and CSA adders have comparable average energy consumption, although the CSA has higher critical path delay. However, when we compare the performance for nearly the same critical path delay, i.e. for iso-throughput, RBA_ref has a 37% lower energy consumption compared to CSA and 56% lower energy consumption compared to CLA. For the same iso-throughput case, RBA_new has a 44% lower energy consumption compared to CSA and 62% lower energy consumption when compared to CLA.



(a) Critical Path Delay



(b) Average Energy Consumption



(c) EDP

Figure 3.9: 16-bit Adder Performance Comparisons

At higher supply voltages, the proposed design has comparable energy to RBA_ref. Since the distribution of randomly-generated input operands for both sets of redundant binary adders is exactly the same, and also since the Verilog-model does predict higher average energy consumption for the proposed single-digit adder, this comparable trend in average energy is solely due to the signal switching and propagation paths of the two adders. Thus, the proposed design is a competing design for RBR addition. In terms of EDP, RBA_new has a superior performance with a nearly 1.9x reduction compared to CSA, 2.7x reduction when compared to CLA, and comparable performance with respect to RBA_ref.

The delay distribution plots for the reference and proposed 16-digit RBR adders are shown in Figure 3.10 and Figure 3.11 respectively. As seen from the plots, a majority of the input operand combinations for the reference RBR adder have the critical path delay, while

the proposed design has a majority of input operand combinations at 90% of the critical path delay.

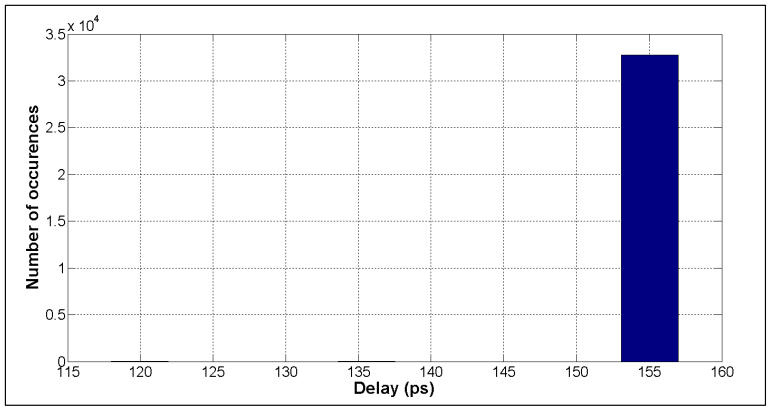


Figure 3.10: Delay distribution histogram: 16-digit RBA_ref

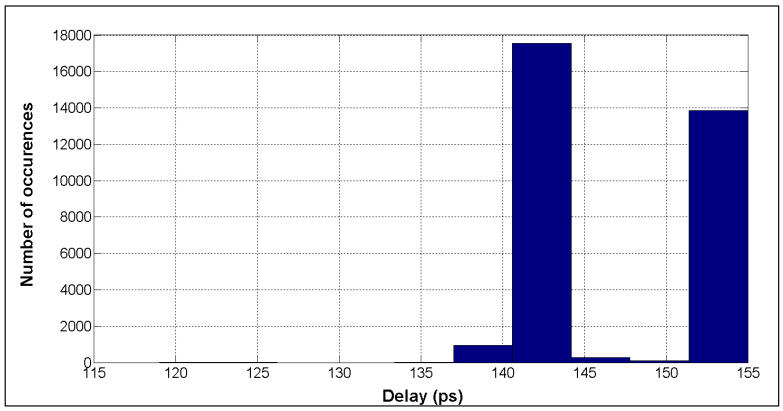
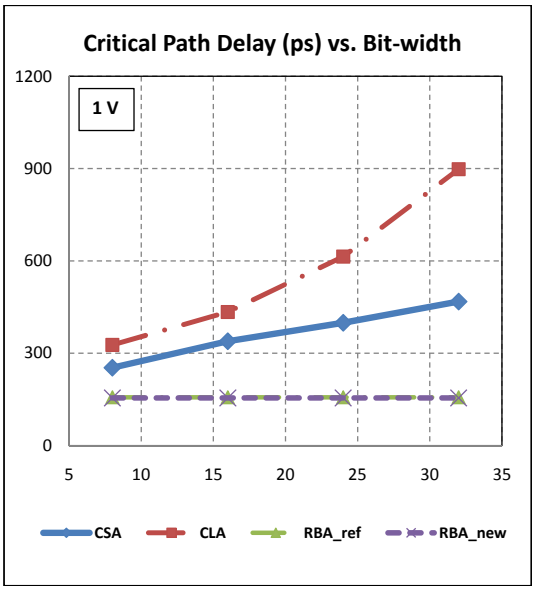


Figure 3.11: Delay distribution histogram: 16-digit RBA_new

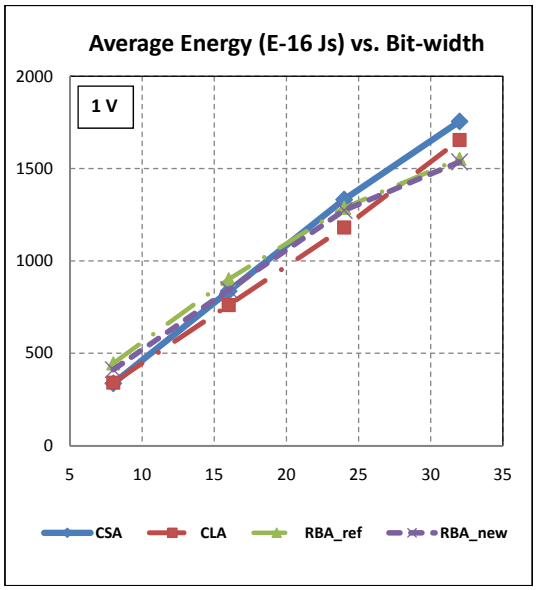
3.3.3 Effect of bit-precision

The performance trends for all adders as a function of bit precision at nominal supply and 0.8 V supply, is given in Figure 3.12 and Figure 3.13 respectively. As seen from Figures 3.12a and 3.13a, the critical path delay of all the RBR and Hybrid multipliers is constant and independent of bit-width for all the bit-precision nodes. In contrast, for both the two's complement adders, the critical path delay increases with increasing bit-precision. All the adders have increasing average energy trends with increase in bit-precision, as expected. In terms of EDP, the RBR adders have a superior EDP performance at all the bit-precision

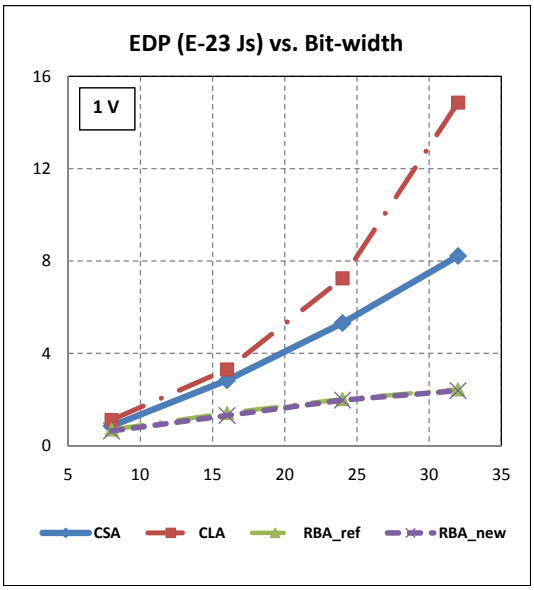
nodes. As the supply voltage is reduced from 1 V to 0.8 V, the critical path delay for all the adders at all bit-precision nodes increases, while average energy shows quadratic reduction.



(a) Critical Path Delay



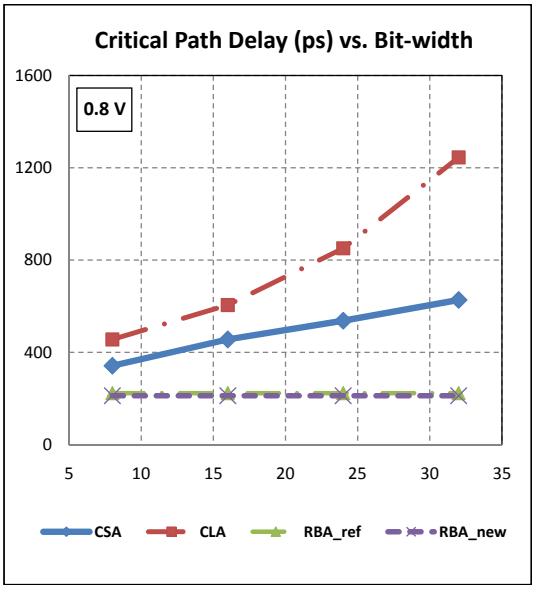
(b) Average Energy Consumption



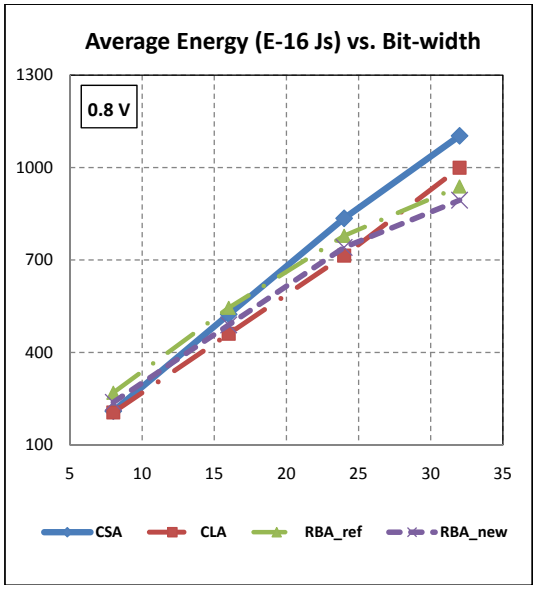
(c) EDP

Figure 3.12: Adder Architectures: Effect of varying bit precision at nominal supply

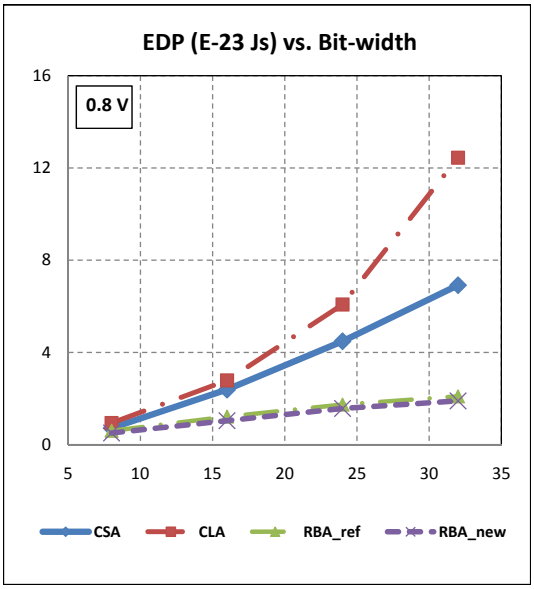
Figure 3.14 depicts the EDP values of the different adders for 8, 16, 24, and 32-bit precisions at iso-throughput. The operating supply voltages are depicted for each adder.



(a) Critical Path Delay



(b) Average Energy Consumption



(c) EDP

Figure 3.13: Adder Architectures: Effect of varying bit precision at 0.8 V

Both RBR adders operate at 0.6 V for iso-throughput case for all bit-widths. As seen from the figure, since the critical path delay of RBR adders is independent of bit-width, the EDP values change only marginally due to increase in average energy with increase in bit-width. However, the CSA and CLA adders have increased delay as well as increase in energy with

bit-width increase, and therefore, their EDP values increases significantly with bit-width. Thus, the EDP performance of RBR systems improves compared to two's complement systems for larger bit-width.

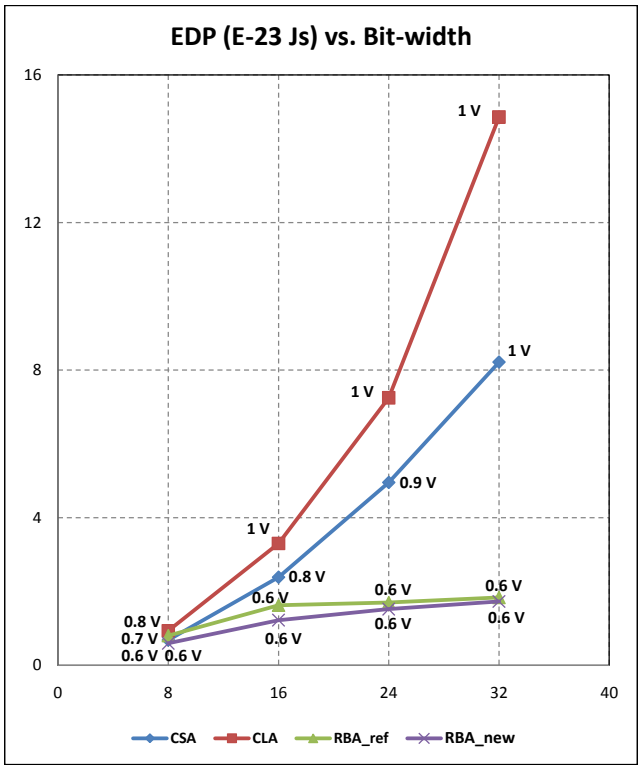


Figure 3.14: Adder architectures: Effect of bit-precision

3.3.4 Area Comparison

Figure 3.15 compares the transistor count for the two's complement and 16-digit RBR adder designs. The proposed adder design has relatively higher layout footprint compared to the other designs. Between the two RBR adder designs, the proposed design has a nearly 28% area overhead. In comparison to two's complement adders, the proposed adder design has a 11% larger area compared to CSA, and a 17% larger area compared to CLA.

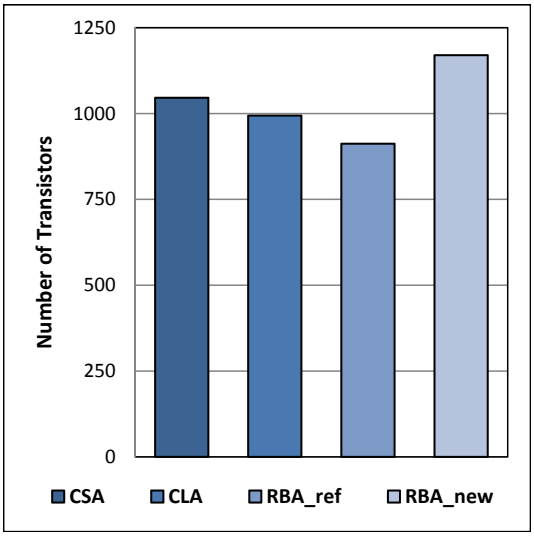


Figure 3.15: Transistor Count for 16-bit adder designs

Chapter 4

MULTIPLIER ARCHITECTURES

High-speed multipliers are essential components of DSP architectures. A multiplication operation essentially consists of partial product generation and accumulation. From a high-level perspective, high speed multipliers may be classified into the following three types [23]: parallel multipliers, serial multipliers, and array multipliers. In parallel multipliers, the partial products are generated in parallel, and the accumulation phase consists of multi-operand adders. The serial multiplier sequentially generates the partial products, and adds each newly generated partial product to the previously accumulated sum. Array multipliers have no separate circuits for partial product generation and accumulation; the two operations are performed simultaneously. Although array multipliers exhibit low latency, they have increased implementation complexity.

In this work, we explicitly focus on parallel multiplier architectures for computation-intensive applications. Several RBR multiplier architectures are investigated and their performance compared to two's complement systems. Two new parallel multiplier architectures are proposed. These include an RBR multiplier which has both of its operands in the RBR form, and a hybrid multiplier which has the multiplicand in RBR form and the other operand in two's complement form. RBR multipliers may be used for systems where both operands are obtained from other RBR computation units. The hybrid multiplier, on the other hand, would be apt for systems where multiplicand is obtained at processing time and the multiplier operand is pre-determined. The assumed encoding scheme for the RBR operands is the same as the representation used for the adders.

4.1 Two's Complement Multiplication

Two's complement arithmetic is the most popular form of computation for data-path components. For two's complement multiplication, one popular scheme for fast accumulation of partial products is the Wallace tree structure [40], which normally uses a tree of binary full adders or (3,2) counters. The (3,2) counters achieve a 3:2 compression ratio, and any

carry propagation is deferred until there are only two partial sums left to be added. The final stage addition is generally performed using a fast parallel adder. In order to achieve high throughput with two's complement multipliers, these designs are heavily pipelined; for instance, six levels of pipelining for 16-bit multipliers.

Higher compression ratios can be achieved by using higher-input counters such as (4,2), (7,3) counters [35, 30]. However, as the compression-level increases, the processing load of a single-stage increases, consequently increasing the stage delay. Compressor-based multipliers have been extensively researched, and the (4,2) compressor approach is presently the most popular one. Figure 4.1 shows a (4,2) compressor based multiplier with a CSA adder in the last stage. Santoro et al. [41] combined a pipelined Wallace tree with (4,2) compressors, and an iterative accumulation approach to implement a 64 x 64-bit pipelined multiplier called the Stanford Pipelined Iterative multiplier (SPIM). Nagamatsu et al. [31] built a non-pipelined 32 x 32-bit multiplier that used Booth's algorithm, (4,2) compressor based Wallace tree and a carry-select final stage adder for fast multiplication. Mori et al. [29] and Goto [14] used a similar approach in their multipliers with the final stage adder being different. Okhubo et al. presented a multiplier based on a Wallace tree of (4,2) compressors in [32]. They proposed a new (4,2) compressor design, shown in Figure 4.2, which is very compact and has only three gate delays. This is the multiplier architecture against which we compare our proposed designs.The parallel adder in the final stage is implemented using a carry-select adder, because of its superior EDP performance compared with the carry look-ahead scheme (from Figure 3.9c).

In case of two's complement multiplication for signed operands, if the generalized Wallace tree structure has to be used, the partial products must be sign extended throughout the summation tree to account for the negative weight of the sign-bit of the multiplicand. In case of a negative multiplier operand, the final partial product bits are inverted and a '1' is added to its least significant bit position. To eliminate the negative-weighted bits from the partial product matrix, Baugh and Wooley suggested an efficient manipulation of the bits in the partial-product matrix [30]. In the modified Baugh-Wooley multiplier of

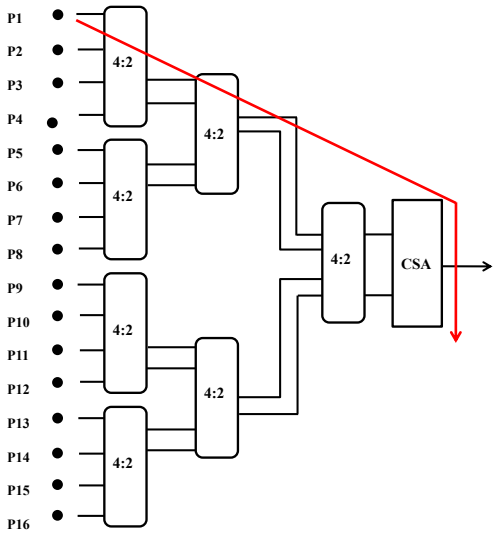


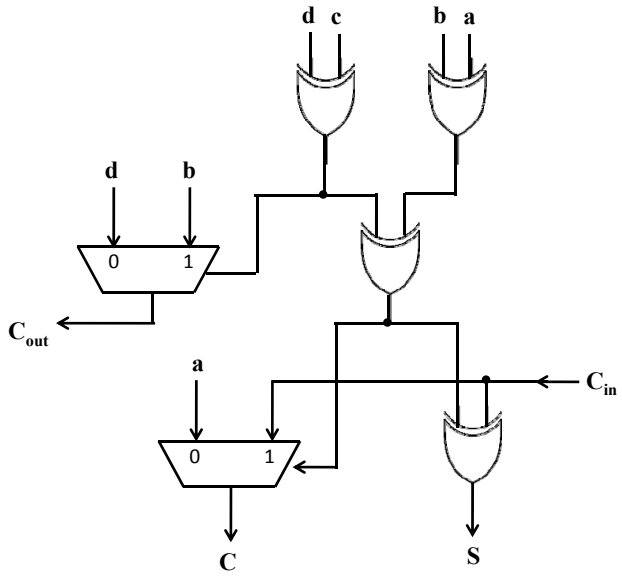
Figure 4.1: Two's complement multiplier - Partial product accumulation using 4:2 compressors

[23, 4, 35], by adding a few entries to the bit matrix, signed multiplication can be implemented with the same number of addition levels and almost the same gate complexity as the unsigned Wallace tree multiplication scheme. Throughout this work, we consider this modified Baugh-Wooley design for two's complement signed multiplication.

4.2 Existing RBR Multiplication schemes

Conventional RBR multipliers typically use a binary tree of RBR parallel adders for summing up the partial products [47]. In these architectures, N partial products are grouped into pairs and added using a tree of N or higher digit RBR adders. Since addition at each tree level is performed in constant time, the multiplication time is proportional to $O(logN)$. The enhanced versions of this multiplier are based on reducing the number of operands for the partial product tree addition by using modified Booth encoding [29, 1, 20] and employing intelligent RBR coding schemes [42, 52, 51, 53].

In the work presented by Makino et al. in [29], a 54x54-bit fast multiplier based on radix-2 RBR is described. In this architecture, the incoming two's complement operand is pre-processed by using modified Booth encoding to halve the number of partial products.



$$a + b + c + d + C_{in} = 2(C + C_{out}) + S$$

Figure 4.2: 4:2 Compressor from [32]

Further reduction in the number of partial products is achieved by grouping together pairs of normal binary partial products to form redundant binary partial products. A total of 15 redundant binary partial products are then summed together using a 4-level binary tree of redundant binary adders.

In [42], an 8x8-bit radix-2 redundant-binary hybrid complex number multiplier is introduced. This algorithm reduces the computation to two redundant-binary multiplications, one for the real part, and one for the imaginary part. Both are implemented using a pipelined binary tree of RBR adders.

A radix-4 16x16-bit RBR complex number multiplier is discussed in [52]. This multiplier accepts RBR operands, and so the Booth strategy of encoding consecutive digits cannot be applied directly. Filtering using binary signed digit recoders is performed to reduce it to a form suitable for radix-4 recoding. This in turn allows operands to halve the number of summands to be added in each of the three real multiplier units. The disadvantage of this scheme is that the BSD recoder is twice as complex and slower than

a regular Booth recoder inputting a binary operand. Similarly, in [20], a radix-4 Booth encoding-based 16-bit multiply-and-accumulate (MAC) unit using a 16x16 redundant binary multiplier and 35-bit redundant binary accumulator is implemented. The partial product summation is performed using a pipelined redundant binary addition tree. The method in [17] proposes a novel technique for Booth encoding of redundant binary operands for partial-product reduction.

The work in [51] describes a radix-2 RBR system that implements an inner-product processer, and the one in [53] describes iterative computations of division and square root algorithms in RBR. Similar to [29], these designs rely on partial product reductions due to efficient pairing and modified Booth encoding of the normal binary partial products. The actual summing operation of the RBR partial products is, as in the preceding examples, implemented as a binary tree of RBR adders.

Ferguson and Ercegovac designed a multiplier that accepts both operands in redundant form [10]. They employ hybrid techniques such as recoding the multiplier to minimally-redundant radix-4, and transforming the redundantly represented multiplicand into radix-2k carry/save form by splitting it into k-digit groups, and assimilating them using conventional (3,2) counters. Their design trades-off area and power for significant speed-up when compared to a conventional two's complement multiplier.

4.3 Redundant Binary Multiplier Architectures using proposed radix-2 RBA

Partial product accumulation using a binary tree of RBR adders is extremely popular owing to its regularity of design and low latency. For such multiplier architectures, the performance of the RBR adder dictates the performance of the multiplier. Figure 4.3 shows the block diagram of an 8x8-bit RBR tree multiplier. In this work, we build and compare RBR tree multipliers that accept both operands - multiplicand and multiplier - in the redundant binary form. These RBR tree multipliers include designs that are based on both the proposed redundant binary adder design and the reference adder design.

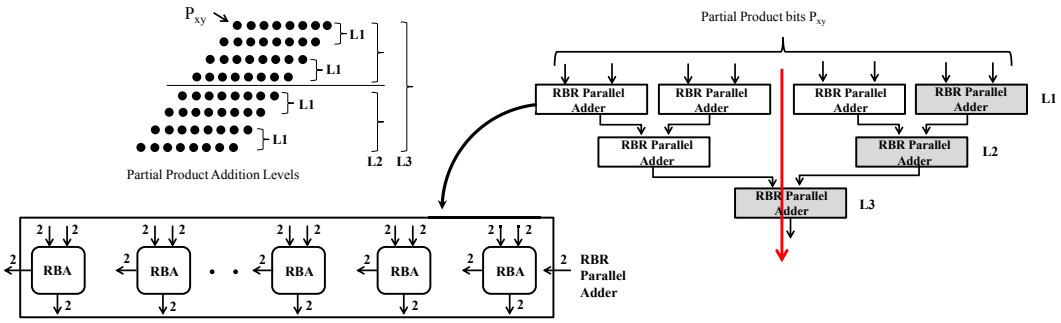


Figure 4.3: Block diagram of an 8x8-bit RBR Tree Multiplier

4.3.1 Partial Product generation

If the RBR multiplicand is represented by $b_j = (b_j^-, b_j^+)$, and the RBR multiplier is represented by $a_j = (a_j^-, a_j^+)$, where a_j, $b_j \in [0, 1, -1]$, then the partial product $p_{ij} = (p_{ij}^-, p_{ij}^+)$, where $p_{ij} \in [0, 1, -1]$ can be generated using the following expressions.

$$p_{ij}^+ = b_j^- . a_i^- + b_j^+ . a_i^+$$

$$p_{ij}^- = b_j^+ . a_i^- + b_j^- . a_i^+$$

4.3.2 Partial Product accumulation

The partial product accumulation for RBR tree multipliers essentially consists of a binary tree of RBR parallel adders (or 2:1 digit-level compressors), as shown in Figure 4.4 for a 8x8-bit multiplier. Each pair of the eight partial products are added, creating four partial sums. The partial sums are again grouped together and added, thus forming a binary tree. In general, for an *nxn* multiplier, the n partial products are added using a binary tree with $O(logn)$ levels. Although this structure leads to high throughput, the average energy consumption of tree multiplier designs is much higher, owing to the gate-intensive RBR adder blocks. The performance plots of the two tree multipliers is presented in section 4.6.

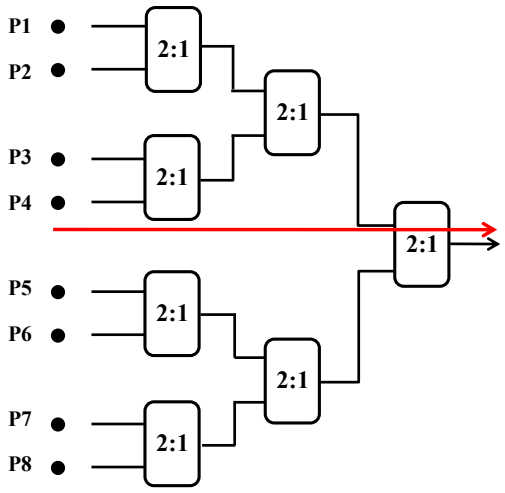


Figure 4.4: Partial product accumulation in a 8x8-bit RBR Tree Multiplier

4.4 Proposed Digit-parallel Hybrid Multiplier Architecture

Hybrid representation systems process both two's complement and redundant number representation operands. Phatak and Koren presented a unified framework for redundant number representations in [38], which they called the hybrid signed digit. This unified framework includes the two's complement representation and the signed digit representation as special cases. The hybrid signed digit system allows some digits to be signed while others may be unsigned, and permits non-uniform distance between the signed digits.

The carry-free arithmetic property of redundant representations has helped realize MSD-first computation systems [22, 21, 49, 9, 7, 43, 15], where MSD stands for most-significant-digit. A hybrid number representation, where one input operand is redundant signed-digit, and the second operand is in two's complement representation is explored in [15, 44, 36]. References [15], [44] illustrate the use of "PPM" (plus-plus-minus), and "MMP" (minus-minus-plus) operators that process one redundant binary digit and a two's complement bit, to build arithmetic computation units that are digit-serial. [36] elucidates the use of these operators for both radix-2 and radix-4 hybrid arithmetic. In [44], a high-speed pipelined digit-parallel two's complement multiplier (input and output operands are

in two's complement form) that internally uses a radix-2 hybrid number representation is presented. It also demonstrates the use of the hybrid number scheme to implement non-restoring division and square-root algorithms.

In this work, a digit-parallel hybrid multiplier architecture is presented. It multiplies a radix-2 RBR operand with a two's complement operand to produce digit-parallel, RBR multiplication output. In the radix-2 redundant representation, each digit of the multiplicand is represented using two bits. The encoding scheme for the radix-2 redundant binary multiplicand is the same as that of the adders; the borrow-save scheme is used, where the digit-set $\{0,1,-1\}$ is represented by $\{00,01,10\}$ respectively.

4.4.1 Digit-Parallel Hybrid Multiplication Algorithm

The algorithm for the digit-parallel hybrid multiplication scheme is described below.

Let a_i represent the bits of the two's complement multiplier, and $b_j = (b_j^-, b_j^+)$ represent the digits of the radix-2 redundant binary multiplicand.

1. Generate the partial products $a_i b_j$ for each digit position of the multiplicand.

2. If the multiplicand digit is negative $(b_j = -1)$, the partial product is simply equal to the two's complement of the multiplier. This essentially allows the generation of partial products with a single-bit at each digit position. In contrast, in conventional partial product generation, since the multiplicand is comprised of two bits, and the multiplier is a single bit, the partial-product would comprise of two bits at each digit position.

3. Since the multiplier is in two's complement form, and signed numbers are allowed, a '1' in the most-significant-bit (MSB) position, $(a_{i(MSB)} = 1)$ indicates a negative value. In order to account for negative multipliers, the partial product generated using this bit, $a_{i(MSB)} b_j$, is subtracted from the result of the previous partial product accumulation. Thus, for a positive multiplier $(a_{i(MSB)} = 0)$, a zero is subtracted, while for a negative multiplier $(a_{i(MSB)} = 1)$, the partial product is subtracted.

4.4.2 *Partial product generation*

The partial product generation logic for the hybrid scheme is described in this section. For a negative multiplicand digit ($b_j = -1$), the partial product is simply the two's complement of the multiplier. Since, the two's complement of a number is simply the bit-wise complement of the number added to 1, for each negative b_j, the partial product can also be generated by inverting the multiplier bit, and adding a bit '1' in the partial product matrix. The partial product generation logic can be summarized as follows

1. If ($b_j^- = 0$), then simply multiply a_i and b_j^+ to generate the partial product. This takes care of multiplication by digit 0 and 1.

2. If ($b_j^- = 1$), then invert a_i to generate the partial product. In other words, for $b_j^- =$ 1 , the partial product is basically the inverted multiplier bit. This takes care of multiplication by digit -1.

Thus the partial product generation scheme can be represented by the following equation. Note that this is just a multiplexing operation.

$$p_{ij} = b_j^- . \overline{(a_i)} + b_j^+ . (a_i)$$

4.4.3 *Illustration of the Hybrid Multiplication Scheme*

The hybrid multiplication scheme is explained with the help of the following multiplication examples. Figure 4.5 depicts example-1, which is the multiplication of a 4-digit redundant binary number -4 with a 4-bit two's complement positive multiplier +6. As seen from example-1, for the least significant three multiplicand digits, the partial products are generated similar to the conventional multiplication algorithm. However, for the most significant multiplicand digit, which is '-1', the partial product is simply the inverted multiplier bit. These inverted multiplier bits are highlighted using the red circles. Since ($b_3 = -1$) a '1' is added to the partial product matrix at digit position 3. This forms the two's complement

of the multiplier. Although the multiplier is positive $(a_{i(MSB)} = 0)$ indicating that the last partial product should be zero, since the multiplicand has a MSD of '-1', the final partial product bits must be subtracted. Thus, the subtraction in the final step takes care of this condition. Figure 4.6 shows example-2 which illustrates multiplication with a negative two's complement operand.

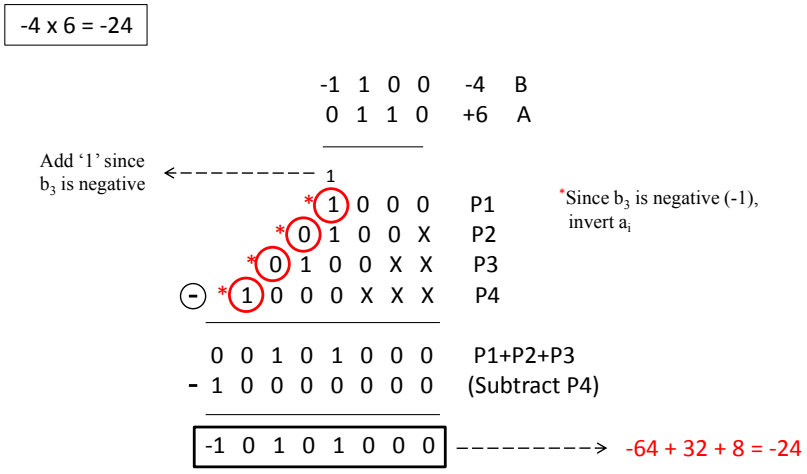


Figure 4.5: Hybrid Multiplication: Example - 1

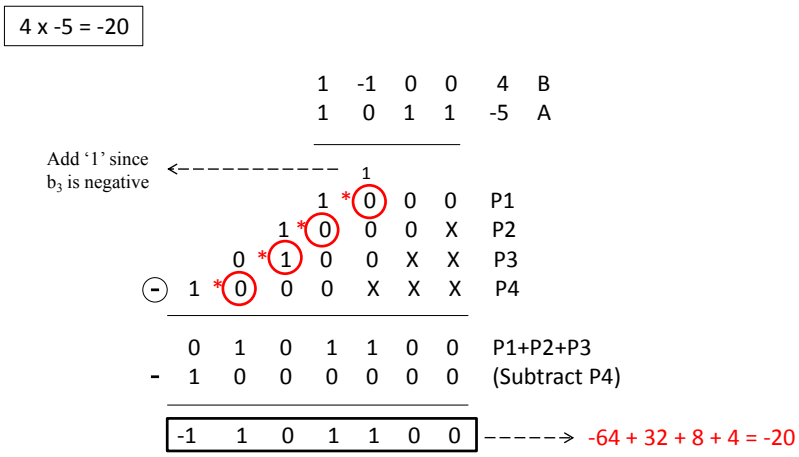


Figure 4.6: Hybrid Multiplication: Example - 2

4.4.4 Plus-Plus-Minus (PPM) Operator [15]

Once the partial product matrix is generated, all the partial products (except the last one) have to be added using a tree of full adders. However, since the last partial product is to be subtracted from the previously accumulated sum, the last stage requires an arithmetic block that can add two vectors generated by the tree, and also, subtract the last partial product vector. This is accomplished by using the "Plus-Plus-Minus" or PPM operator that was first designed for on-line (or MSD-first) arithmetic systems that used redundant binary intermediate forms [15, 36]. Figure 4.7 shows the circuit diagram for the PPM operator, which essentially performs parallel addition of a radix-2 redundant binary digit and a two's complement bit. Fig. 6 shows the circuit diagram for the PPM operator, which essentially performs parallel addition of a radix-2 RBR digit and a two's complement bit. This operator assumes an encoding scheme of $\{00 \text{ or } 11, 01, 10\}$ for the radix-2 digit set $\{0, 1, -1\}$ respectively. The PPM operator logic function can be summarized as $[S^*, S^{**}] = x_i^+ - x_i^- + y_i$, where $x_i = [x_i^-, x_i^+] = x_i^+ - x_i^-$ is a redundant binary digit, and y_i is a two's complement bit.

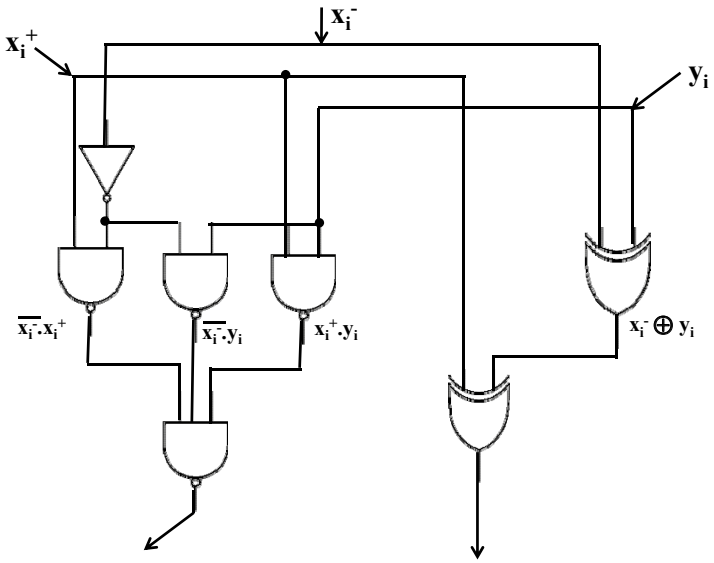
$$S^{**} = x_i^- \oplus x_i^+ \oplus y_i$$

$$S^* = \overline{x_i^-}.x_i^+ + \overline{x_i^-}.y_i + x_i^+.y_i$$

4.4.5 Proposed Plus-Plus (PLPL) Operator

For the blocks in the tree edges, which generate the lower significant output bits, PPM operators are required in order that the lower significant bits comply with the PPM addition scheme. However, these blocks have to process only two bits and add them to produce the output in PPM format. This can be done using another operator that can be obtained by reducing the Boolean expression of the PPM operator to yield the equations given below. Since this operator simply adds two bits, x_i, y_i, it is known as the PLPL operator. Figure 4.8 shows the logic diagram.

$$S^{**} = x_i \oplus y_i$$

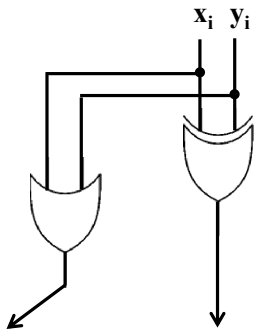


$$S^* = \overline{x_i^-}.x_i^+ + \overline{x_i^-}.y_i + x_i^+.y_i \qquad S^{**} = x_i^- \oplus y_i \oplus x_i^+$$

$$-S^{**} + 2S^* = -x_i^- + x_i^+ + y_i$$

Figure 4.7: PPM Operator [15]

$$S^* = x_i + y_i$$



$$S^* = x_i + y_i \qquad S^{**} = x_i \oplus y_i$$

$$-S^{**} + 2S^* = x_i + y_i$$

Figure 4.8: Proposed PLPL Operator

4.4.6 Block Diagram of the Digit-parallel Hybrid Multiplier

Figure 4.9 shows the block diagram of 4x4 hybrid multiplier architecture. The diagram depicts all the bit values for the multiplication example in Figure 4.5 that multiplies the RBR and two's complement operands '-4' and '+6' respectively. The incoming redundant binary multiplicand digits are combined with the two's complement multiplier bits to generate the partial product bits. The partial product bits in turn are added using a tree of full adders (FA) as shown in Figure 4.9. The partial product generator, as discussed in Section 4.4.2, takes care of complementing the multiplier bits for any negative multiplicand digit. The last stage PPM operator implements the dual functions of adding the tree-accumulated sum vectors as well as subtraction of the last partial product in case it is negative. This leaves adding the bit '1' to the partial product matrix for any negative multiplicand digit at the corresponding digit position. This is accomplished by applying the negative bit of each redundant binary digit, b_j^-, to the inputs of the first level of full adders as shown in the figure. The final output digits of the hybrid multiplier are generated in parallel by the PPM units. Since the PPM outputs are obtained by concatenating the output bits of adjacent PPM blocks, the edges of the tree that generate the lower significant output bits are comprised of the newly introduced PLPL blocks. Finally, since the PPM digit-set includes $\{00 or 11, 01, 10\}$, while the assumed digit-set is $\{00, 01, 10\}$, any '11' bit combination generated, must be converted to '00'. This is accomplished by the following conversion function. Let (a_i^-, b_i^+) be the final concatenated output digits of the PPM unit, and (m_i^-, m_i^+) be the final multiplier outputs at the i^{th} digit position.

$$m_i^+ = \overline{a_i}.b_i = \overline{a_i + \overline{b_i}}$$

$$m_i^- = a_i.\overline{b_i} = \overline{\overline{a_i} + b_i}$$

4.5 Proposed Digit-parallel Redundant Binary Multiplier Architecture

In this section, a new architecture for a RBR multiplier which has both its operands in RBR form is introduced. The algorithm for this multiplication scheme is derived from the hybrid

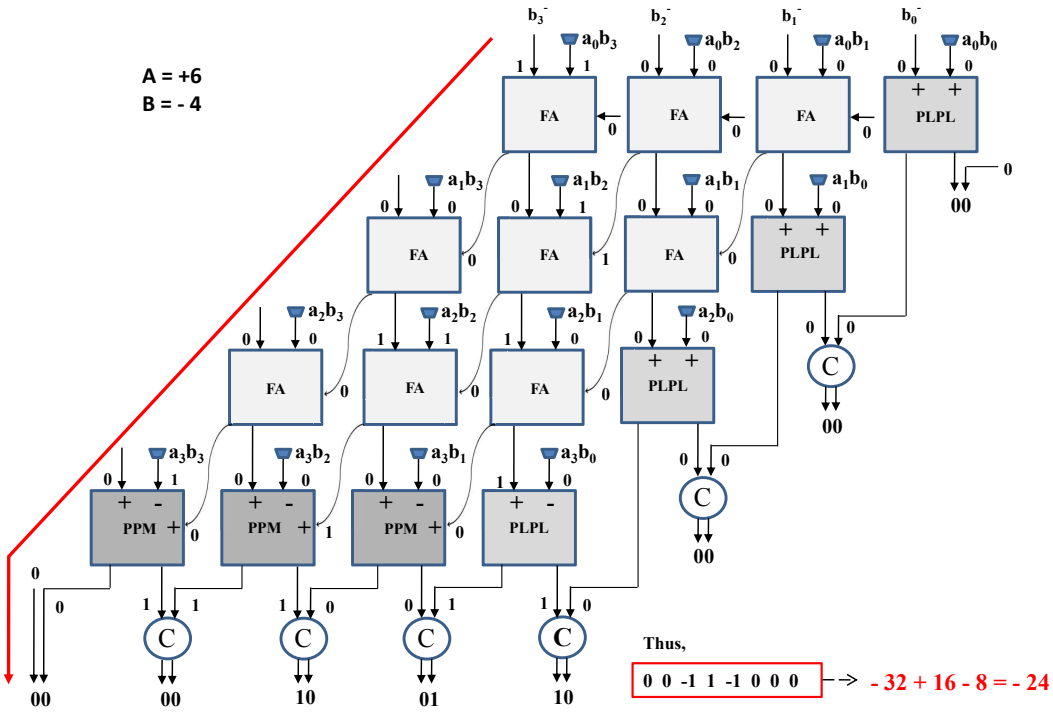


Figure 4.9: Block diagram of proposed hybrid multiplier architecture

multiplication scheme, described in detail in Section 4.4. The RBR multiplication algorithm generates n+1 partial products of n+1 bit each for n-digit multiplication, as opposed to 2n-bit partial products that are generated in case of the conventional method. As a result the implementation complexity and thus the average energy consumption are significantly reduced.

4.5.1 Digit-Parallel Novel Redundant Binary Multiplication Algorithm

Consider a RBR multiplicand, -5, represented by four bits $(-1\ 0\ 1\ 1)$, which is to be multiplied by the RBR multiplier, -11, represented by $(-1\ -1\ 0\ 1)$. This multiplication can also be performed by splitting up the multiplier into two two's complement numbers, the number 1, that is, $(0\ 0\ 0\ 1)$ and the number 12, represented as $(1\ 1\ 0\ 0)$, generating the multiplication with 1 and 12 and subtracting the multiplication result with 12 from the multiplication result with 1. The split multiplier values, 1 and 12, are obtained by mapping

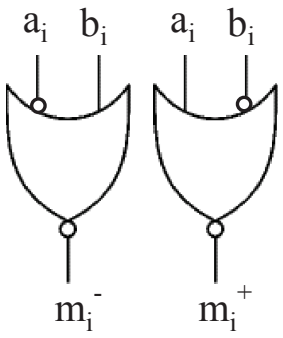


Figure 4.10: Conversion circuit to eliminate the '11' combination

only the 1 and -1 digit positions of the RBR multiplier with the bit 1 in the corresponding bit position of the two's complement multiplier. Figure 4.11 shows the partial product accumulation of this split multiplication scheme using the hybrid multiplication algorithm.

The proposed RBR multiplication algorithm is based on this split multiplication technique. Specifically, the separate partial product accumulation for each of the split multipliers is fused. The partial product arrays are simply folded together, and the following multiplication algorithm is obtained. Let p_{ij} represent the partial product of the multiplier digit a_i and the multiplicand digit b_j.

(1) Generate the partial product bits p_{ij} as explained below:

(a) If the RBR multiplier digit a_i is '1' and the RBR multiplicand digit b_j is also '1', then the partial product bit p_{ij}=1. Also, left shift the generated partial product by '1'.

(b) If the RBR multiplier digit a_i is '-1' and the RBR multiplicand digit b_j is also '-1', then the partial product bitp_{ij}=1. Also, left shift the generated partial product by '1'.

(c) If the RBR multiplier digit a_i is '0', then $p_{ij} = 1$ at all the bit positions for which

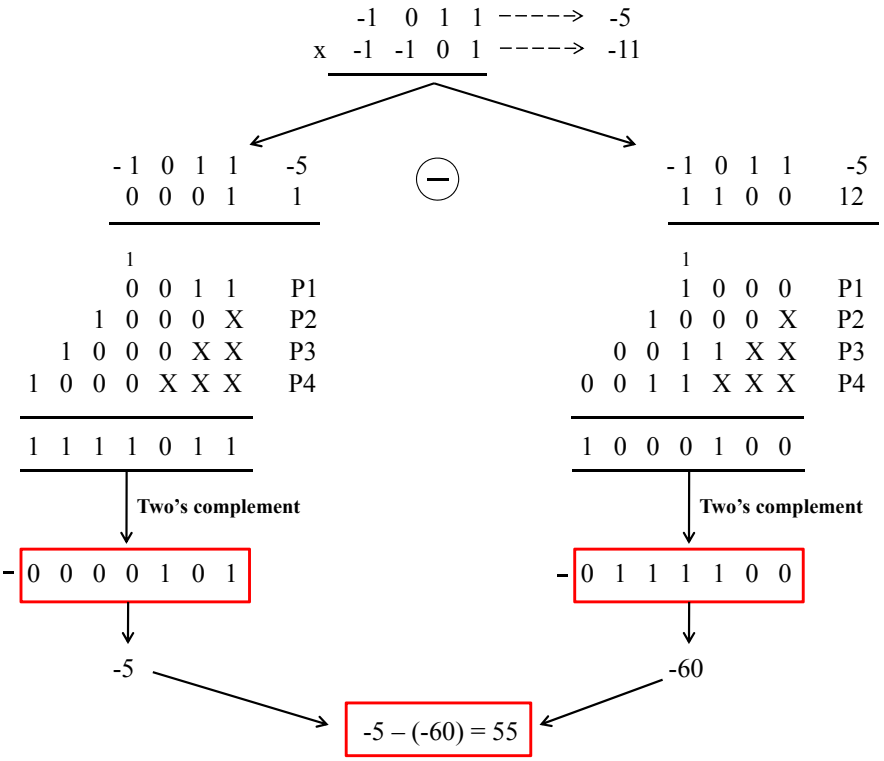


Figure 4.11: Illustration of RBR Multiplication using split multipliers

b_j is '1' or '-1'.

(2) Add the bit '1' at all the bit positions at which b_j is '1' or '-1'.

(3) Since the hybrid multiplication algorithm assumes a two's complement multiplier, the above steps would take care of all the cases where the split multipliers have the MSB bit as '0'. However, if there is, a '1' in the MSB position, it has to be interpreted as a positive weight in that bit position, unlike two's complement numbers, where a '1' in the MSB position implies negative numbers. To account for this case, an extra multiplier bit '0' is added to the RBR multiplier in the MSB position. The partial product for this bit is generated similar to step 1(c) above.

(4) The final partial product, generated with the additional MSB bit, is simply subtracted, similar to the hybrid multiplication scheme from the accumulated partial products for the other RBR multiplier digits.

An example of the proposed RBR multiplication scheme is depicted in Figure 4.12 below.

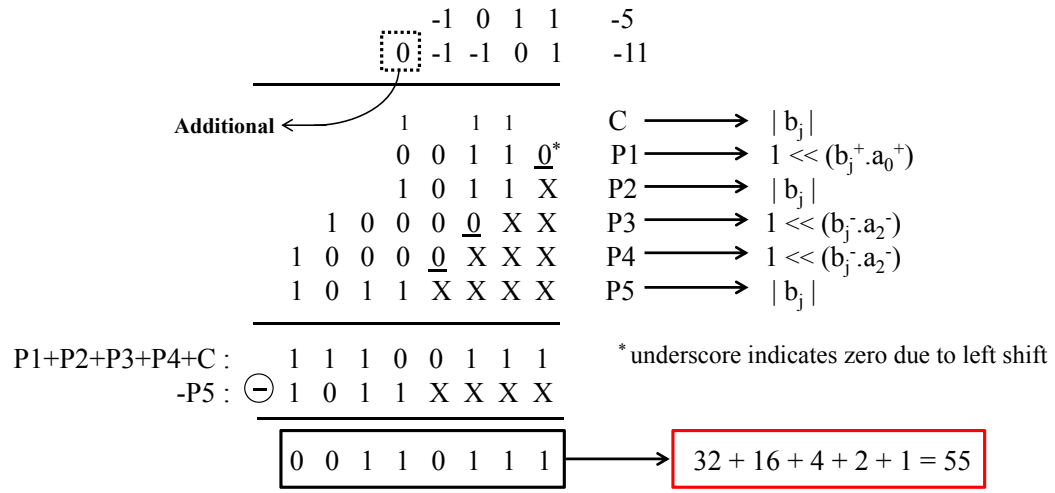


Figure 4.12: Example illustrating proposed RBR multiplication scheme

4.5.2 Partial product generation

The partial product generation logic, which is crucial for the RBR multiplication scheme, is described in this section. Detecting if the multiplier digit a_i is '1' or '-1' is equivalent to the modulus operation, $|a_i| = 1$, that is, $|a_i| = a_i^+ + a_i^-$. Similarly, detecting if the multiplicand digit b_j is '1' or '-1' amounts to the modulus operation, $|b_j| = b_j^+ + b_j^-$. Based on the partial product generated algorithm listed in section 4.5.1, the partial product bits p_{ij} can be generated as follows:

1 If $(|a_i| = 0)$, then $p_{ij} = |b_j|$.

2 If $(|a_i| = 1)$, then $p_{ij} = m_{ij-1}$, where m_{ij} is generated by the following multiplexing operation, $m_{ij} = b_j^+ . a_i^+ + b_j^- . \overline{a_i^+}$

This yields a partial product generation equation as follows, which is yet another multiplexing operation.

$$p_{ij} = |a_i| . m_{ij-1} + \overline{|a_i|} . |b_j|$$

4.5.3 RBR Multiplier Architecture

In terms of the high-level multiplier architecture, the structure for the partial product accumulation stage and subtraction of partial product in the last stage is exactly the same as the hybrid multiplication scheme, shown in Figure 4.9. It is also composed of (4,2) compressor blocks, PPM and PLPL blocks; only the partial product generation circuitry is different. Since the effective number of partial products for the hybrid case is n+2, for n-bit multiplication, an extra level of full adders or (3,2) compressor blocks is required, which increases the critical path delay of the RBR multiplier relative to the hybrid multiplier case. The implementation complexity of the partial product generation circuit is higher than that of the hybrid multiplier, and so the overall complexity of the RBR multiplier is higher, as will be illustrated in the next section.

4.6 Performance Comparisons

The delay, average energy, and EDP performance comparisons of the following five 16x16 bit multiplier architectures are depicted in Figures 4.13a, 4.13b, and 4.13c respectively. The architectures are (4,2) compressor-based Wallace Tree Multiplier (WTM_4:2), RBR tree multiplier based on the reference adder (RBR_Tree_ref), RBR tree multiplier based on the proposed adder (RBR_Tree_new), the proposed novel RBR multiplier architecture (MRBR_new), and the proposed novel hybrid multiplier (Hybrid_new). As seen from Figure 4.13a, the critical path delay of the two's complement system is significantly higher than the RBR and hybrid designs. In terms of average energy consumption, shown in Figure 4.13b, all the RBR multiplier designs, including the tree multipliers as well as MRBR_new, have higher average energy consumption compared to WTM_4:2. The hybrid multiplier design has a significantly lower average energy consumption compared to all the other architectures. In terms of EDP, both the proposed multiplier architectures have significantly lower EDP among all the designs, as evident from Figure 4.13c.

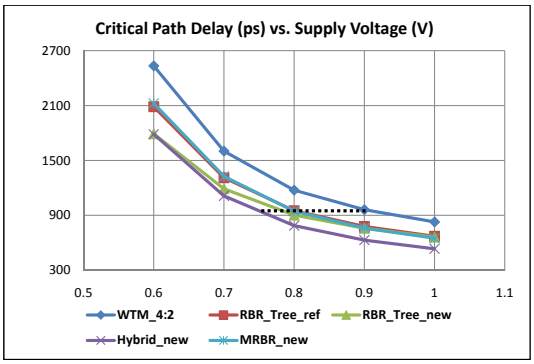
The RBR tree multipliers RBR_Tree_ref and RBR_Tree_new have overall similar performance trends. The RBR_Tree_new has a lower EDP compared to RBR_Tree_ref at

lower voltages. This is because the critical path delay at lower supply voltages is larger for RBR_Tree_ref due to slow signal propagation through the chain of transmission gates. The proposed RBR multiplier architecture has a significantly better EDP performance compared to the RBR tree multipliers. At nominal supply voltage, MRBR_new has a 25% lower EDP compared to RBR_Tree_reference, and a 19% lower EDP compared to RBR_Tree_new. With respect to the two's complement WTM_4:2, MRBR_new has an 8.1% lower EDP. The hybrid multiplier is the architecture with the lowest EDP. At nominal supply, Hybrid_new has a 45.5% better EDP performance in comparison to WTM_4:2 and 40% lower EDP compared to MRBR_new.

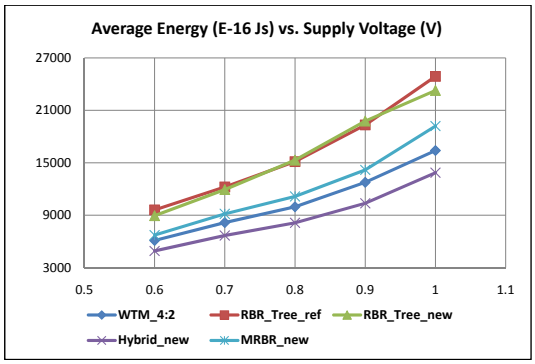
For an iso-throughput comparison, that is, comparing all the 16-bit multiplier designs for the same critical path delay, for a clock period of approximately 950 ps, both RBR tree multiplier designs have a higher EDP compared to WTM_4:2. However, the two new multiplication schemes, namely, MRBR_new and Hybrid_new, have a significantly lower EDP compared to WTM_4:2. Hybrid_new has an appreciably large 42% EDP performance improvement while, MRBR_new has a 14% EDP performance improvement.

4.7 Effect of Bit Precision

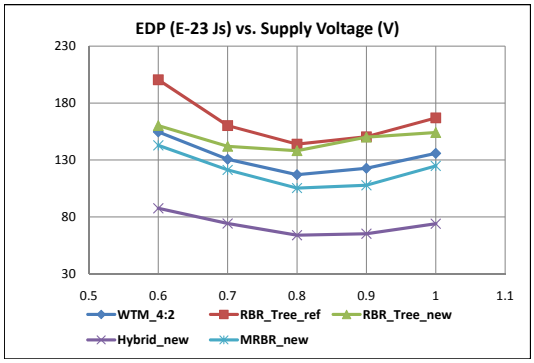
The performance trends for all multipliers at nominal supply and 0.8 V supply as a function of bit precision are shown in Figures 4.14 and 4.15, respectively. As seen from Figures 4.14a and 4.15a, the critical path delay of all the RBR and Hybrid multipliers increases from 8-bit to 12-bit precision, due to increase in the number of addition levels in the partial product accumulation stage. For 16-bit precision, since the number of addition levels is the same as for 12-bit, the critical path delay remains the same. In contrast, for the two's complement WTM_4:2, the critical path delay increases with increasing bit-precision. All the multipliers have increasing average energy trends with increase in bit-precision, as expected. In terms of EDP, the proposed multipliers have a better EDP performance with respect to all other multipliers at all the bit-precision nodes. As the supply voltage is reduced from 1 V to 0.8 V, the critical path delay for all the multipliers at all bit-precision nodes increases, while average energy shows quadratic reduction. Hence, all multipliers



(a) Critical Path Delay



(b) Average Energy Consumption

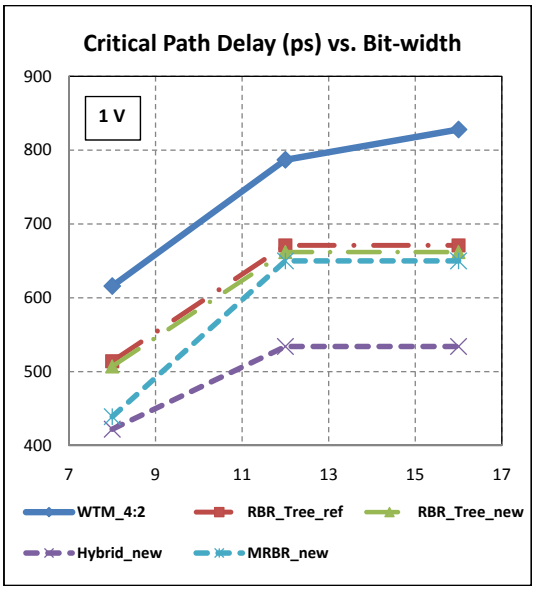


(c) EDP

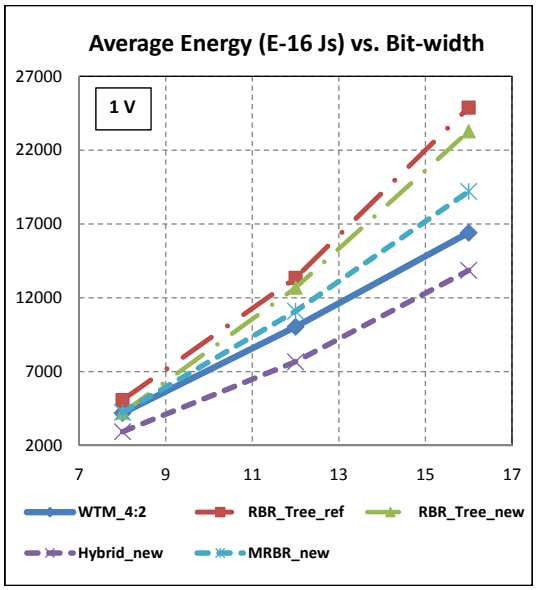
Figure 4.13: 16-bit Multiplier Performance Comparisons

exhibit reduced EDP at 0.8 V supply.

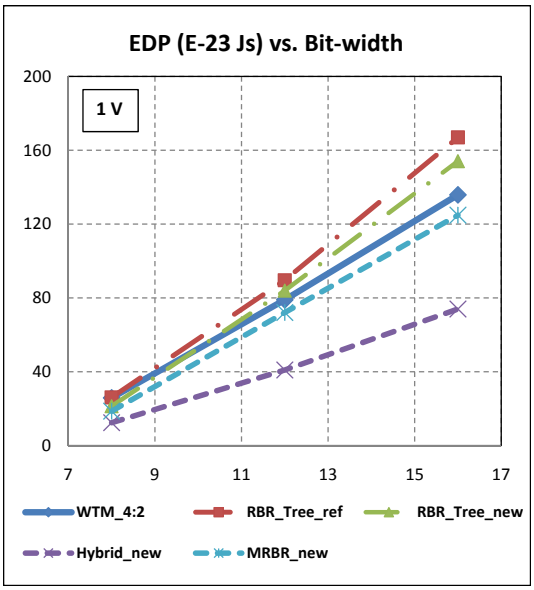
Figure 4.16 shows the EDP plots for all the multiplier architectures as a function of bit-width, for iso-throughput conditions. For 8-bit precision and iso-throughput, the EDP performance of both tree multipliers is not significantly different compared to WTM_4:2. The EDP performance improvement is 8.5% and 19% for RBR_Tree_ref and RBR_Tree_new repectively. The EDP performance of the tree multipliers degrades with increase in bit-precision, with comparable EDP at 12-bit precision, and significantly higher EDP at 16-bit precision.



(a) Critical Path Delay

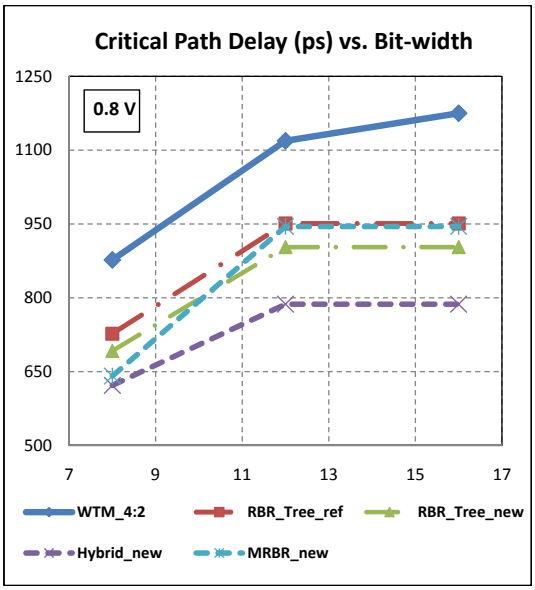


(b) Average Energy Consumption

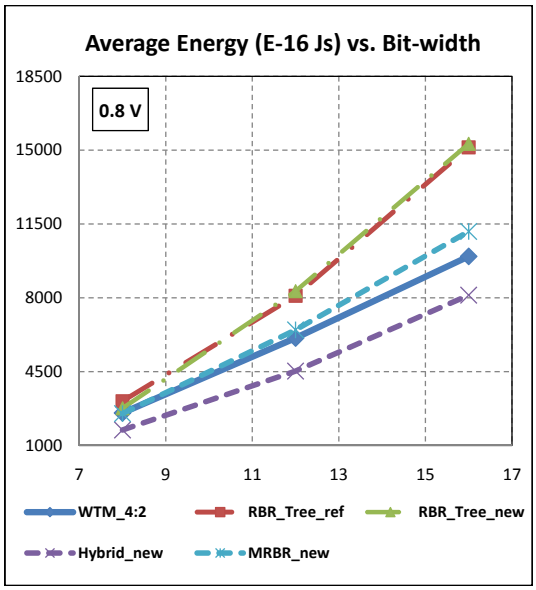


(c) EDP

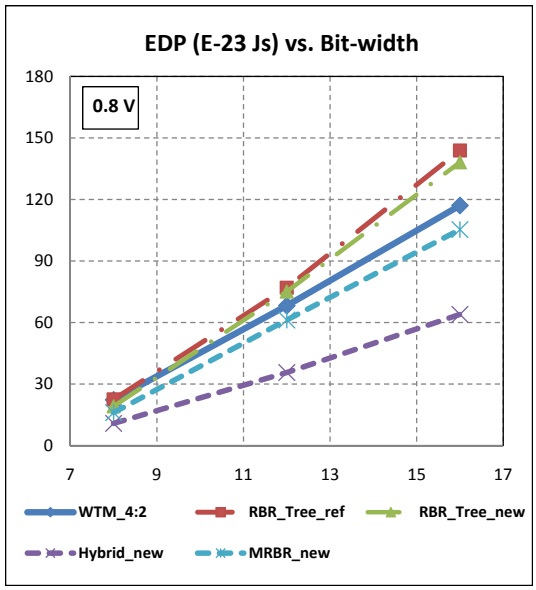
Figure 4.14: Multiplier Architectures: Effect of varying bit precision at nominal supply



(a) Critical Path Delay



(b) Average Energy Consumption



(c) EDP

Figure 4.15: Multiplier Architectures: Effect of varying bit precision at 0.8 V

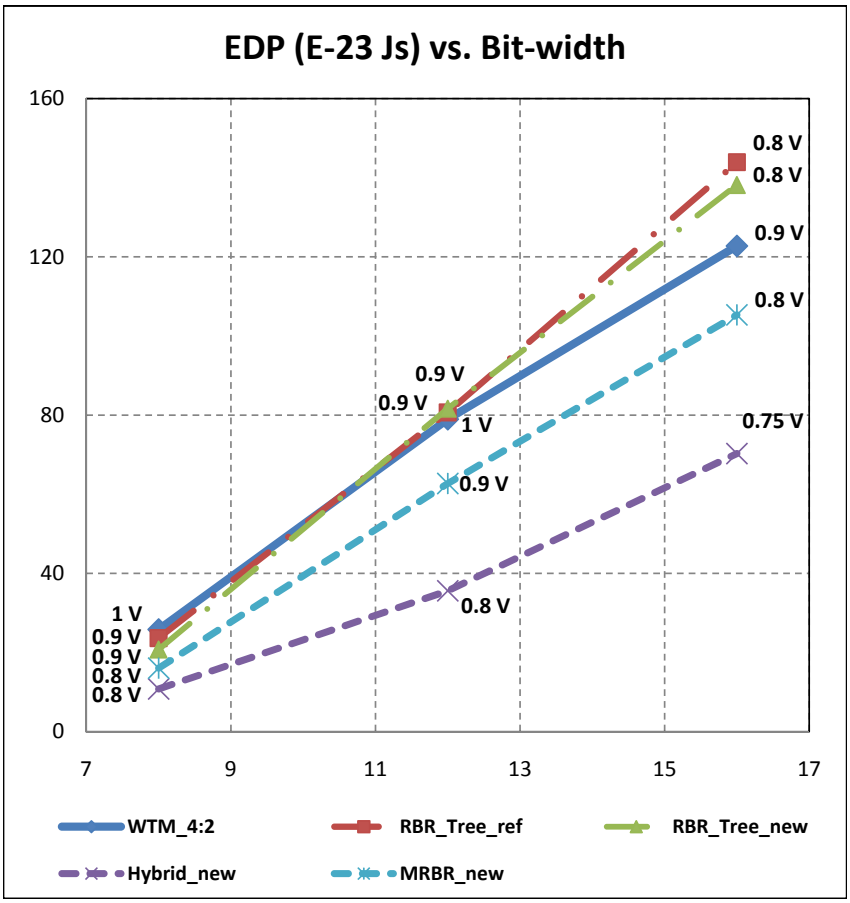


Figure 4.16: Multiplier Architectures: EDP as a function of bit precision at iso-throughput

The proposed designs, clearly have a better EDP performance over WTM_4:2 for all the bit precision nodes considered. For 8-bit precision and iso-throughput, MRBR_new has a 37% lower EDP, while Hybrid_new has a substantial 58% lower EDP. At 12-bit precision, the EDP performance gain marginally reduces to 55% and 20% respectively. At 16-bit precision, as demonstrated before in Figure 4.13c, Hybrid_new has a 42% better EDP, while MRBR_new has a 14% better EDP. This EDP trend with respect to varying bit precision is different from that seen for RBR adders, where the performance gain improves for greater bit precision. This is primarily because for 12-bit and 16-bit precision, the high-level multiplier structure for both proposed RBR and hybrid multiplier designs is precisely the same. The number of 4:2 compressor levels is exactly the same, and hence, the critical path delay is exactly the same. The only change is in the number of compressors at each level, with more units required for 16-bit multiplier than for 12-bit multiplier, leading to higher energy consumption, and hence, relatively higher EDP.

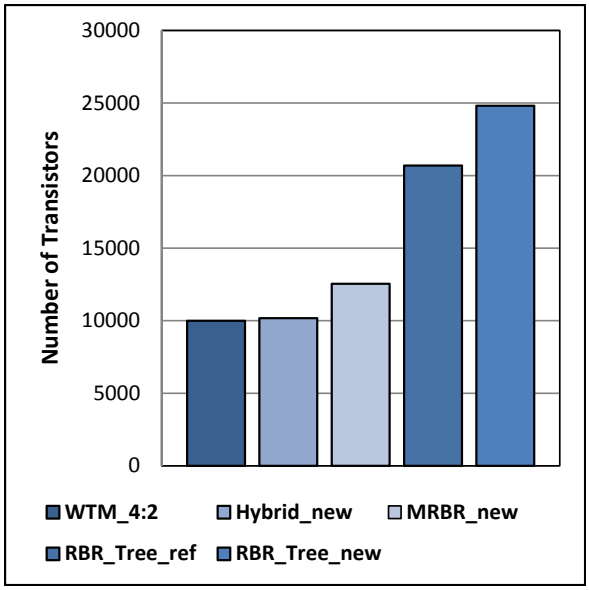


Figure 4.17: 16-bit Multipliers - Area Comparison

4.8 Area Comparison

Figure 4.17 compares the transistor count of all the 16-bit multiplier designs considered here. The two's complement multiplier has the lowest implementation area, as expected.

The RBR tree multipliers have the largest area overhead, as seen from the figure. This is because the partial product accumulation in the tree multipliers is composed of the gate-intensive RBR adder blocks. The proposed RBR multiplier design MRBR_new, has a relatively higher layout footprint of around 25% compared to WTM_4:2. In stark contrast, the hybrid multiplier design, Hybrid_new has a comparable implementation area compared to WTM_4:2.

Chapter 5

SYSTEM LEVEL DSP APPLICATIONS

5.1 RBR Overhead

As mentioned in Section 2.2.5, RBR systems have an associated conversion overhead. This is primarily because any processing kernel using RBR representation must communicate with other units, which are conventionally implemented in two's complement form. Figure 2.2 shows the block diagram of a RBR system. At the input end, the two's complement-to-RBR conversion is trivial and only requires explicit handling of the MSB bit. If the incoming two's complement number is negative, that is, its MSB bit is a '1', then the corresponding most significant digit in the equivalent RBR representation is a '-1'. At all other bit positions, for the case of the assumed encoding scheme, the '+' bit of each equivalent RBR digit is simply the corresponding two's complement bit, while the '-' bit is '0'.

The RBR-to-two's complement conversion block, however, presents significant overhead. It is typically implemented by converting the RBR result to two positive two's complement numbers R_+ and R_- obtained by mapping only the '1' digit and '-1' digit positions respectively, and then taking their difference, $(R_+ - R_-)$. In this work, a carry-select adder is used as the conversion module. Although this is an inevitable overhead for RBR systems, it can be shown that if the RBR computation module is of moderate to large size, this overhead is amortized over the computation operation, as illustrated by the following case studies. The delay, energy, and EDP performance of two's complement based DSP computation systems is evaluated and compared with that of RBR computation systems. In all cases, the overhead of converting from RBR to two's complement representatiion is taken into account.

5.2 Case Studies

5.2.1 Edge Detection

Edge detection is one of the most common image processing kernels used to extract features in images. It involves estimating the size of the transition (edge magnitude) and direction in which the intensity changes most rapidly. These are generally obtained from the partial derivatives of the image function and is implemented using two-dimensional convolution, with a window of known weights. The Sobel operator [13, 24] is considered here, whose $3x3$ kernels are given by $W_x = \begin{pmatrix} -1 & 0 & -1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$ and $W_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$. To compute the $(k,l)^{th}$ outputs, OUT_x (k, l)and OUT_y (k, l), we consider a window of $3x3$ pixels centered at (k, l).

Edge detection using Sobel operator is implemented as shown in Figures 5.1 and 5.2 for two's complement and RBR systems respectively. The input pixels can assume values from -256 to +255 and are represented by 9 bits. After each stage of addition, the precision is increased by one-bit. This is a pipelined implementation with registers after each adder. For the two's complement implementation (Sobel_2comp), which uses CSA adders, number of pipe stages is three, and the critical path delay is determined by the 12-bit adder delay. For the RBR implementation (Sobel_RBR), the proposed parallel RBR adder design, RBA_new is used, and the RBR-to-two's complement conversion is implemented using a 13-bit CSA adder in the last stage. A two-stage pipelined 13-bit CSA adder is employed, so as to clock the RBR system at a higher rate. The entire RBR computation operation is spread over five pipe stages.

Figures 5.3a, 5.3b, and 5.3c compare the critical path delay, energy and EDP performance of the two's complement and RBR Sobel edge detection systems, respectively. As evident from the trends, Sobel_RBR has no EDP performance gain over two's complement Sobel. While Sobel_RBR has a lower critical path delay compared to Sobel_2comp,

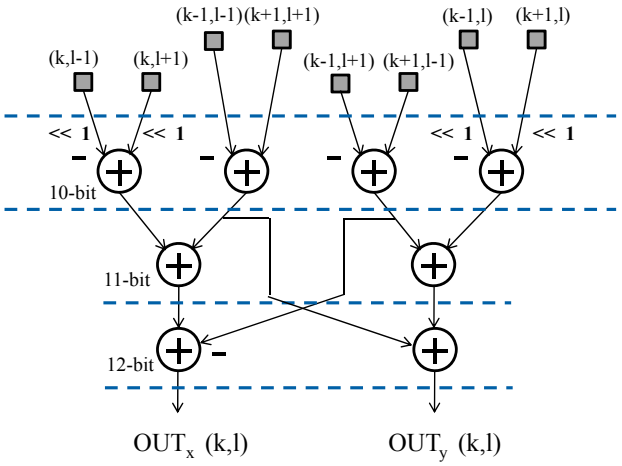


Figure 5.1: Data Flow graph of Edge Detection using Sobel operator - Two's complement system

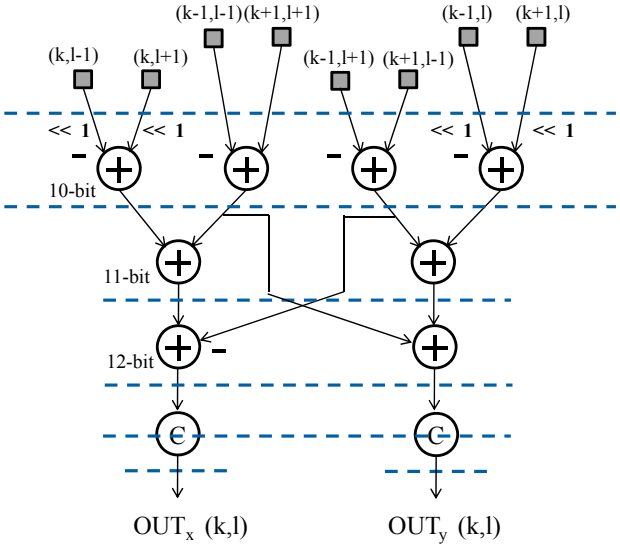
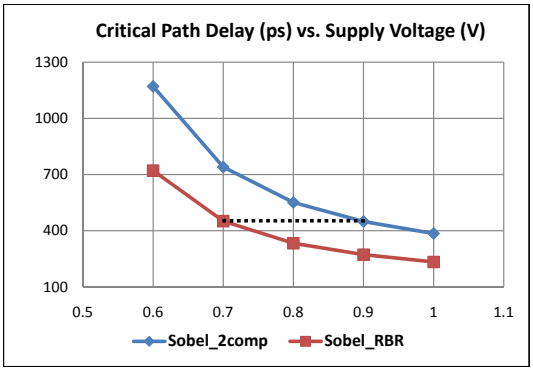
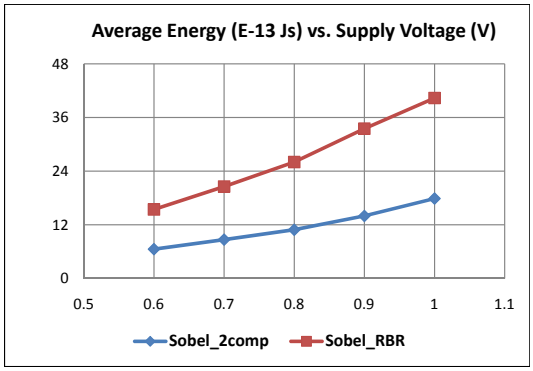


Figure 5.2: Data Flow graph of Edge Detection using Sobel operator - RBR system

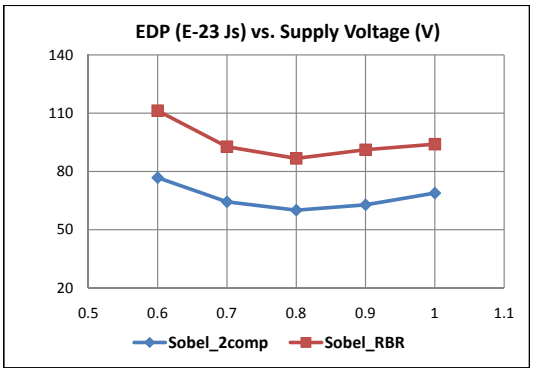
its average energy consumption over the entire computation operation spanning five pipe stages is significantly higher than Sobel_2comp. This leads to EDP degradation at all supply voltage nodes. Even for iso-throughput comparison, as depicted in Figure 5.3a, there is no improvement in EDP performance, with a significant 47% EDP degradation. Thus, it is clear that for low complexity systems, such as Sobel edge detection, the conversion overhead overpowers the EDP performance gain seen with the RBR adder itself.



(a) Critical Path Delay



(b) Average Energy Consumption

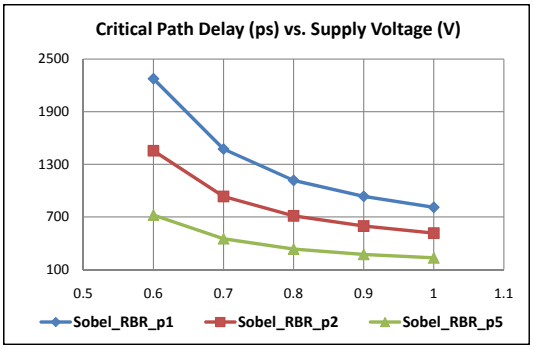


(c) EDP

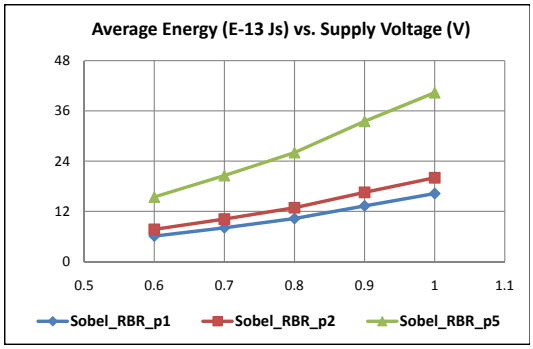
Figure 5.3: Edge Detection using Sobel operator: Performance Comparisons

Figures 5.4a, 5.4b, and 5.4c respectively compare the critical path delay, energy and EDP performance of the RBR Sobel edge detection system, for variable number of pipeline stages. Sobel_RBR_p1, Sobel_RBR_p2, and Sobel_RBR_p5, are respectively one-stage, two-stage, and five-stage pipelined implementations of the RBR Sobel system. As seen from the figures, with increase in the number of pipeline stages, the system throughput increases. However, due to increase in the number of registers, the average energy consumption increases as well. In terms of EDP, the five-stage pipeline design shows the lowest EDP, primarily due to the low critical path delay.

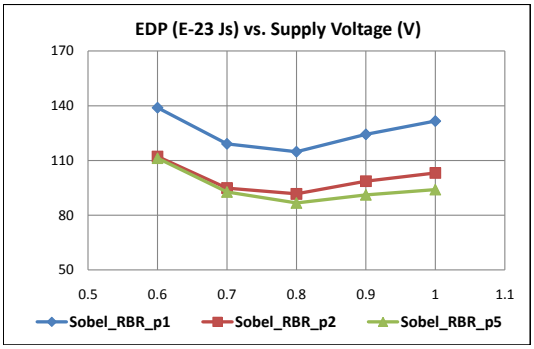
Figure 5.5 compares the total implementation area in terms of transistor count for both systems. Because of the significant conversion overhead, the area overhead for Sobel_RBR is significant as well with a 60% higher area compared to Sobel_2comp.



(a) Critical Path Delay



(b) Average Energy Consumption



(c) EDP

Figure 5.4: Edge Detection using Sobel operator: Effect of pipeling for RBR implementations

5.2.2 Discrete Cosine Transform

The Discrete Cosine Transform (DCT) is a common image processing kernel used in many image and video codecs. Due to its efficiency in spectral compaction of signals, DCT finds several applications in data compression. 2-D DCT can be separated into two 1-D DCTs and implemented using a single DCT and a transpose unit as illustrated in Figure 5.6. 1-D DCT transform of 8x8 DCT used in JPEG can be expressed as follows:

$$w_i = \frac{c_i}{2} \sum_{k=0}^{7} x_k \cos \frac{(2k+1)k\pi}{16}$$

$$c_i = \frac{1}{2} \; for \; i = 0$$

$$c_i = 1 \; for \; i = 1, ..., 7$$

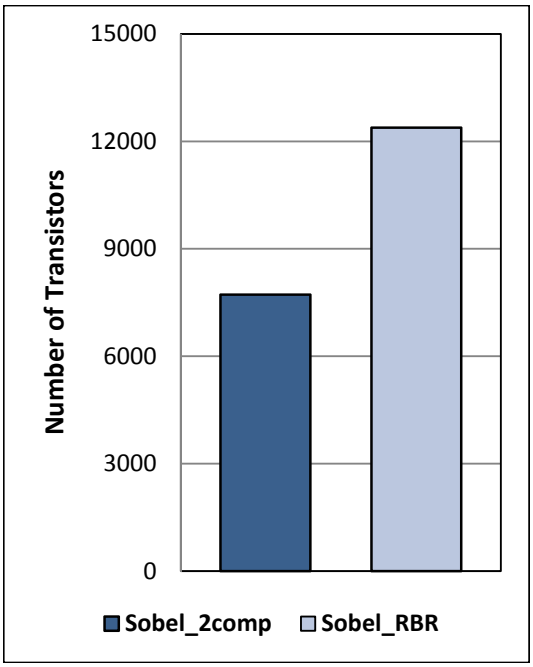


Figure 5.5: Edge Detection using Sobel operator - Area Comparison

The DCT architecture considered here is the popular add-shift architecture [37] and shown in Figures 5.7 and 5.8 for two's complement system and Figure 5.9 for RBR system. The input data-width is assumed to be 9 bits, and the internal precision is 12-bits (for the assumption that the same computation unit maybe used for row and column DCT). Pipeline registers are added for a high-throughput design. For the two's complement system, two cases are considered, a two-stage pipeline (DCT_pipe_2), with a critical path delay of three 12-bit adders, and a three-stage pipeline (DCT_pipe_2), with a critical path delay of two 12-bit adders. For the RBR system, a three-stage pipeline (DCT_RBR) is considered, with double the number of registers. In this case, the critical path is comprised of a single 12-digit RBR parallel adder, RBA_new, and a 12-bit CSA, which acts as the RBR-to-two's complement converter.

Figures 5.10a, 5.10b, and 5.10c compare the critical path delay, energy and EDP performance of the two's complement and RBR DCT systems respectively. While DCT_RBR has a lower critical path delay compared to both DCT_pipe_2 and DCT_pipe_3, its energy consumption is higher. In terms of EDP, DCT_RBR shows better performance over both

Figure 5.6: 2D DCT architecture using 1-D DCT and transpose unit

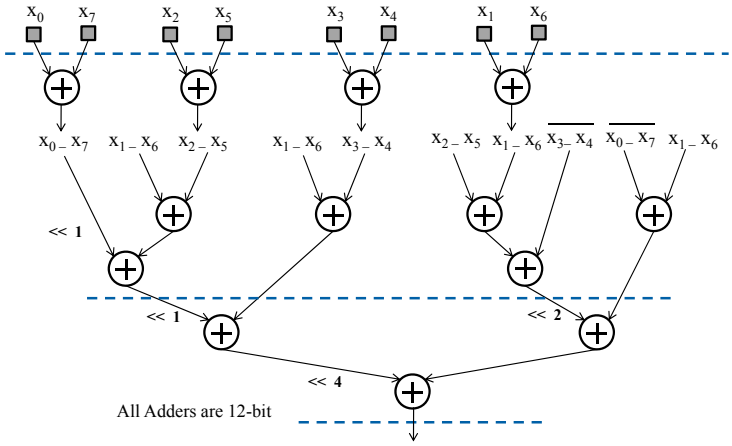


Figure 5.7: Data flow graph of the DCT kernel - Two's complement system (2-stage pipeline)

two's complement systems from 0.8V to 0.6 V, and comparable performance for the 0.9 V and 1 V nodes. For iso-throughput, as highlighted in Figure 5.10a, DCT_pipe_2 operates at 0.9V and DCT_RBR operates at 0.7V, and the RBR system has a marginal EDP performance gain of around 4.5%. In comparison to DCT_pipe_3 (0.9 V operation), for iso-throughput, DCT_RBR operates at 0.8 V, and the EDP performance gain is about 7%. The conversion overhead of RBR restricts the relative performance gain observed for RBR parallel adders.

Figure 5.11 shows the area comparison of DCT_pipe_2, DCT_pipe_3 and DCT_RBR. While both the two's complement pipelined systems have comparable area, DCT_RBR has a relatively higher layout footprint of around 33%.

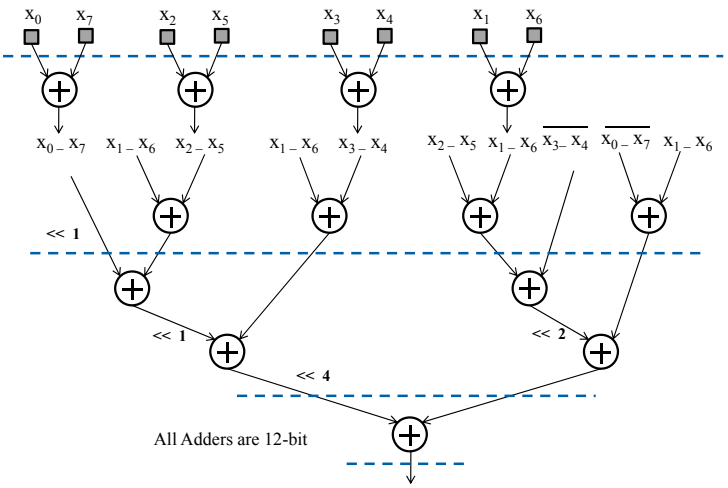


Figure 5.8: Data flow graph of the DCT kernel- Two's complement system (3-stage pipeline)

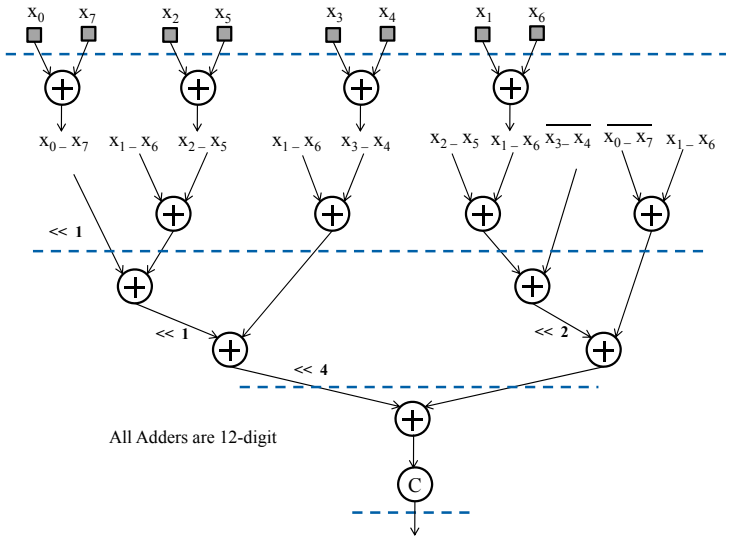
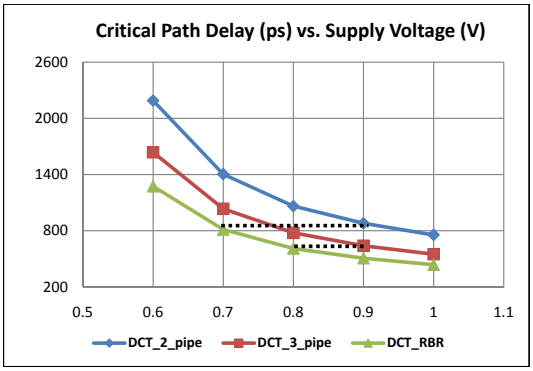


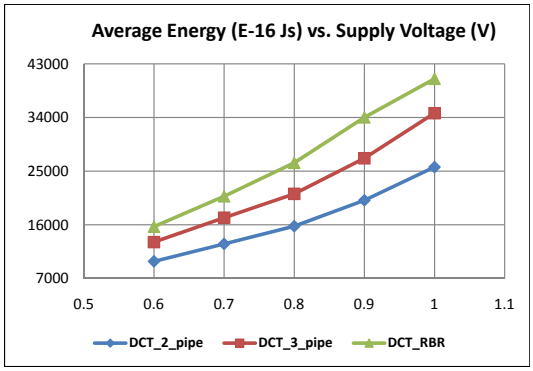
Figure 5.9: Data flow graph of the DCT kernel - RBR system

5.2.3 Complex Multiplication

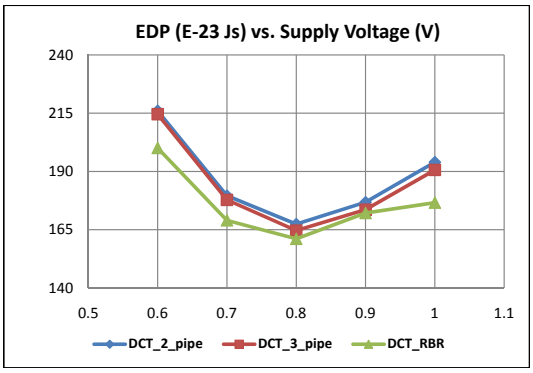
Complex number multiplication is used in many DSP operations such as the Fast Fourier Transform (FFT) butterfly, and other applications where the data streams and coefficients have complex number representations. Conventional implementation of complex number



(a) Critical Path Delay



(b) Average Energy Consumption



(c) EDP

Figure 5.10: Discrete Cosine Transform: Performance Comparisons

multiplication involves four real multiplications and two additions. Here, we consider the implementation in [52], where algebraic transformations reduce the computation complexity to three multiplications and five additions as shown below:

$$A = (A + jB)(C + jD) = R + jI$$

$$m_0 = (C - D)B$$

$$m_1 = (A - B)C$$

$$m_2 = (A + B)D$$

$$R = m_0 + m_1$$

$$I = m_0 + m_2$$

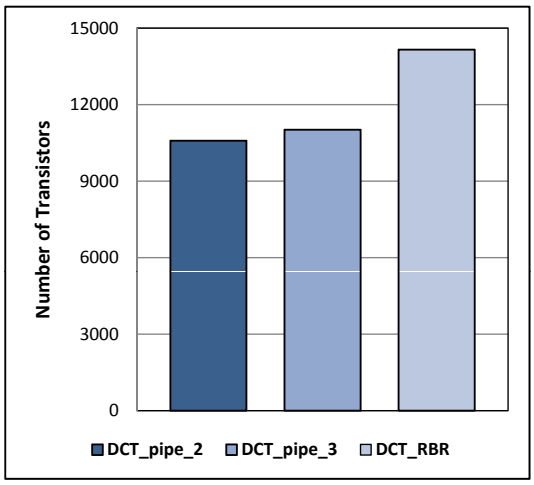
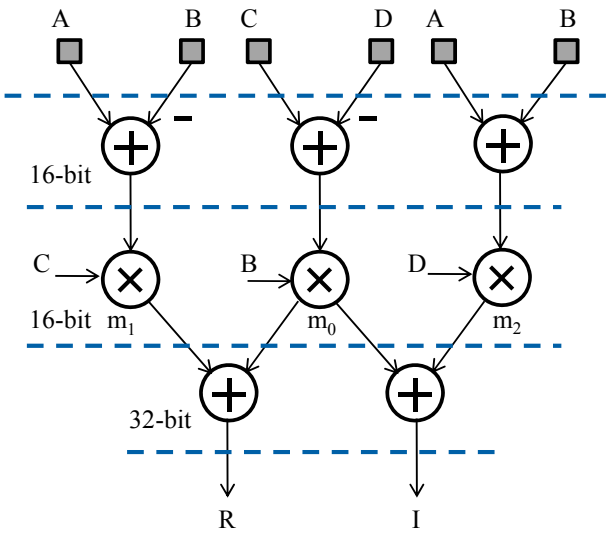


Figure 5.11: DCT - Area Comparison

Figures 5.12 and 5.13 depict the pipelined structure for two's complement and RBR/Hybrid systems respectively, to compute the product of two complex numbers, namely, (A + jB) (C+ jD). The inputs A, B, C, D are 16-bits, while the outputs R and I are 32 bits. Three implementations are considered, namely, two's complement, RBR and hybrid representation systems. All three systems use 16-bit adders and multipliers in addition to a final-stage 32-bit adder. A high-throughput three-stage pipelined implementation is considered for all systems.

The two's complement implementation CM_2comp uses 16-bit CSA adders and 16-bit 4:2 compressor-based Wallace tree multiplier (WTM_4:2) such that the critical path is the 16-bit multiplier delay. The RBR system CM_RBR uses the proposed designs, 16-digit parallel adder (RBA_new), and 16-digit proposed RBR multiplier MRBR_new, and double the number of registers. The hybrid system, CM_Hybrid, is composed of the proposed hybrid multiplier Hybrid_new, and RBA_new, and also consists of almost double the number of registers. In both the RBR and Hybrid systems, the RBR outputs obtained at the end of the computation operation, are converted to two's complement form using 32-bit CSA adders. This 32-bit CSA is incorporated comfortably in the third pipeline stage itself.

Figures 5.14a, 5.14b, and 5.14c depict the critical path delay, the average energy

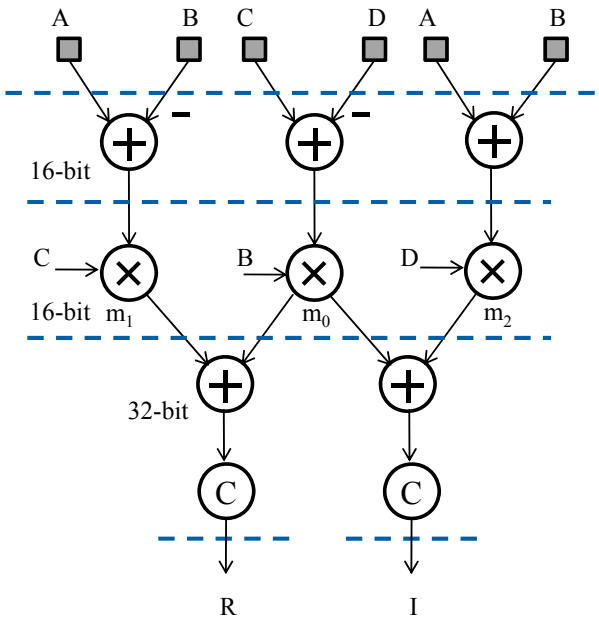


All Adders are 20-bit, All Multipliers are 16-bit

Figure 5.12: Dataflow graph of complex multiplication - Two's complement system

consumption and the EDP performance plots for the two's complement, hybrid, and RBR CM kernels respectively. The critical path of both CM_MRBR and CM_Hybrid is shorter than CM_2comp. In terms of average energy consumption, CM_2comp has a better performance compared to CM_MRBR, while CM_Hybrid has comparable performance. The shorter critical path and comparable average energy consumption of the CM_Hybrid lead to a lower EDP with respect to CM_2comp. The EDP performance of CM_MRBR is better with respect to CM_2comp from nominal to 0.8 V supply, below, which, the EDP performance becomes comparable. This is largely due to the partial product generation becoming slower and relatively more energy consuming due to slow transition times through the multiplexers at reduced supply voltages.

At nominal supply, CM_MRBR shows a 5% lower EDP while CM_Hybrid has a 35% EDP performance gain compared to CM_2comp. For iso-throughput, with a clock period of 850ps, CM_MRBR has around 12% EDP performance gain compared to CM_2comp, while CM_Hybrid has a high 41% EDP performance improvement, largely because the 16-bit multipliers constitute a major part of the computation operation.



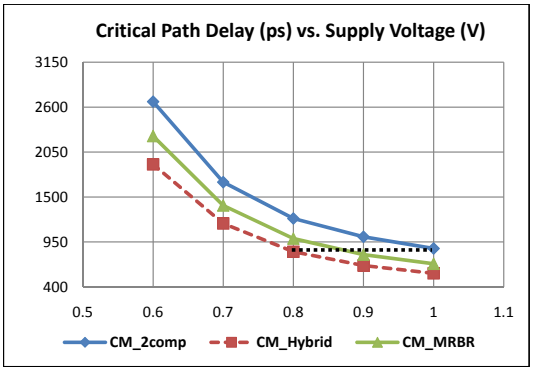
All Adders are 20-digit, All Multipliers are 16-digit

Figure 5.13: Dataflow graph of complex multiplication - RBR and Hybrid systems

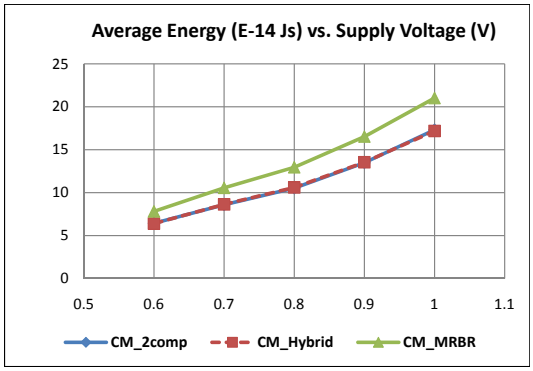
Figure 5.15 depicts the total implementation area of all three complex multiplication systems. It is evident from the figure that the implementation overhead for RBR representation reduces as the system complexity increases. The hybrid design CM_Hybrid has only 13% area overhead compared to CM_2comp with significantly high EDP performance gain. CM_MRBR has a 35% area overhead owing to the relatively large area of both the MRBR multiplier as well as additional registers.

5.2.4 Lifting-based Discrete Wavelet Transform (9, 7) Filter

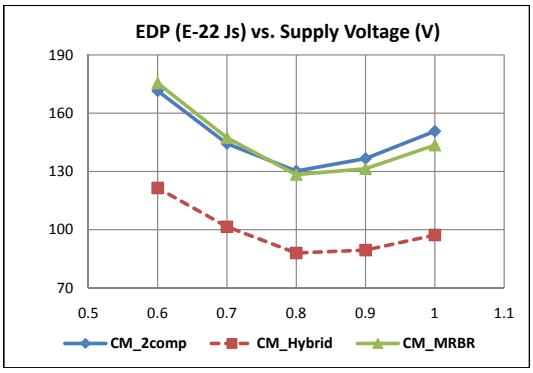
The discrete wavelet transform (DWT), which uses the concept of multi-resolution representation of signals, is popularly used in image compression. In particular, the lifting-based discrete wavelet transform (LDWT) [45] has been adopted in the JPEG2000 image compression standard. Some of the advantages of the lifting-based implementation include increased computation efficiency as compared to the convolution-based approach, limited memory resource requirement, and ease of architecture-level parallelism [2].



(a) Critical Path Delay



(b) Average Energy Consumption



(c) EDP

Figure 5.14: 16-bit Complex Multiplication: Performance Comparisons

In this section, we consider the simple (9, 7) filter used in JPEG2000. The most efficient factorization of the polyphase matrix for (9, 7) filter is as under [6]:

$$P(z) = \begin{pmatrix} 1 & a(1+z^{-}) \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ b(1+z) & 1 \end{pmatrix} \begin{pmatrix} 1 & c(1+z^{-}) \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ d(1+z) & 1 \end{pmatrix} \begin{pmatrix} K & 0 \\ 0 & 1/K \end{pmatrix}$$

$$where,\ a = -1.586134342, b = -0.0529801185, c = 0.882911076, d = 0.443506852,$$

$$K = 1.149604398$$

Figure 5.16 shows the direct-mapped form of the data-flow graph of the (9, 7) LDWT filter [27]. The input data is split into even and odd samples and the outputs are obtained with four pipelined lifting steps. The even terms of the output stream are the samples of the low pass subband and odd terms are the outputs of the high pass subband, for samples numbered from zero onwards. The hardware utilization of the data-path components can

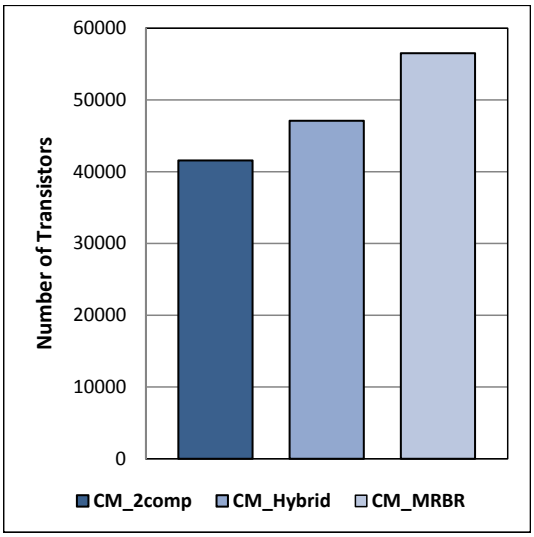


Figure 5.15: 16-bit Complex Multiplication - Area Comparison

be increased by folding the last two pipeline stages into the first two stages, as shown in the folded architecture proposed in [26]. This folded architecture is implemented using the two's complement, RBR and hybrid representations, as shown in figures 5.17 and 5.18 respectively. The incoming input data is assumed to be 8-bits, and the co-efficients a, b, c, d, K, and 1/K are represented in integer form using 10-bits. Based on the co-efficient values, an internal precision of 12-bits is chosen for the folded implementation. The two's complement implementation, LDWT_2comp, uses 12-bit CSA and 12x12 WTM_4:2. The RBR implementation, LDWT_MRBR, uses 12-digit parallel adder RBA_new, and 12x12 MRBR_new, while the hybrid system LDWT_Hybrid uses 12-digit RBA_new and 12x12 Hybrid_new.

Figures 5.19a, 5.19b, and 5.19c plot the critical path delay, average energy consumption and EDP performance for the two's complement, hybrid, and RBR LDWT kernels respectively. The critical path delay of LDWT_MRBR and LDWT_Hybrid are both significantly less compared to LDWT_2comp. In terms of average energy consumption, both LDWT_Hybrid and LDWT_MRBR have a relatively higher energy consumption compared to LDWT_2comp. The EDP performance of both LDWT_MRBR and LDWT_Hybrid is significantly better than LDWT_2comp for all the supply voltage nodes.

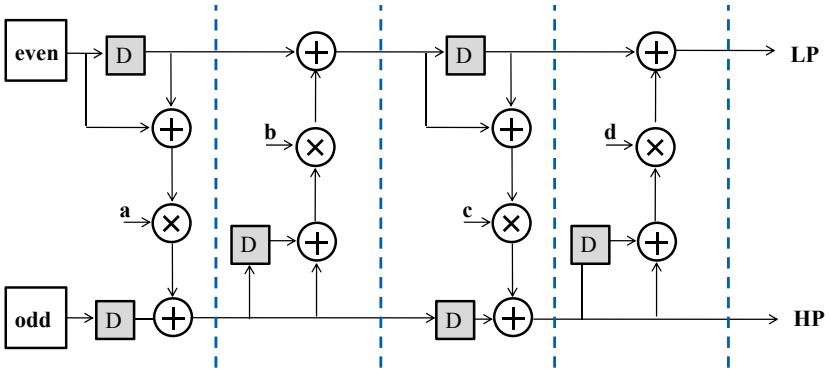


Figure 5.16: Direct form of LDWT (9, 7) filter Data flow graph

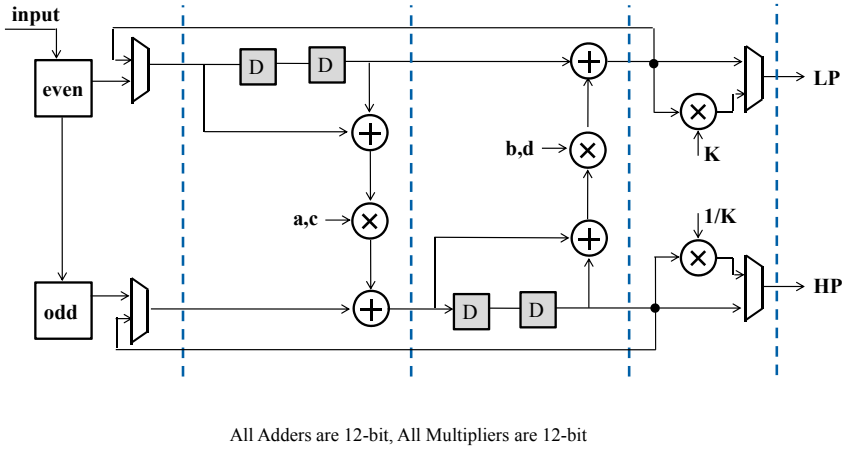


Figure 5.17: Data flow graph of LDWT (9, 7) filter (folded architecture) - Two's Complement system

For iso-throughput comparison, as depicted inFigure 5.19a, the EDP performance of both LDWT_MRBR and LDWT_Hybrid is appreciably better as compared to LDWT_2comp. For iso-throughput with a clock period of nearly 1.25 ns, LDWT_MRBR had an EDP performance gain of 19%, while LDWT_Hybrid shows an EDP performance gain of 38%. Thus, the conversion overhead is found to be insignificant for the folded LDWT (9, 7) architecture, for iso-throughput comparison, and considerable EDP performance improvement is obtained from both RBR and Hybrid systems.

Figure 5.20 depicts the total implementation area for all three LDWT (9, 7) fil-

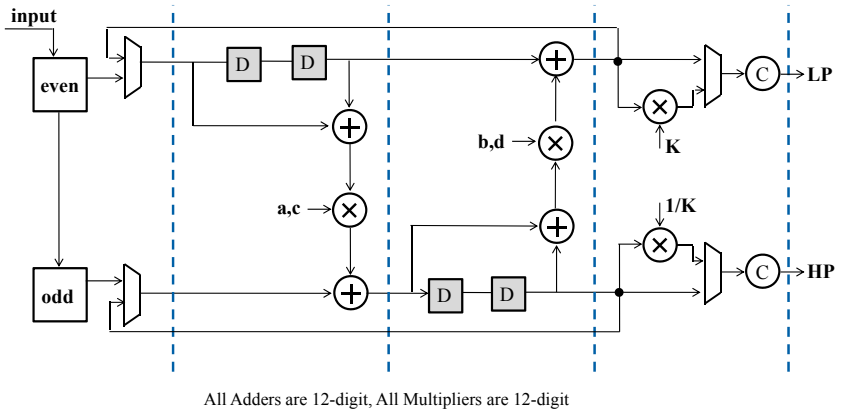


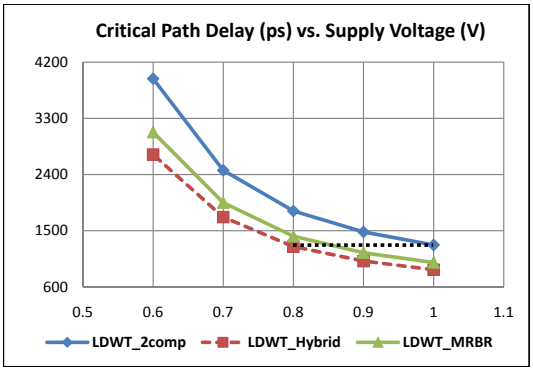
Figure 5.18: Data flow graph of LDWT (9, 7) filter (folded architecture) - RBR system

ter systems. The hybrid design LDWT_Hybrid has only 14% area overhead compared to LDWT_2comp with significantly high EDP performance gain, while LDWT_MRBR has a 37% area overhead over LDWT_2comp with considerable EDP performance gain.

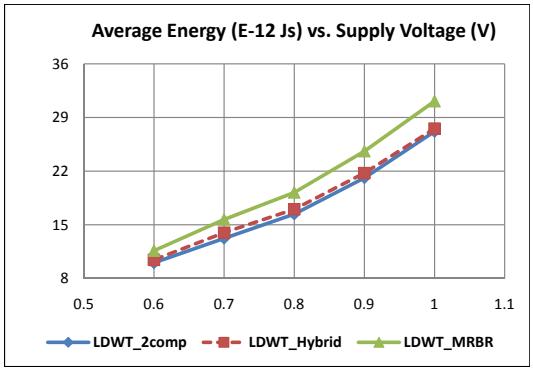
5.2.5 FIR Filter

Consider a pipelined 10-tap FIR filter implementation as shown in 5.21 and 5.22 for two's complement, and RBR based systems respectively. The two's complement system, FIR_2comp, uses 20-bit CSA and 16x16 WTM_4:2 modules. The RBR system, FIR_RBR, uses 20-digit RBA_new, and 16x16 MRBR_new. The hybrid system, FIR_Hybrid is comprised of 20-digit RBA_new and 16x16 Hybrid_new. The critical path delay of the non-pipelined implementation, is determined by $t_{critical} = t_{multiplier} + n * t_{adder}$; n = 10 in this case. A 4-stage balanced pipeline implementation is considered here for both systems. For the FIR_2comp system, even when registers are introduced after the multiplier followed by three adders, the critical path delay is determined by the 16-bit multiplier delay. In contrast, for FIR_RBR and FIR_Hybrid, registers are introduced after the multiplier, and a chain of four RBR adders and so, the critical path delay is four RBR adder delays. The last pipeline stage is comprised of one 20-digit RBR parallel adder, and a 20-bit CSA, which is the RBR-to-two's complement converter.

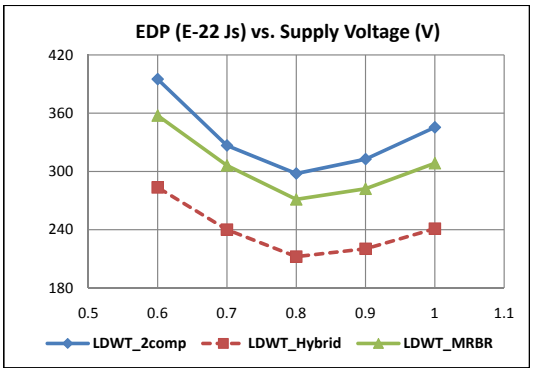
Figures 5.23a, 5.23b, and 5.23c plot the critical path delay, the average energy



(a) Critical Path Delay



(b) Average Energy Consumption



(c) EDP

Figure 5.19: LDWT (9, 7) filter: Performance Comparisons

consumption and the EDP for the two's complement, hybrid, and RBR FIR filter kernels respectively. The critical path of the FIR_MRBR and FIR_Hybrid designs is shorter than FIR_2comp. In terms of average energy consumption, FIR_2comp has a better performance compared to FIR_MRBR. However, the average energy consumption of FIR_Hybrid is lower than FIR_2comp, primarily because of the lower total average energy consumption of the ten multipliers. The shorter critical path and lower average energy consumption of the FIR_Hybrid leads to a relatively high EDP performance gain when compared to FIR_2comp, despite the conversion overhead. The EDP performance of FIR_MRBR is better with respect to FIR from nominal to 0.8 V supply, below, which, the EDP performance becomes comparable, again due to the effect of slow transition times through multiplexers at lower operating voltages. At nominal supply, FIR_MRBR shows an 8%

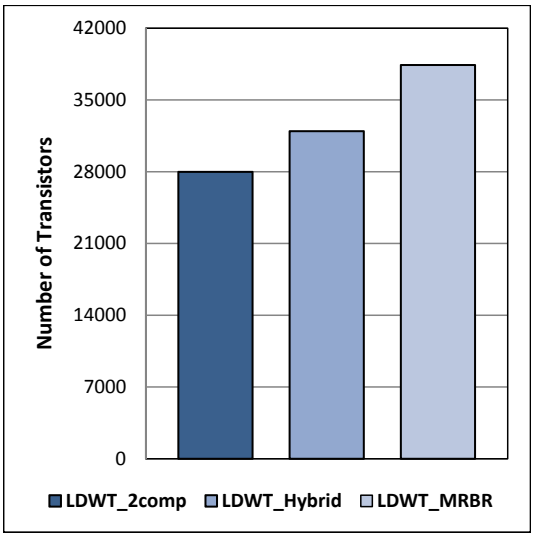


Figure 5.20: LDWT (9, 7) filter - Area Comparisons

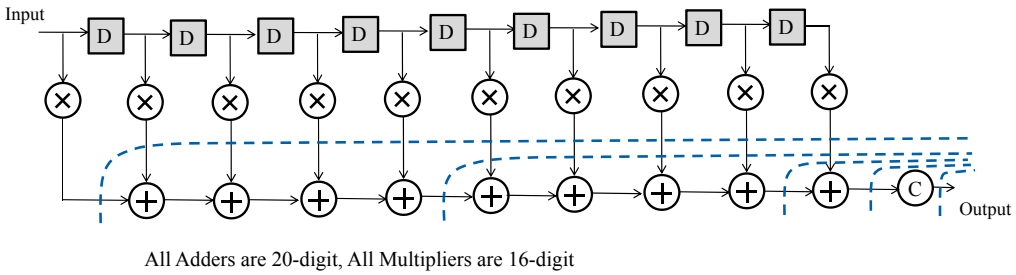


Figure 5.21: Data flow graph of FIR Filter - Two's Complement system

EDP performance gain while FIR_Hybrid has a 31.5% EDP performance gain compared to FIR_2comp. For iso-throughput, with a clock period of 1ns, FIR_MRBR has 11% EDP performance improvement compared to FIR_2comp, while FIR_Hybrid has a 32% EDP performance improvement. The relatively large EDP performance improvements observed for the FIR filter design shows that for high complexity systems, both the conversion overhead, as well as the register overhead of RBR systems are minor components in the overall EDP performance.

Figure 5.24 shows the area comparison plot for all three FIR filter implementations. FIR_Hybrid has an 8% larger implementation area compared to FIR_2comp, while FIR_MRBR has a 28% larger implementation area. However, both the hybrid and the RBR

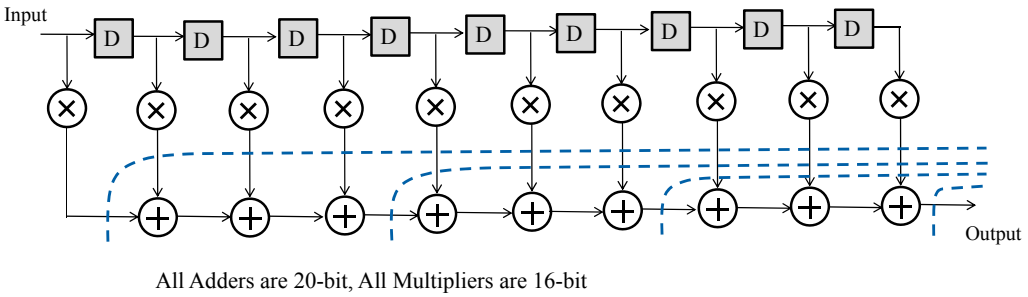
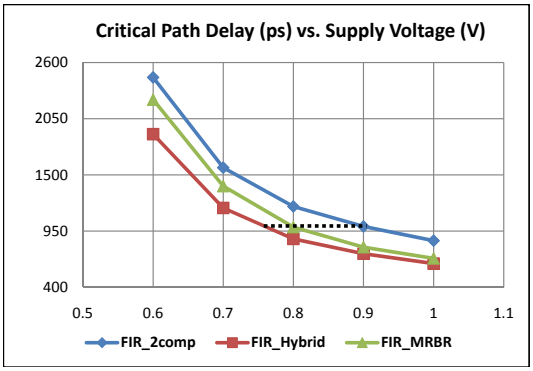


Figure 5.22: Data flow graph of FIR Filter - RBR system

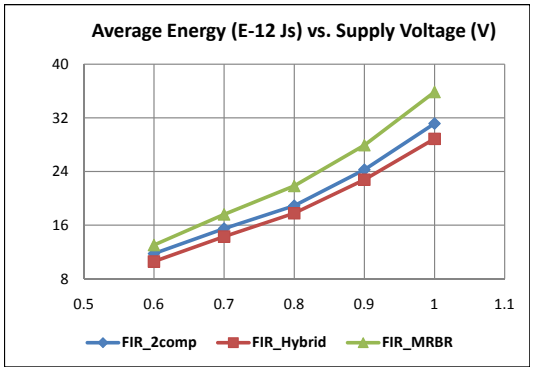
implementations have a significant EDP performance gain that makes the EDP-area tradeoff possible for high complexity deisgns.

5.2.6 Summary

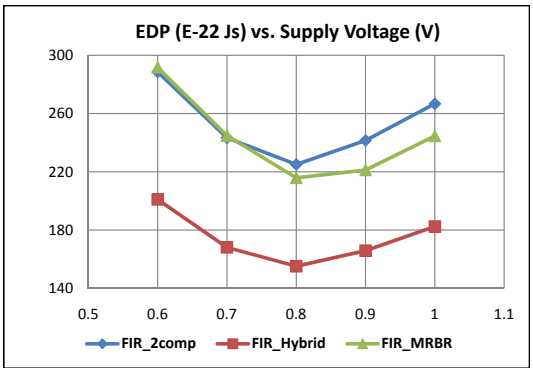
Figure 5.25 summarizes the performance of RBR systems for all the case studies considered. In case of low to medium complexity systems, the data-path is comprised only of adders, and the RBR representation yields very little performance gain. However, for high complexity systems (with nearly 30,000 transistors or higher) that are comprised of both multipliers and adders, the RBR system shows significant EDP performance gain for iso-throughput comparison. The hybrid system, especially shows an improved EDP performance gain.



(a) Critical Path Delay



(b) Average Energy Consumption



(c) EDP

Figure 5.23: FIR Filter: Performance Comparisons

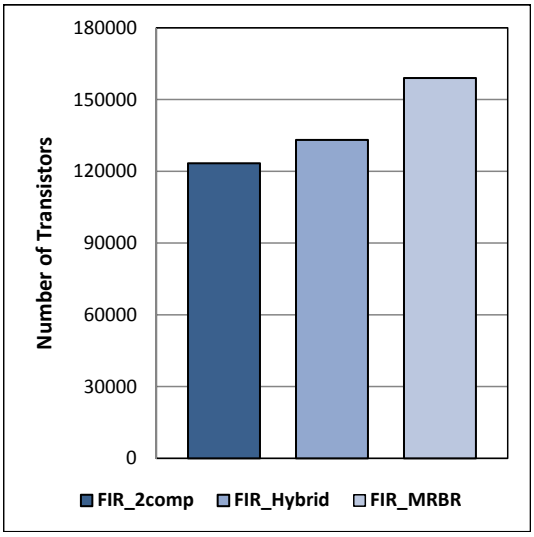


Figure 5.24: FIR Filter - Area Comparisons

DSP kernel	Data-path	Total Transistor Count*	Complexity	Iso-throughput EDP Performance
Edge Detection	Adders only	7718	Low	No gain
Discrete Cosine Transform	Adders only	10584	Low to Medium	Comparable (5% Gain)
LDWT (9, 7) Filter	Adders & Multipliers	27984	High	38% Gain - Hybrid 19% Gain - RBR
Complex Multiplier	Adders & Multipliers	41576	High	41% Gain - Hybrid 12% Gain - RBR
10-tap FIR Filter	Adders & Multipliers	123338	High	32% Gain - Hybrid 11% Gain - RBR

* Indicates the total implementation area for conventional two's complement system

Figure 5.25: Summary of RBR system performance with varying system complexity

Chapter 6

Conclusion

This thesis investigated the applicability of radix-2 RBR arithmetic for data-path design of high-throughput, energy-efficient DSP systems. A case is presented here for voltage-scaled RBR systems that have reduced energy consumption for iso-throughput compared to conventional two's complement systems. It is shown that for applications where energy consumption is a primary design constraint and area constraints are relaxed, RBR systems provide a viable alternative that should be seriously considered.

6.1 Summary

A new design for a RBR parallel adder with good EDP performance is proposed. It is shown that the proposed RBR parallel adder has lower critical path delay and better EDP performance compared to two's complement CSA and CLA adders. The EDP performance gain of RBR parallel adders increases with increasing bit-precision, owing to the critical path delay being independent of bit-width.

Novel multiplier architectures that process operands in RBR representation are proposed, and their performance evaluated. A new RBR multiplier design that multiplies two RBR operands is proposed and its performance compared with existing RBR tree multiplier architectures. The new design has 22% lower EDP for 16-bit precision, with a significant implementation area reduction of around 40%. In comparison to 16-bit two's complement multiplier, the new architecture is shown to have an 8.1% EDP performance gain at nominal supply voltage. For iso-throughput comparison, the RBR multiplier is found to have a 14% EDP performance gain, but with a 25% higher area compared to a two's complement multiplier.

A novel hybrid multiplier architecture where the multiplicand is in RBR form and the second operand is in two's complement form is also proposed. The hybrid design has a superior EDP performance among all the investigated multiplier architectures, specifi-

cally the conventional two's complement multiplier. The hybrid design shows a 45% EDP performance gain over the two's complement multiplier at nominal supply voltage, and a 40% EDP performance gain over the proposed RBR multiplier for 16-bit precision. For iso-throughput, the hybrid multiplier has a substantial 42% EDP performance gain over the two's complement multiplier. The hybrid design is also an area-efficient implementation, with comparable area with respect to the two's complement multiplier.

The energy-efficacy of using RBR arithmetic modules at the system-level is shown through five popular signal processing kernels. These include kernels whose data-path is comprised only of adders such as Discrete Cosine Transform, and edge detection using Sobel operator, and kernels with both adders and multipliers in the data-path such as complex multiplication, lifting-based Discrete Wavelet Transform (9, 7) filter, and FIR filter. The conversion overhead of RBR systems is taken into account for all these computation modules. It is shown that for low complexity computation systems, such as Sobel edge detection, the conversion overhead of RBR systems presents a significant overhead, and consequently the EDP performance deteriorates. For low to medium complexity systems such as the DCT add-shift architecture, the EDP performance gain of RBR systems is found to be very small.

As the system complexity increases, the conversion overhead of RBR modules gets amortized. For all three high complxity systems considered here, including complex multiplication, lifting-based DWT (9, 7) filter, and FIR filter systems, the RBR and hybrid systems depict significant EDP performance gain compared to conventional two's complement implementations. In case of the complex multiplier, for iso-throughput, both the RBR and hybrid designs exhibit 12% and 41% EDP performance improvement with respect to the two's complement system, respectively. The RBR and hybrid implementations of the folded lifting-based DWT (9, 7) filter architecture also have a significant ED performance gain over the two's complement implementation. The RBR implementation is shown to have a relative EDP performance gain of 18% for iso-throughput, while the relative performance gain of the hybrid system is higher, with 38% lower EDP. For the FIR filter system,

for iso-throughput, the RBR system demonstrates a 11% performance gain, while the hybrid design shows a 31% performance gain.

6.2 Future Work

RBR systems have very high-throughput compared to conventional systems, and so RBR representation could be suitable for near-threshold computing. Near threshold operation gives 10x more energy efficiency than nominal voltage operation, at the expense of 10x reduction in frequency. The reduction in frequency can be partly compensated by using RBR units which have significantly lower critical path delay. This presents an open-ended issue in low power computing.

Floating point operations are quite energy-intensive. Since RBR systems are found to perform better for higher bit-width systems, the use of RBR representation for floating point units should be researched. Floating point operations typically require 24-bit mantissa multiplication for single precision, which increases to 54-bit multiplication for double precision. The energy-efficient RBR and hybrid multiplier structures proposed here may be used for internal RBR representation in floating point kernels. However, comparison and rounding operations in RBR representation are non-trivial, and present a design challenge that will have to be explored.

REFERENCES

[1] Z. Abid and R. Mudassir. Signed Booth multiplier using redundant-binary-number system. In *International Conference on Electrical and Computer Engineering*, pages 145 –148, Dec. 2006.

[2] T. Acharya and P.-S. Tsai. *JPEG2000 Standard for Image Compression: Concepts, Algorithms and VLSI Architectures*. Wiley-Interscience, 2004.

[3] A. Avizienis. Signed-digit number representations for fast parallel arithmetic. *IRE Transactions on Electronic Computers*, EC-10(3):389 –400, Sept. 1961.

[4] C. R. Baugh and B. A. Wooley. A two's complement parallel array multiplication algorithm. *IEEE Transactions on Computers*, C-22:1045–1047, 1973.

[5] C. Y. Chow and J. E. Robertson. Logical design of a redundant binary adder. In *Proceedings of 4th Symposium on Computer arithmetic*, pages 109 – 115, Oct 1978.

[6] I. Daubechies and W. Sweldens. Factoring wavelet transforms into lifting steps. *J. Fourier Anal. Appl*, 4:247–269, 1998.

[7] J. Duprat, Y. Herreros, and J.-M. Muller. Some results about on-line computation of functions. In *Proceedings of 9th Symposium on Computer Arithmetic*, pages 112 –118, Sep 1989.

[8] H. Edamatsu, T. Taniguchi, T. Nishiyama, and S. Kuninobu. A 33 Mflops floating point processor using redundant binary representation. In *IEEE International Solid-State Circuits Conference*, page 152, Feb. 1988.

[9] M. D. Ercegovac and T. Lang. Fast multiplication without carry-propagate addition. *IEEE Transactions on Computers*, 39:1385–1390, 1990.

[10] M. I. Ferguson and M. D. Ercegovac. A multiplier with redundant operands. In *Proc. 33rd Asilomar Conf. Signals Systems and Computers*, pages 1322–1326, 1999.

[11] A. F. Glew. Processor with architecture for improved pipelining of arithmetic instructions by forwarding redundant intermediate data forms. In *IEEE Symposium on Computer Arithmetic*, April 1997.

[12] A. F. González and P. Mazumder. Redundant arithmetic, algorithms and implementations. *Integr. VLSI J.*, 30:13–53, November 2000.

[13] R. C. Gonzalez and R. E. Woods. *Digital Image Processing (3rd Edition)*. Prentice-Hall, Inc, Upper Saddle River, NJ, USA, 2006.

[14] G. Goto, T. Sato, M. Nakajima, and T. Sukemura. A 54x54-b regularly structured tree multiplier. *IEEE Journal of Solid-state Circuits*, 27:600–605, 1992.

[15] A. Guyot, Y. Herreros, and J.-M. Muller. Janus, an on-line multiplier/divider for manipulating large numbers. In *Proceedings of 9th Symposium on Computer Arithmetic*, pages 106 –111, Sep 1989.

[16] Y. Harata, Y. Nakamura, H. Nagase, M. Takigawa, and N. Takagi. A high-speed multiplier using a redundant binary adder tree. *IEEE Journal of Solid-State Circuits*, 22(1):28 – 34, Feb 1987.

[17] Y. He and C.-H. Chang. A new redundant binary booth encoding for fast 2^n-bit multiplier design. *IEEE Transactions on Circuits and Systems*, 56(6):1192 –1201, June 2009.

[18] http://ptm.asu.edu/.

[19] http://www.cse.psu.edu/research/mdl/mji/mjicourses/575/cse575-3SDadders.ppt/view.

[20] X. Huang, W. Liu, and B. Wei. A high-performance CMOS redundant binary multiplication-and-accumulation (mac) unit. *IEEE Trans. Circ. and Syst.-I: Fundamental Theory and Applicat*, 41:145–148, Jan 1984.

[21] M. Irwin and R. Owens. Design issues in digit serial signal processors. In *Circuits and Systems, 1989., IEEE International Symposium on*, pages 441 –444 vol.1, May 1989.

[22] M. J. Irwin and R. M. Owens. Digit-pipelined arnthmetic as illustrated by the paste-up system: A tutorial. *IEEE Computer*, 20:61–73, April 1987.

[23] I. Koren. *Computer Arithmetic Algorithms*. Prentice-Hall, Inc, Englewood Cliffs, N.J, 1993.

[24] S. Kung. *VLSI Array Processors*. Prentice-Hall, Inc, 1988.

[25] S. Kuninobu, T. Nishiyama, H. Edamatsu, T. Taniguchi, and N. Takagi. Design of high speed mos multiplier and divider using redundant binary representation. *Proceedings of the 8th IEEE Symposium on Computer Arithmetic*, pages 80–86, may 1987.

[26] C.-J. L., K. Chen, H. Chen, and L. Chen. Lifting based discrete wavelet transform architecture for JPEG2000. In *IEEE International Symposium on Circuits and Systems*, pages 445–448, 2001.

[27] C. Liu, Y. Shiau, and J. Jou. Design and implementation of a progressive image coding chip based on the lifted wavelet transform. *11th VLSI Design/CAD Symposium*, pages 49–52, 2000.

[28] C. N. Lyu and D. Matula. Redundant binary Booth recoding. In *Proceedings of the 12th Symposium on Computer Arithmetic*, pages 50 –57, Jul 1995.

[29] H. Makino, Y. Nakase, H. Suzuki, H. Morinaka, H. Shinohara, and K. Mashiko. An 8.8-ns 5454-bit multiplier with high speed redundant binary architecture. *IEEE Journal of Solid-state Circuits*, 31:773–783, 1996.

[30] M. Mehta, V. Parmar, and E. Swartzlander. High-speed multiplier design using multi-input counter and compressor circuits. In *IEEE Symposium on Computer Arithmetic*, pages 43–50, 1991.

[31] M. Nagamatsu, S. Tanaka, J. Mori, K. Hirano, T. Noguchi, and K. Hatanaka. A 15-ns 32x32-b CMOS multiplier with an improved parallel structure. *, IEEE Journal of Solid-State Circuits*, 25(2):494 –497, Apr 1990.

[32] N. Ohkubo, M. Suzuki, T. Shinbo, T. Yamanaka, A. Shimizu, K. Sasaki, and Y. Nakagome. A 4.4 ns CMOS 54x54-b multiplier using pass-transistor multiplexer. *IEEE Journal of Solid-State Circuits*, 30(3):251 –257, Mar 1995.

[33] B. Parhami. Generalized signed-digit number systems: A unifying framework for redundant number representations. *IEEE Trans. on Computers*, 39:89–98, 1990.

[34] B. Parhami. On the implementation of arithmetic support functions for generalized signed-digit number systems. *IEEE Trans. on Computers*, 42:379–384, March 1993.

[35] B. Parhami. *Computer Arithmetic Algorithms And Hardware Designs*. Oxford University Press, 2000.

[36] K. Parhi. *VLSI Digital Signal Processing Systems: Design and Implementation*. Wiley, NY, 1999.

[37] J. Park, J. H. Choi, and K. Roy. Dynamic bit-width adaptation in dct: An approach to trade off image quality and computation energy. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 18(5):787 –793, May 2010.

[38] D. Phatak and I. Koren. Hybrid signed-digit number systems: a unified framework for redundant number representations with bounded carry propagation chains. *IEEE Transactions on Computers*, 43(8):880 –891, Aug 1994.

[39] T. N. Rajashekhara and O. Kal. Fast multiplier design using redundant signed-digit numbers. *International Journal of Electronics*, 69(3):359–368, 1990.

[40] W. C. S. A suggestion for a fast multiplier. *IEEE Transactions on Electronic Computers*, pages 14–17, Feb 1964.

[41] M. R. Santoro and M. A. Horowitz. Spim: a pipelined 64*64-bit iterative multiplier. *IEEE Journal of Solid-state Circuits*, 24:487–493, 1989.

[42] K.-W. Shin, B.-S. Song, and K. Bacrania. A 200-MHz complex number multiplier using redundant binary arithmetic. *IEEE Journal of Solid-state Circuits*, 33:904–909, 1998.

[43] A. Skaf and A. Guyot. VLSI design of on-line add/multiply algorithms. In *International Conference on Computer Design*, pages 264–267, 1993.

[44] H. R. Srinivas and K. K. Parhi. High-speed vlsi arithmetic processor architectures using hybrid number representation. *J. VLSI Signal Process. Syst.*, 4:177–198, May 1992.

[45] W. Sweldens. The lifting scheme: A custom-design construction of biorthogonal wavelets. *Applied and Computational Harmonic Analysis*, 3:186–200, 1996.

[46] H. Y. T. Nakanishi and H. Yoshimura. CMOS radix-2 signed-digit adder by binary code representation. *The Transactions of the IECE of Japan*, E69(3):261–263, April 1986.

[47] N. Takagi, H. Yasuura, and S. Yajima. High-speed VLSI multiplication algorithm with a redundant binary addition tree. *IEEE Trans. on Comput.*, 34:789–796, September 1985.

[48] M. Tonomura. High-speed digital circuit of Discrete Cosine Transform. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E78:957–962, 1995.

[49] K. Trivedi and M. Ercegovac. On-line algorithms for division and multiplication. *IEEE Transactions on Computers*, C-26(7):681 –687, july 1977.

[50] J. Vuillemin. A very fast multiplication algorithm for V.L.S.I. implementation. Technical Report RR-0183, INRIA, Jan 1983.

[51] G. Wang and M. Tull. The implementation of an efficient and high-speed inner-product processor. *Proc. 35th Asilomar Conf, Signals, Systems, and Computers*, 2:1362–1366, Oct 2001.

[52] B. Wei, H. Du, and H. Chen. A complex-number multiplier using radix-4 digits. In *Proceedings of the 12th Symposium on Computer Arithmetic*, pages 84 –90, jul 1995.

[53] L. Zheng, S. Xu-Bang, and P. Zuo-Hui. The application of redundant encoding in iterative implementation of division and square root. In *Proceedings of the 4th International Conference on ASIC*, pages 603–606, 2001.