International Journal of Trend in Research and Development, Volume 2(4), ISSN 2394-9333
www.ijtrd.com

Design Quaternary Multiplication using Quaternary Sign Digit Number addition

[1]P. Hareesh, [2]Ch. KalyanChakravarthi and [3]D. TirumalaRao
[1]M.Tech ECE (VLSI&ES), [2]Assistant Professor and [3]Associate Professor,
[1,2,3]Department of ECE, GMR Institute of Technology, Rajam, India.

Abstract

Common binary arithmetic operations such as addition/subtraction and multiplication suffer from O(n) carry propagation delay where n is the number of digits. Carry lookahead helps to improve the propagation delay to O(log n), but is bounded to a small number of digits due to the complexity of the circuit. A carry-free arithmetic operation can be achieved using a higher radix number system such as Quarternary Signed Digit (QSD). In QSD, each digit can be represented by a number from -3 to 3. This number system allows multiple representations of any integer. By exploiting this feature, we can design an adder without ripple carry. The implementation of quarternary addition and multiplication results in a fix delay independent of the number of digits. Operations on a large number of digits such as 64, 128, or more, can be implemented with constant delay and less complexity. This paper focuses on the implementation of quarternary addition and multiplication. Results are verified and the performance is shown to be consistent with the constant delay model.

Keywords: quaternary signed digit, programmable logic.

1. Introduction

Arithmetic operations are widely used and play important roles in various digital systems such as computers and signal processors. QSD number representation has attracted the interest of many researchers. Additionally, recent advances in technologies for integrated circuits make large scale arithmetic circuits suitable for VLSI implementation [1][2]. However, arithmetic operations still suffer from known problems including limited number of bits, propagation time delay, and circuit complexity.

In this paper, we propose a high speed QSD arithmetic logic unit which is capable of carry free addition, borrow free subtraction, up-down count and multiply operations. The QSD addition/subtraction operation employs a fixed number of minterms for any operand size. The multiplier is composed of partial product generators and adders. For convenience of testing and to verify results, we choose to implement the units using a programmable logic device.

This paper is organized as follows. Section 2 presents the quaternary signed digit (QSD) number. The adder/subtractor and multiplier design are detailed in section 3 and section 4, respectively. Section 5 presents results and performance. Section 6 presents conclusions and future work.

2. QSD Numbers

QSD numbers are represented using 3-bit 2's complement notation. Each number can be represented by

$$D = \sum_{i}^{n} x_i 4^i, \qquad (1)$$

where x_i can be any value from the set $\{\bar{3},\bar{2},\bar{1},0,1,2,3\}$ for producing an appropriate decimal representation. A QSD negative number is the QSD complement of the QSD positive number i.e., $\bar{3} = -3$, $\bar{2} = -2$ and $\bar{1} = -1$. For example, $1233_{QSD} = 23_{10}$ and $1\ 23\bar{3}_{QSD} = -23_{10}$.

3. Adder/Subtractor Design

Addition is the most important arithmetic operation in digital computation. A carry-free addition is highly desirable as the number of digits becomes large. We can achieve carry-free addition by exploiting the redundancy of QSD numbers and the QSD addition. The redundancy allows multiple representations of any integer quantity i.e., $6_{10}=12_{QSD}=22\bar{}_{QSD}$.

There are two steps involved in the carry-free addition. The first step generates an intermediate carry and sum from the addend and augend. The second step combines the intermediate sum of the current digit with the carry of the lower significant digit. To prevent carry from further rippling, we define two rules. The first rule states that the magnitude of the intermediate sum must be less than or equal to 2. The second rule states that the magnitude of the carry must be less than or equal to 1. Consequently, the magnitude of the second step output

cannot be greater than 3 which can be represented by a single-digit QSD number; hence no further carry is required. In step 1, all possible input pairs of the addend and augend are considered. The output ranges from -6 to 6 as shown in Table 1.

Table 1. The outputs of all possible combinations of a pair of addend (A) and augend (B).

A \ B	-3	-2	-1	0	1	2	3
-3	-6	-5	-4	-3	-2	-1	0
-2	-5	-4	-3	-2	-1	0	1
-1	-4	-3	-2	-1	0	1	2
0	-3	-2	-1	0	1	2	3
1	-2	-1	0	1	2	3	4
2	-1	0	1	2	3	4	5
3	0	1	2	3	4	5	6

The range of the output is from -6 to 6 which can be represented in the intermediate carry and sum in QSD format as show in Table 2. Some numbers have multiple representations, but only those that meet the defined rules are chosen. The chosen intermediate carry and sum are listed in the last column of Table 2.

Table 2. The intermediate carry and sum between -6 to 6.

Sum	QSD represented number	QSD coded number
-6	$\bar{2}2,1\,\bar{2}$	$\bar{1}\,\bar{0}$
-5	$\bar{2}3,\bar{1}\,\bar{1}$	$\bar{1}\,1$
-4	$\bar{1}0$	$\bar{1}0$
-3	$\bar{1}1,0\bar{3}$	$\bar{1}\bar{1}$
-2	$\bar{1}2,0\bar{2}$	$0\bar{2}$
-1	$\bar{1}3,0\,\bar{0}$	$0\,\bar{0}$
0	00	00
1	$01,1\bar{3}$	01
2	$02,1\bar{2}$	02
3	$03,\bar{1}$	$1\,\bar{0}$
4	10	10
5	$11,2\bar{3}$	11
6	$12,2\bar{2}$	12

Both inputs and outputs can be encoded in 3-bit 2's complement binary number. The mapping between the inputs, addend and augend, and the outputs, the intermediate carry and sum are shown in binary format in Table 3. Since the intermediate carry is always between -1 and 1, it requires only a 2- bit binary representation. Finally, five 6-variable Boolean expressions can be extracted. The intermediate carry and sum circuit is shown in Figure 1.

Table 3. The mapping between the inputs and outputs of the intermediate carry and sum.

INPUT				**OUTPUT**				
QSD		Binary		Decimal	QSD		Binary	
A_i	B_i	A_i	B_i	Sum	C_i	S_i	C_i	S_i
3	3	011	011	6	1	2	01	010
3	2	011	010	5	1	1	01	001
2	3	010	011	5	1	1	01	001
3	1	011	001	4	1	0	01	000
1	3	001	011	4	1	0	01	000
2	2	010	010	4	1	0	01	000
1	2	001	010	3	1	-1	01	111
2	1	010	001	3	1	-1	01	111
3	0	011	000	3	1	-1	01	111
0	3	000	011	3	1	-1	01	111
1	1	001	001	2	0	2	00	010
0	2	000	010	2	0	2	00	010
2	0	010	000	2	0	2	00	010
3	-1	011	111	2	0	2	00	010
-1	3	111	011	2	0	2	00	010
0	1	000	001	1	0	1	00	001
1	0	001	000	1	0	1	00	001
2	-1	010	111	1	0	1	00	001
-1	2	111	010	1	0	1	00	001
3	-2	011	110	1	0	1	00	001
-2	3	110	011	1	0	1	00	001
0	0	000	000	0	0	0	00	000
1	-1	001	111	0	0	0	00	000
-1	1	111	001	0	0	0	00	000
2	-2	010	110	0	0	0	00	000
-2	2	110	010	0	0	0	00	000
-3	3	101	011	0	0	0	00	000
3	-3	011	101	0	0	0	00	000
0	-1	000	111	-1	0	-1	00	111
-1	0	111	000	-1	0	-1	00	111
-2	1	110	001	-1	0	-1	00	111
1	-2	001	110	-1	0	-1	00	111
-3	2	101	010	-1	0	-1	00	111
2	-3	010	101	-1	0	-1	00	111
-1	-1	111	111	-2	0	-2	00	110
0	-2	000	110	-2	0	-2	00	110
-2	0	110	000	-2	0	-2	00	110
-3	1	101	001	-2	0	-2	00	110
1	-3	001	101	-2	0	-2	00	110
-1	-2	111	110	-3	-1	1	11	001
-2	-1	110	111	-3	-1	1	11	001
-3	0	101	000	-3	-1	1	11	001
0	-3	000	101	-3	-1	1	11	001
-3	-1	101	111	-4	-1	0	11	000
-1	-3	111	101	-4	-1	0	11	000
-2	-2	110	110	-4	-1	0	11	000
-3	-2	101	110	-5	-1	-1	11	111
-2	-3	110	101	-5	-1	-1	11	111
-3	-3	101	101	-6	-1	-2	11	110

In step 2, the intermediate carry from the lower significant digit is added to the sum of the current digit

to produce the final result. The addition in this step produces no carry because the current digit can always absorb the carry-in from the lower digit. Table 4 shows all possible combinations of the summation between the intermediate carry and the sum.

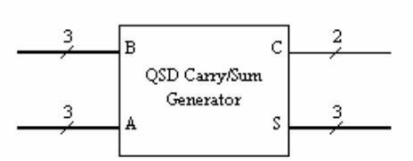

Figure 1. The intermediate carry and sum generator.

Table 4. The outputs of all possible combinations of a pair of intermediate carry (A) and sum (B).

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

Theresultof addition in this steprangesfrom -3 to 3. Since carry is not allowed in this step, the result becomes a single digit QSD output. The inputs, the intermediate carry and sum, are 2-bit and 3-bit binary respectively. The output is a 3-bit binary represented QSD number. The mapping between the 5-bit input and the 3-bit output is shown in Table 5.

Table 5. The mapping between inputs and outputs of the second step QSD adder.

INPUT				OUTPUT		
QSD		Binary		Decimal	QSD	Binary
A_i	B_i	A_i	B_i	Sum	S_i	S_i
1	2	01	010	3	3	111
1	1	01	001	2	2	010
0	2	00	010	2	2	010
0	1	00	001	1	1	001
1	0	01	000	1	1	001
-1	2	11	010	1	1	001
0	0	00	000	0	0	000
1	-1	01	111	0	0	000
-1	1	11	001	0	0	000
0	-1	00	111	-1	-1	111
-1	0	11	000	-1	-1	111
1	-2	01	110	-1	-1	111
-1	-1	11	111	-2	-2	110
0	-2	00	110	-2	-2	110
-1	-2	11	110	-3	-3	001

Three 5-variable Boolean expressions can be extracted from Table 5. Figure 2 shows the diagram of the second step adder. The implementation of an *n*-digit QSD adder requires *n* QSD carry and sum generators and *n* -1 second step adders as shown in Figure 3. The result turns out to be an *n*+1-digit number.

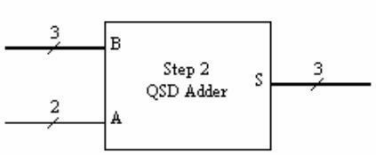

Figure 2. The second step QSD adder.

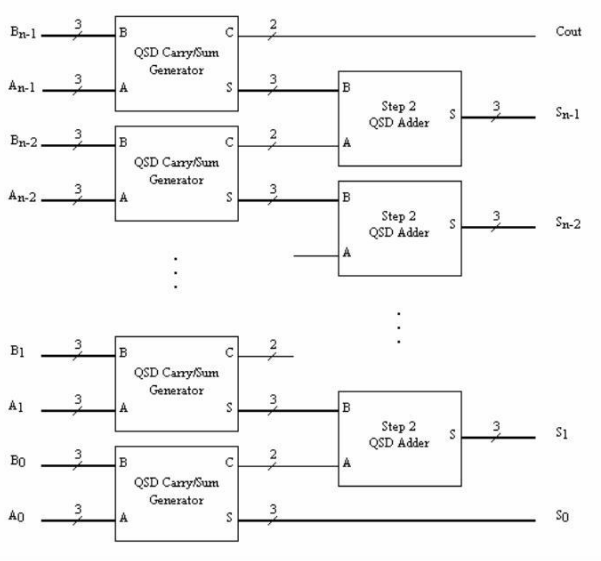

Figure 3. *n*-digit QSD adder.

4. Multiplier Design

There are generally two methods for a multiplication operation: parallel and iterative. QSD multiplication can be implemented in both ways, requiring a QSD partial product generator and QSD adder as basic components. A partial product, M_i, is a result of multiplication between an *n*-digit input, A_{n-1}-A_0, with a single digit input, B_i, where i = 0..*n*-1. The primitive component of the partial product generator is a single-digit multiplication unit whose functionality can be expressed as shown in Table 6.

Table 6. The outputs of all possible combinations of a pair of multiplicand (A) and multiplier (B).

A \ B	-3	-2	-1	0	1	2	3
-3	9	6	3	0	-3	-6	-9
-2	6	4	2	0	-2	-4	-6
-1	3	2	1	0	-1	-2	-3
0	0	0	0	0	0	0	0
1	-3	-2	-1	0	1	2	3
2	-6	-4	-2	0	2	4	6
3	-9	-6	-3	0	3	6	9

The single-digit multiplication produces *M* as a result and *C* as a carry to be combined with *M* of the next digit. The range of both outputs, *M* and *C*, is

between -2 and 2. According to Table 8, and using the same procedure as in creating Table 3 and 5, the mapping between the 6-bit input, *A* and *B*, to the 6-bit output, *M* and *C*, results in six 6-varible Boolean expressions which represent a single-digit multiplication operation. The diagram of a single-digit QSD multiplier is shown in Figure 4.

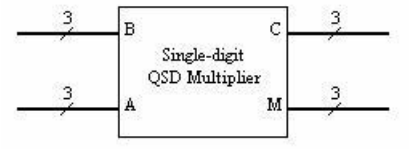

Figure 4. A single-digit QSD multiplier.

The implementation of an *n*-digit partial product generator uses *n* units of the single-digit QSD multiplier. Gathering all the outputs to produce a partial product result presents a small challenge. The QSD representation of a single digit multiplication output, shown in Table 7, contains a carry-out of magnitude 2 when the output is either -9 or 9. This prohibits the use of the second step QSD adder alone as a gatherer. In fact, we can use the complete QSD adder from the previous section as the gatherer. Furthermore, the intermediate carry and sum circuit can be optimized by not considering the input of magnitude 3. The QSD partial product generator implementation is shown in Figure 5.

Table 7. The QSD representation of a single-digit multiplication output.

Mult	QSD represented Number	QSD coding Number
-9	$\bar{2}\ \bar{1},33$	$\bar{2}\ \bar{1}$
-6	$\bar{2}2,1\ \bar{2}$	$\bar{1}\ \bar{2}$
-4	$\bar{1}0$	$\bar{1}0$
-3	$\bar{1}\ 1,0\bar{3}$	$\bar{1}1$
-2	$\bar{1}\ 2,0\bar{2}$	$0\bar{2}$
-1	$\bar{1}3,0\ \bar{1}$	$0\ \bar{1}$
0	00	00
1	$01,1\bar{3}$	01
2	$02,1\bar{2}$	02
3	$03,1\ \bar{1}$	$1\ \bar{1}$
4	10	10
6	$12,2\bar{2}$	12
9	$21,3\bar{3}$	21

An *n*x*n*-digit QSD multiplication requires *n* partial product terms. In an iterative implementation, a *2n*-digit QSD adder is used to perform add-shift operations between the partial product generator and the accumulator. After *n* iterations, the multiplication is complete. In contrast, a parallel implementation requires *n* partial product circuits and *n*-1 QSD adder units. A binary reduction sum is applied to reduce the propagation delay to O(log *n*). The schematic of a 4x4 parallel QSD multiplication is shown in Figure 6.

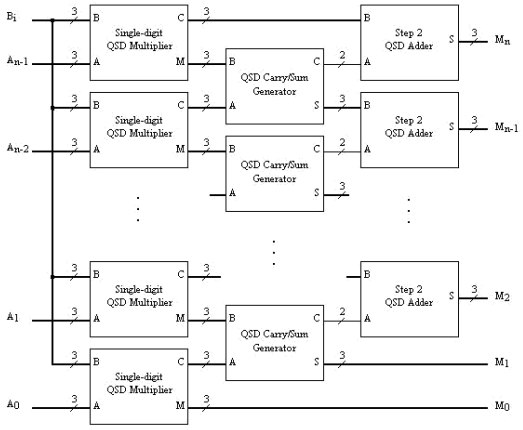

Figure 5. The *n*-digit QSD partial product generator.

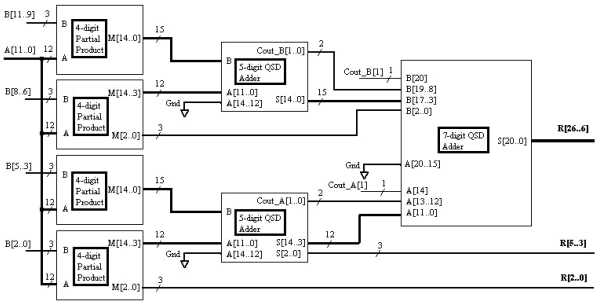

Figure 6. The 4x4 parallel QSD multiplication circuit

5. Results

The QSD adder and multiplication circuit are written in VHDL and synthesized on Altera FPGA devices using LeonardoSpectrum^{tm} from Mentor Graphics. The results of the implemented QSD addition and multiplication operations were collected from the timing simulation of the Altera MAX+plus II software. The correctness of the results is confirmed.

The device chosen for implementation is an Altera FLEX10K device. The Altera FLEX10K devices [5] are the industry's first embedded PLDs. Based on reconfigurable CMOS SRAM elements, the Flexible Logic Element MatriX (FLEX) architecture incorporates all features necessary to implement common gate array megafunctions with up to 250,000 gates. The EPF10K70 device is ideal for intermediate to advanced design including computer architecture,

communications, and DSP applications. The EPF10K70 device has over 70,000 typical gates, 3,744 Logic Elements (LEs), and 9 Embedded Array Blocks (EABs).

We test the performance of the QSD adder against the binary ripple carry adder and the high performance Altera megafunction adder. The comparison of the registered performance is shown in Figure 7 and Table 8. The test is performed for various sizes of the adder. The QSD number is capable of representing twice as much magnitude in each digit compared to the binary representation. Therefore, for an *n*-bit binary adder comparison, the *n/2*-bit QSD implementation is used for comparison.

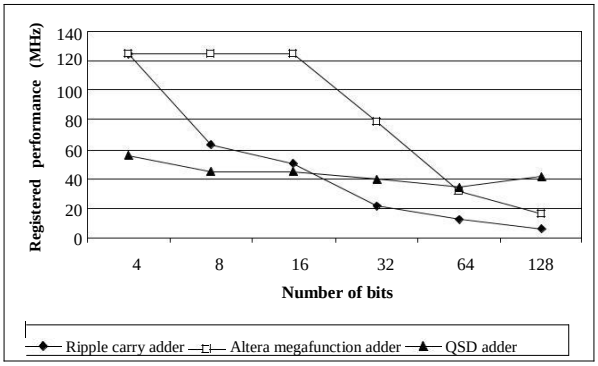

Figure 7. Comparison of the registered performance of all the test adders.

Table 8. Adder registered performance (MHz).

Number of bits	Ripple carry adder	Altera megafunction adder	QSD adder
4	125.00	125.00	55.55
8	62.89	125.00	45.24
16	51.02	125.00	45.04
32	21.97	78.74	39.84
64	12.46	31.94	34.60
128	5.89	16.52	41.49

The registered performance confirms that the QSD adder is superior to other adders for large numbers beyond 64-bits. The delay of the QSD adder is theoretically constant, but due to the limitation of the FPGA implementation and optimization, the delay varies slightly.

The complexity of the adders is measured in terms of the number of logic cells used in each implementation. The results are listed in Figure 8 and Table 9. The complexity comparison shows that all the test adders grow linearly in proportion to the number of bits, which is better than those of the carry lookahead scheme. The considerably high complexity of QSD adder is due to the inefficiency of the 6-variable Boolean expression implementation in the FPGA. A customized VLSI design can reduce the complexity to an affordable level.

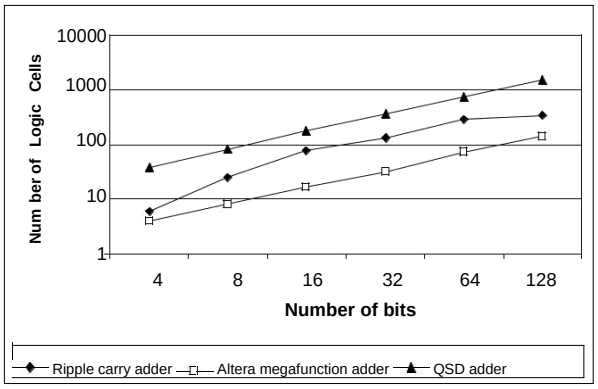

Figure 8. Comparison of the number of logic cells used in all the test adders.

Table 9. The number of logic cells required for implementation.

Number of bits	Ripple carry adder	Altera megafunction adder	QSD adder
4	6	4	37
8	25	8	83
16	78	16	175
32	127	32	361
64	291	72	730
128	333	137	1468

6. Conclusions

The implementation of QSD addition and multiplication are presented. The test confirms the superior performance of the QSD adder implementation over other adders beyond 64-bits due to the carry-free addition scheme. The complexity of the QSD adder is linearly proportional to the number of bits which are of the same order as the simplest adder, the ripple carry adder. This QSD adder can be used as a building block for other arithmetic operations such as multiplication, division, square root, etc. With the QSD addition scheme, some well-known arithmetic algorithms can be directly implemented.

REFERENCES

[1] I. M. Thoidis, D. Soudris, J. M. Fernandez, A. Thanailakis, *"The circuit design of multiple-valuedlogic voltage-mode adders,"* 2001 IEEE

International Symposium on Circuits and Systems, pp 162-165, Vol. 4 , 2001.

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

[3] A. K. Cherri, *"Canonical quaternary arithmeticbased on optical content-addressable memory (CAM)"* Proceedings of the 1996 NationalAerospace and Electronics Conference, pp. 655-661, Vol. 2, 1996.

[4] J. U. Ahmed, A. A. S. Awwal, "*Multiplier designusing RBSD number system",* Proceedings of the1993 National Aerospace and Electronics Conference, pp. 180-184, Vol. 1, 1993.

[5] *FLEX 10K Embedded Programmable Logic Family Data Sheet*, version 4.1, http://www.altera.com,March 2001.