Carry-Free Implementations of Arithmetic Operations in FPGA

Vikram Voleti
International Institute of Information Technology, Hyderabad
Email: vikram.voleti@gmail.com

***Abstract*—Arithmetic operations like addition suffer from carry propagation which significantly delays the processing of signals in circuits. This paper details carry-free implementation of arithmetic operations. Detailed truth tables, logic circuits and circuit designs are constructed for carry-free addition, subtraction and multiplication. The circuits are implemented on an FPGA, and the path delays and space utilization are compared with those of standard implementation. The problem of implementing carry-free logic is analyzed, and design improvements to method proposed are provided.**

I. Introduction

Even with recent development in technology of integrated circuits, various high-speed circuits with regular structures and low power design still suffer from problems of path delay, limited range of the number of bits, and complexity in hardware. It is well-known that the addition of radix-B numbers suffers from carry propagation. Since the speed of digital processors depends mostly on the speed of the adders used in the system, it is critical to optimize the addition operation to speed up all computations.

Simple carry-ripple adders take time proportional to the length of the longest path from input to output. Even though this can be reduced to a depth of $O(log(n))$ by carry-lookahead adders [1], the depth still grows with the number of digits, and cannot be improved upon unless a different number system is used. Signed digit number systems [2-8] could be used to eliminate carry propagation and perform addition in $O(1)$ time. However, as [8] explains, the standard addition algorithms do not work for binary numbers (B = 2).

[9, 10] suggest to recode the input numbers so that subsequent addition and subtraction of binary numbers become carry-free. Parhami [9], in particular, recodes the numbers into a format that allows carry-free arithmetic operations effectively. In this paper, this format is used to recode binary numbers for carry-free arithmetic operations (see section IV). Since the focus is on using carry-free logic to perform arithmetic operations on binary numbers, the goal is to make arithmetic modules that take binary numbers as input, and perform addition or multiplication on them. To consider least memory requirement, only the minimal digit set $\{-1, 0, 1\}$ has been considered. It is also of significant advantage to have these circuits as reusable modules in building circuits for more complex arithmetic operations, like modular reduction, which has significant relevance in cryptographic applications.

In this paper, circuits for the carry-free addition of recoded numbers, binary numbers in a non-naive way, and carry-free multiplication of binary numbers have been designed. The modular nature of these circuits is exemplified by the use of Carry-Free Adders to build a Carry-Free Multiplier. These circuits have been implemented on an FPGA, and their speeds and area requirements have been compared with those of standard addition. They have then been analyzed for limitations. The contents of this paper are: (i) details of the conversion between binary, signed digit, and Recoded Binary Signed Digit (see section IV) numbers, with truth table and logic equations, (ii) implementation of arithmetic operations of addition, subtraction, multiplication using carry-free logic (sections V, VI), (iii) comparison of the carry-free implementations and their standard versions (section VII), (iv) analysis of the carry-free implementations (section VIII).

II. Related Work

[11] begins to design a carry-free multiplier. This paper details all the design features involved to create an end-to-end carry-free multiplier using carry-free adders as modules. [12] uses the same digit format as us, that mentioned in [9], but uses trinary signed digits. [13] uses bit-pair recording to generate partial products, and then adds them in the form of a binary tree using a carry-free adder. This paper's carry-free multiplier uses the Karatsuba multiplication algorithm [14] (see section VI) and performs all additions using a carry-free adder.

The quarternary signed digit system has been explored in light of carry-free implementations of addition [15-18] and multiplication [19-21]. We, however, restrict our paper to the format specified in [9], which only uses a binary signed digit format, recoded for carry-free operations. In addition, we provide our own circuit designs for carry-free adders of recoded numbers, binary numbers, and multiplier that uses carry-free adders as internal modules (see sections V, VI).

III. Our Implementation

To achieve carry-free implementations of arithmetic operations, the input binary numbers are first converted to a signed digit format. This paper uses Recoded Binary Signed Digit (RBSD, see section IV) format as described in [9]. Arithmetic operations in the domain of RBSD numbers can be implemented very effectively in a carry-free way.

Addition and multiplication are then performed on these RBSD numbers to produce binary signed digits.

These arithmetic operations can be summarized in the form of truth tables, as they are combinatorial in nature. The truth tables are then converted to Product-of-Sum equations so that electronic circuits could be designed that implement the tables. These electronic circuits are implemented in Verilog and run on an FPGA to measure the path delay and LUT usage. The results are compared with those of standard implementation of addition. The path delays are also analyzed to reveal the costliest path.

RBSD numbers and the conversions between binary, BSD and RBSD numbers are detailed in section IV. We mention why using RBSD numbers leads to carry-free arithmetic. Our carry-free implementations of arithmetic operations on RBSD numbers are described in section V. In section VI, we design end-to-end carry-free arithmetic operations on binary numbers.

IV. Recoded Binary Signed Digit Numbers

Carry-free operations can be implemented by first converting binary numbers to binary signed digits $\{-D..0..D\}$. The negative numbers in the digit set help extend the operation set of the field of numbers to carry-free operations.

Among the different methods of converting binary numbers into signed digits [7-10], we chose the solution provided by Parhami [9] to recode a given binary number x of length n to an equivalent Signed Digit number z of length $n+1$ such that there are no two neighboring digits z_{i+1} and z_i with $z_{i+1} * z_i = 1$. As proven in [9], this is essential to implementing carry-free addition, and shall be called "Recoded Binary Signed Digits", or RBSD. We limit our digit set to $\{-1, 0, +1\}$, and show that computational gains by using the least number of signed digits are significant in themselves.

In the following subsections, conversions between Binary, BSD and RBSD formats are detailed. As these are combinatorial circuits, there is no delay from carry propagation.

A. Binary to RBSD

Binary numbers are first converted to RBSD before performing any arithmetic operation on them. The table of conversion from two consecutive binary digits to one RBSD digit at the higher binary digits position is given in Table I. For example, the binary number 10110 is converted to $1\bar{1}10\bar{1}0$. Moving bit-by-bit from lower to higher bit, $0(0) \rightarrow 0$, $10 \rightarrow \bar{1}$, $11 \rightarrow 0$, $01 \rightarrow 1$, $10 \rightarrow \bar{1}$, $(0)1 \rightarrow 1$. As can be seen, the RBSD number contains one bit more than the original binary number.

The conversion from binary to RBSD is such that the value of the number remains the same, but there are no two consecutive 1s or -1s. This, in effect, eliminates the possibility of a Carry beyond one position. It is easy to see the conversion from binary to RBSD:

$$Z_i^s = X_i \ . \ \neg X_{i-1} \tag{1}$$

$$Z_i^v = (X_i \ . \ \neg X_{i-1}) + (\neg X_i \ . \ X_{i-1}) \tag{2}$$

Here, Z_i^s is the sign bit and Z_i^v is the value bit of the RBSD number. '.' is the and operation, '+' is the or operation, and '$\neg$' is $negation$.

TABLE I
Binary to RBSD Conversion

X_i	X_{i-1}	Z_i
0	0	0
0	1	1
1	0	-1 or 1
1	1	0

B. BSD to RBSD

It is important to note that the field of RBSD numbers is not closed under addition. This means that the addition of two RBSD numbers shall result in a BSD sum, but which is not necessarily an RBSD number. Thus, a Carry-Free Addition module of binary numbers (converted to RBSD) needs to be followed by a BSD-to-RBSD Converter. Although this makes for another overhead in computation, we shall see in section VII that the overall computation time is significantly lesser than that for standard addition.

The conversion from BSD to RBSD is a more complex process, because a binary number has only two possible digits $\{0, 1\}$, while a BSD number has three possible inputs $\{-1, 0, 1\}$. This conversion table has been provided in [9], which is summarized in Table II. In the table, 'X' implies "*don't care*", i.e. either of $\{-1, 0, 1\}$. As explained in [9], the conversion of a BSD bit y_i into an RBSD bit z_i requires the 3 following bits in the BSD number, y_{i-1}, y_{i-2}, y_{i-3}. It can be seen that this is also a combinatorial circuit, and takes $O(1)$ time.

TABLE II
BSD to RBSD Conversion

y_i	y_{i-1}	y_{i-2}	y_{i-3}	z_i	y_i	y_{i-1}	y_{i-2}	y_{i-3}	z_i
-1	-1	-1	X	0	0	1	-1	X	0
		0	-1	0			0	-1	0
		0	0	1			0	0	1
		0	1	1			0	1	1
		1	X	1			1	X	1
-1	0	-1	X	1	1	-1	-1	X	0
		0	X	-1			0	-1	0
		1	X	-1			0	0	1
-1	1	-1	X	-1			0	1	1
		0	-1	-1			1	X	1
		0	0	0	1	0	-1	X	1
		0	1	0			0	X	-1
		1	X	0			1	X	-1
0	-1	-1	X	-1	1	1	-1	X	-1
		0	-1	-1			0	-1	-1
		0	0	0			0	0	0
		0	1	0			0	1	0
		1	X	0			1	X	0
0	0	X	X	0					

C. BSD to Binary

Table III can be used to convert a BSD or RBSD number into its Binary form. It details the conversion of one bit of the BSD digit along with the carry bit at its position (0 for the least significant), into the Binary bit at the same position, and a carry bit for the next digit. The process involves carry propagation.

TABLE III
BSD (OR RBSD) TO BINARY CONVERSION

BSD (Z)	carry_in (c)	Binary (B)	carry_out (C)
-1	0	1	1
-1	1	0	1
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

The conversions in Table III can be represented through logical equations, Z^s is the sign bit, and Z^v is the value bit of Z as equations (3) and (4):

$$B = (\neg Zv \ . \ c) + (Zv \ . \ \neg c) \tag{3}$$
$$C = (\neg Zv \ . \ c) + Zs \tag{4}$$

V. CARRY-FREE OPERATIONS OF RBSD NUMBERS

In the following subsections, carry-free addition and subtraction of RBSD numbers are described.

A. Carry-Free Addition of RBSD numbers

Before dealing with Binary numbers, we first tackle the easier problem of designing carry-free addition of two RBSD numbers using the truth table in Table I. The sum digit of two numbers depends on two consecutive digits of each of the input numbers, since carry has to be accounted for. So we consider two bits each from the two inputs, X_i, X_{i-1}, Y_i, Y_{i-1}, to compute the sum at the higher bits position, S_i. The resulting truth table is documented in Table IV.

Table IV can transformed into a Product-of-Sum form to implement an electronic circuit that performs Carry-Free RBSD Addition. The optimized Product-of-Sum form of Table IV is given in equations (10) and (11):

$$S0 = \neg X_i^s + Y_i^s + \neg Y_i^v \tag{5}$$
$$S1 = \neg X_i^v + Y_i^v + \neg X_{i-1}^s + \neg Y_{i-1}^s \tag{6}$$
$$S2 = X_i^v + \neg Y_i^v + \neg X_{i-1}^s + \neg Y_{i-1}^s \tag{7}$$
$$S3 = \neg X_i^s + Y_i^s + \neg X_{i-1}^v + Y_{i-1}^s + \neg Y_{i-1}^v \tag{8}$$
$$S4 = X_i^v + \neg Y_i^s + X_{i-1}^s + \neg X_{i-1}^v + \neg Y_{i-1}^v \tag{9}$$

$$S_i^s = S0 \ . \ S1 \ . \ S2 \ . \ S3 \ . \ S4 \ . \ (\neg X_i^v + \neg Y_i^s) \ . \ (X_i^s + Y_i^s + X_{i-1}^s) \ . \ (X_i^s + Y_i^s + Y_{i-1}^s) \tag{10}$$

$$S_i^v = S0 \ . \ S1 \ . \ S2 \ . \ S3 \ . \ S4 \ . \ (X_i^s + \neg X_i^v + \neg Y_i^s) \ . \ (\neg X_i^v + \neg Y_i^v + X_{i-1}^v) \ . \ (\neg X_i^v + \neg Y_i^v + Y_{i-1}^v) \ . \ (X_i^v + Y_i^v + X_{i-1}^v) \ . \ (X_i^v + Y_i^v + Y_{i-1}^v) \ . \ (X_i^v + Y_i^v + \neg X_{i-1}^s + Y_{i-1}^s) \ . \ (X_i^v + Y_i^v + X_{i-1}^s + \neg Y_{i-1}^s) \tag{11}$$

Equations (10) and (11) describe the i-th sign and value bits of the Sum of two binary numbers X and Y at the i-th position. The computation of the sum bit at each position is performed by the "add_RBSD" module that uses circuits described by equations (10) and (11). It is important to note that the computation of the first and last bits require phantom 0s to complete their inputs.

TABLE IV
BSD TO RBSD CONVERSION

X_i	Y_i	X_{i-1}	Y_{i-1}	S_i	X_i	Y_i	X_{i-1}	Y_{i-1}	S_i
-1	-1	0	0	0	0	0	0	X	0
		0	1	0			1	-1	0
		1	0	0			1	1	1
		1	1	0	0	1	-1	-1	0
-1	0	0	X	-1			X	0	1
		1	-1	-1			0	-1	1
		1	0	-1			1	-1	1
		1	1	0	1	-1	-1	0	0
-1	1	0	-1	0			1	1	0
		0	0	0			0	0	0
		1	-1	0			0	1	0
		1	0	0	1	0	-1	-1	0
0	-1	X	0	-1			-1	0	1
		-1	1	-1			-1	1	1
		0	1	-1			0	X	1
		1	1	0	1	1	-1	-1	-1
0	0	-1	-1	-1			-1	0	0
		X	0	0			0	-1	0
		-1	1	0			0	0	0

B. Carry-Free Subtraction of RBSD Numbers

The negative of an RBSD number can be obtained by interchanging the 1s and -1s in it. For example, RBSD $[10-1]$ is decimal 3, while RBSD $[-101]$ is -3. Thus, by carrying out Carry-Free Addition of the first number and the negative of the second number, Carry-Free Subtraction is achieved.

VI. CARRY-FREE ARITHMETIC OPERATIONS OF BINARY NUMBERS

The following subsections describe carry-free addition and multiplication of binary numbers.

A. Carry-Free Addition of Binary Numbers

A naive way of performing the addition of two binary numbers is by simply connecting a Binary-to-RBSD Converter module in cascade to the Carry-Free Adder for RBSD Numbers, designed previously in Section IV.A. Although this is functionally correct, a better approach would be to combine the truth tables of these two modules, and make a single truth table that takes as input binary digits, and produces their sum.

Thus, a new truth table, Table V, is constructed. The inputs consist of three consecutive bits each of the two input binary numbers, A and B, since every bit of an RBSD number depends on two bits in the binary number, and RBSD addition at a bit requires two RBSD bits. So there are 64 combinations of binary inputs to be taken care of. Each combination results in two RBSD numbers, which then needed to be added using Table IV. The output sum bit is placed in the most significant position among the three input positions. The columns in Table V are respectively A_i, A_{i-1}, A_{i-2}, B_i, B_{i-1}, B_{i-2}, and the sum S_i.

Then Table V was simplified into the optimal Product-of-Sum form as given in equations (19) and (20):

$$S0 = A_i + A_{i-1} + \neg B_i + \neg B_{i-1} \quad (12)$$

$$S1 = A_i + A_{i-1} + \neg A_{i-2} + \neg B_i + B_{i-1} + \neg B_{i-2} \quad (13)$$

$$S2 = A_i + \neg A_{i-1} + A_{i-2} + B_i + \neg B_{i-1} + \neg B_{i-2} \quad (14)$$

$$S3 = \neg A_i + A_{i-1} + B_i + \neg B_{i-1} \quad (15)$$

$$S4 = \neg A_i + A_{i-1} + \neg A_{i-2} + B_i + B_{i-1} + \neg B_{i-2} \quad (16)$$

$$S5 = \neg A_i + \neg A_{i-1} + A_{i-2} + \neg B_i + \neg B_{i-1} + \neg B_{i-2} \quad (17)$$

$$S6 = \neg A_i + \neg A_{i-1} + \neg A_{i-2} + \neg B_i + \neg B_{i-1} \quad (18)$$

$$S_i^s = S0 \,.\, S1 \,.\, S2 \,.\, S3 \,.\, S4 \,.\, S5 \,.\, S6 \,.\, (A_i + A_{i-1} + B_i) \,.\, (A_i + \neg A_{i-1} + A_{i-2} + B_i + B_{i-1}) \,.\, (A_i + \neg A_{i-1} + A_{i-2} + \neg B_i) \,.\, (A_i + \neg A_{i-1} + \neg A_{i-2}) \,.\, (\neg A_i + A_{i-1} + \neg B_i + B_{i-1}) \,.\, (\neg A_i + \neg A_{i-1} + B_i) \quad (19)$$

$$S_i^v = S0 \,.\, S1 \,.\, S2 \,.\, S3 \,.\, S4 \,.\, S5 \,.\, S6 \,.\, (A_i + A_{i-1} + A_{i-2} + B_i + B_{i-1}) \,.\, (A_i + A_{i-1} + \neg A_{i-2} + B_i + B_{i-1}) \,.\, (A_i + \neg A_{i-1} + \neg B_i + B_{i-1}) \,.\, (A_i + A_{i-1} + A_{i-2} + \neg B_i + \neg B_{i-1} + B_{i-2}) \,.\, (A_i + \neg A_{i-1} + \neg A_{i-2} + B_i + \neg B_{i-1}) \,.\, (\neg A_i + A_{i-1} + A_{i-2} + \neg B_i + B_{i-1}) \,.\, (\neg A_i + A_{i-1} + \neg A_{i-2} + \neg B_i + B_{i-1} + B_{i-2}) \,.\, (\neg A_i + \neg A_{i-1} + B_i + B_{i-1}) \,.\, (\neg A_i + \neg A_{i-1} + A_{i-2} + B_i + \neg B_{i-1} + B_{i-2}) \quad (20)$$

B. Carry-Free Subtraction of Binary Numbers

It is fairly straightforward to see that subtraction of two binary numbers can be achieved by the addition of one number with the negative of the other, as mentioned in Section IV.B. In this case, the two binary numbers must be converted to their 1s complement or 2s complement forms before converting them into RBSD numbers. Section VIII.A deals with this further.

C. Carry-Free Multiplication of Binary Numbers

Multiplication forms a very critical part of arithmetic operations in signal processing applications. The school-book multiplication algorithm is time-intensive ($O(n^2)$), and so is not a preferable option for core operations.

Karatsuba Algorithm [14] for multiplication proves to be the ideal algorithm, providing a much faster $O(n^{Log_2 3})$ time. Making this algorithm carry-free can prove quite useful. Since multiplication involves repeated addition, a cascading of carry-free addition modules produces carry-free multiplication.

The Karatsuba algorithm implements multiplication of two numbers, x and y, by first splitting them into two parts: $x = x_1 B^m + x_0$, $y = y_1 B^m + y_0$, at some digit place m. The standard multiplication of x and y would require 4 multiplications, as can be seen from equation (21).

TABLE V
CARRY-FREE ADDITION OF BINARY NUMBERS

0	0	0	0	0	0	0	1	0	0	0	0	0	-1
			0	0	1	0				0	0	1	-1
			0	1	0	1				0	1	0	0
			0	1	1	1				0	1	1	0
			1	0	0	-1				1	0	0	0
			1	0	1	-1				1	0	1	0
			1	1	0	0				1	1	0	-1
			1	1	1	0				1	1	1	-1
0	0	1	0	0	0	0	1	0	1	0	0	0	-1
			0	0	1	1				0	0	1	0
			0	1	0	1				0	1	0	0
			0	1	1	1				0	1	1	0
			1	0	0	-1				1	0	0	0
			1	0	1	0				1	0	1	1
			1	1	0	0				1	1	0	-1
			1	1	1	0				1	1	1	-1
0	1	0	0	0	0	1	1	1	0	0	0	0	0
			0	0	1	1				0	0	1	0
			0	1	0	1				0	1	0	0
			0	1	1	0				0	1	1	1
			1	0	0	0				1	0	0	-1
			1	0	1	0				1	0	1	-1
			1	1	0	0				1	1	0	-1
			1	1	1	1				1	1	1	0
0	1	1	0	0	0	1	1	1	1	0	0	0	0
			0	0	1	1				0	0	1	0
			0	1	0	0				0	1	0	1
			0	1	1	0				0	1	1	1
			1	0	0	0				1	0	0	-1
			1	0	1	0				1	0	1	-1
			1	1	0	1				1	1	0	0
			1	1	1	1				1	1	1	0

$$x * y = (x_1 B^m + x_0) * (y_1 B^m + y_0) = (x_1 * y_1) B^{2m} + (x_1 * y_0 + x_0 * y_1) B^m + x_0 * y_0 \quad (21)$$

But the Karatsuba algorithm requires only 3 multiplications, given in equations (22), (23), and (24).

$$z_0 = x_1 * y_1 \quad (22)$$

$$z_1 = x_0 * y_0 \quad (23)$$

$$z_2 = (x_1 + x_0) * (y_1 + y_0) \quad (24)$$

Using $z0$, $z1$ and $z2$, the product of x and y is computed in equation (25).

$$x * y = z_0 B^{2m} + (z_2 - z_1 - z_0) B^m + z_1 \quad (25)$$

Thus, the Karatsuba algorithm decreases the number of multiplications by 1, while increasing the number of additions (subtractions are counted as additions) by 1. Since addition is an $O(1)$ process using the carry-free adder designed in the previous section, this is a highly feasible compromise.

Figure 1 is the circuit implementation of the Karatsuba Algorithm, for carry-free multiplication of two 66-bit binary numbers A and B. As can be seen, the three multiplications are performed on binary numbers. One of these required the conversion from BSD to binary, which is a carry-full step, and is expected to contribute significantly to the path delay. It shall be seen in section VII that it is just so. However, it shall also be seen that despite this, the path delay of

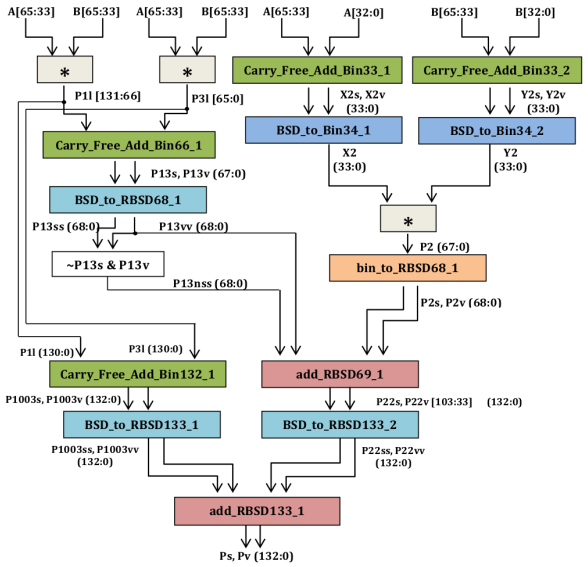

Fig. 1. Circuit design of carry-free muliplication of binary numbers.

carry-free multiplication is lesser than that of standard addition. It is to be noted that the output of every module is converted to RBSD, to ensure carry-free operation in the subsequent modules.

VII. Results

All circuits were designed in Verilog and run using Xilinx-9 on a Virtex-5 FPGA 9(5vlx30ff324-3). All modules were also coded in Octave to verify their computations.

A. Addition

Table VI lists the comparison of computation times of addition and multiplication between standard implementations and carry-free implementations described in the previous sections. Two metrics were analysed to compare the performance path delay, Slice Logic Utilization, i.e. the percentage of Slice LUTs on the FPGA utilized, which is equivalent to the Slice Logic Distribution, i.e. the number of LUT Flip-Flop pairs utilized, out of the 19200 present. As can be seen from Table VI, carry-binary binary addition takes lesser memory, as well as significantly lesser time than standard addition.

B. Multiplication

Table VI also lists the path delay and LUT usage for the carry-free implementation of multiplication. As can be seen, carry-free multiplication takes lesser time than even standard addition, it can be inferred that it takes significantly lesser time than standard multiplication.

Table VII lists the path delays of individual modules in the Carry-Free Multiplier, in ascending order of time taken. It can be observed that the most time-consuming module is the BSD_to_Bin33 module, which involves carry-full operation. All carry-free operations clearly take less than a third of the time taken by a simple BSD to binary conversion. The advantage of carry-free operation is highlighted in this result.

TABLE VI
Comparison of Results

Metric	Standard Add	CF RBSD Add	CF Binary Add	CF Mul
Path delays	71.3 ns	4.8 ns	4.0 ns	25 ns
Slice LUTs	2%	4%	1%	11%
LUT FF pairs	384	928	265	2223

TABLE VII
Path Delays in Carry-Free Multiplication

Module	Delay	BSD_to_RBSD68	4.0 ns
Bin_to_RBSD68	3.6 ns	BSD_to_RBSD132	4.0 ns
CF_Add_Bin33	4.0 ns	add_RBSD_69	4.8 ns
CF_Add_Bin66	4.0 ns	add_RBSD_133	4.8 ns
CF_Add_Bin132	4.0 ns	BSD_to_Bin33	12.8 ns

VIII. Analysis

A. Use of 2's Complement binary numbers

Throughout the paper, binary numbers have been assumed to be positive, where n-bits of the binary number convert to $2*(n+1)$ bits of the RBSD number ($n+1$ each of sign and value bits). However, negative numbers, i.e. the possible use of binary numbers as being in their 2's complement forms, has not been explored. The good news here is that this incorporation is particularly easy. In its 2's complement form, there shall be no increase in number of bits in the binary number's conversion to RBSD, since it already has an extra bit included. If the number is negative, the most significant bit of its RBSD form shall inevitably be -1. Thus, the number of bits would increase to just $2n$.

B. Carry Free Adder + BSD-to-RBSD converter

The field of RBSD numbers is not closed under addition, so the sum of two RBSD numbers results in a BSD number that is not necessarily RBSD. Since many applications require further use of the sum (e.g. multiplication), it is imperative to convert it into RBSD before using it elsewhere. Fortunately, BSD to RBSD conversion is combinational, so it does not delay the circuit much. This is why every Carry-Free Adder is followed by a BSD-to-RBSD converter in the Carry-Free Multiplier (Figure 1).

It can be argued that a Carry-Free Adder and a BSD-to-RBSD converter should be combined to make one single truth table, so that there is no delay in adding RBSD or binary numbers into an RBSD sum. However, making such a concise table is quite painful. The BSD-to-RBSD converter takes 4 consecutive digits of the input BSD number, and produces 1 output digit at the most significant digit position. A Carry-Free Adder takes 2 consecutive digits from each RBSD input, or 3 from each binary input, to produce one output digit. Since these act as inputs to the BSD-to-RBSD Converter, the Carry-Free Adders require 5 consecutive RBSD digits, or 6 consecutive binary bits to produce one output digit. This leads to a table having $3^5 = 243$ rows in case of RBSD Adder, and $2^6 = 64$ rows in case of Binary adder. It is practically infeasible, and quite unnecessary, to manually construct a Product-of-Sum form for the combined truth table.

C. Carry-Free Karatsuba Multiplication

1) Multiplication of BSD numbers: The first step of Karatsuba multiplication involves splitting the input number into two parts. In case of BSD numbers, simply splitting the number at the middle is mathematically incorrect, and leads to erroneous results. Moreover, mathematically sound splitting involves carry propagation, which opposes our goal of fast computation. So, the three multiplications in the circuit are of binary numbers (see Figure 1), BSD-to-Binary conversion having explicitly been done in one case for this purpose.

2) Carry-full 34-bit BSD-to-Binary converter: As seen from Table VII, the two 34-bit RBSD-to-Binary converters used in the multiplier (see Figure 1) are the major contributors to the path delay of this circuit, as they involve carry propagation. It cannot be avoided.

It could be argued that the use of two 17-bit RBSD-to-Binary converters can help decrease path delay, since they can be run in parallel. However, this would involve splitting the RBSD number which, as discussed above, results in some carry propagation of its own. It is to be seen whether the overall path delay is indeed lesser.

3) Multiplication of integers: Ultimately, multiplication operation is performed using integer arithmetic on binary numbers, which involves carry propagation. To speed this up, it can be argued that the 34-bit numbers can be split into two 17-bit numbers, and integer multiplication can be carried out in parallel over them. However, this is equivalent to adding one more level to the Karatsuba multiplication. This shall result in thrice the number of multiplications involved. It needs to be checked through experimentation whether this incurs lesser path delay.

4) Multiplication of binary numbers with more than 64 bits: The original design was meant for 64-bit binary numbers, the extra two bits can be set to 0 to add two n-bit numbers, where $n <= 64$. For $n > 64$, each input number could be split into the lowest 64 bits and the rest. The Karatsuba algorithm can then be recursively employed on the two parts.

IX. Conclusion

It has been quantifiably verified that carry-free implementations of arithmetic operations have significant advantage over standard implementations in terms of path delay and memory. A carry-free multiplier thus designed takes lesser time than standard addition itself. It has been analyzed that the most optimized carry-free implementations are painful to design, so a modular approach of combining Carry-Free Adders to make more complex arithmetic modules is practical.

Acknowledgment

The author would like to thank Sujoy Sinha Roy and Prof. Ingrid Verbauwhede of KU Leuven, Belgium, for the opportunity to collaborate with them on this work.

References

[1] P. Kogge and H. Stone, *A parallel algorithm for the efficient solution of a general class of recurrences*, IEEE Transactions on Computers (T-C), vol. 22, pp. 786-793, 1973.

[2] B. Parhami, *Computer Arithmetic - Algorithms and Hardware Designs*. Oxford University Press, 2000.

[3] A. Avizienis, *Signed-digit number representations for fast parallel arithmetic*, IRE Transactions on Electronic Computers, vol. 10, no. 3, pp. 389-400, 1961.

[4] B. Parhami, *Generalized signed-digit number systems: A unifying framework for redundant number representations*, IEEE Transactions on Computers (T-C), vol. 39, no. 1, pp. 89-98, 1990.

[5] F. Kharbash, G. M. Chaudhry, *Reliable Binary Signed Digit Number Adder Design*, IEEE Computer Society Annual Symposium on VLSI, pp 479-484, 2007.

[6] J. Moskal, E. Oruklu and J. Saniie, *Design and Synthesis of a Carry-Free Signed-Digit Decimal Adder*, IEEE International symposium on Circuits and Systems, pp. 1089-1092, 2007.

[7] R. Rani, N. Sharma, L. K. Singh, *Fast Computing using Signed Digit Number System*, IEEE proceedings of International Conference on Control, Automation, Communication And Energy Conservation, 2009.

[8] K. Schneider, A. Willenbucher, *A New Algorithm for Carry-Free Addition of Binary Signed-Digit Numbers*, IEEE 22nd International Symp. Field-Programmable Custom Computing Machines, 2014.

[9] B. Parhami, *Carry-free addition of recoded binary signed-digit numbers*, IEEE Transactions on Computers (T-C), vol. 37, no. 11, pp. 1470-1476, 1998.

[10] M. Joye and S. M. Yen, *Optimal left-to-right binary signed-digit recoding*, IEEE Transactions on Computers (T-C), vol. 49, no. 7, pp. 740-748, 2000.

[11] J. U. Ahmed, A. A. S. Awwal, *Multiplier design using RBSD number system*, Proceedings of the 1993 National Aerospace and Electronics Conference, vol. 1, pp. 180-184, 1993.

[12] A. K. Cherri, M. S. Alam, *Recoded and nonrecoded trinary signed-digit multipliers designs using redundant bit representations*, Aerospace and Electronics Conference 1998. NAECON 1998. Proceedings of the IEEE 1998 National, pp. 505-512, 1998, ISSN 0547-3578, 1998.

[13] T. N. Rajashekhar and O. Kal, *Fast Multiplier Design using Redundant Signed-Digit Numbers*, International Journal of Electronics, vol .69, no. 3, pp. 359-368, 1990.

[14] A. Karatsuba and Y. Ofman, *Multiplication of Many-Digital Numbers by Automatic Computers*, Proceedings of the USSR Academy of Sciences. 145: 293-294. Translation in the academic journal Physics-Doklady, 7 (1963), pp. 595-596, 1962.

[15] A. A. S. Awwal and J. U. Ahmed, *Fast carry free adder design using QSD number system*, Proceedings of the IEEE 1993 National Aerospace and Electronic Conference, vol 2, pp. 1085-1090, 1993.

[16] R. Rani, L. K. Singh and N. Sharma, *A Novel design of High Speed Adders Using Quaternary Signed Digit Number System*, International Journal of Computer and Network Security (IJCNS), Vol. 2, No. 9, pp. 62-66, 2010.

[17] N. W. Umredkar, M. A. Gaikwad, *Review of Quaternary Adders in Voltage Mode Multi-Valued Logic*, International Journal of Computer Applications (09758887), 2013.

[18] S. Dubey, R. Rani, *VLSI Implementation of Fast Addition using Quaternary Signed Digit Number System*, IEEE International Conference on Emerging Trends in Comp...ICECCN 2013.

[19] O. Ishizuka, A. Ohta, K. Tannno, Z. Tang, D. Handoko, *VLSI design of a quaternary multiplier with direct generation of partial products*, Proceedings of the 27th International Symposium on Multiple-Valued Logic, pp. 169-174, 1997.

[20] S. Datla and M. Thornton, *Quaternary Voltage-Mode Logic Cells and Fixed-Point Multiplication Circuits*, Multiple-Valued Logic (ISMVL) 2010 40th IEEE International Symposium on, pp. 128-133, 2010, ISSN 0195-623X, 2010.

[21] N. Sharma, B. S. Rai and A. Kumar, *Design of RBSD Adder and Multiplier Circuits for High Speed Arithmetic Operations and Their Timing Analysis*, Special Russian Issue: Advances in Computer Science and Engineering, Research in Computing Science, pp. 243-25, 2006.