The CSI Journal on
Computer Science and Engineering
Vol. 13, No. 1, 2015
Pages 14-23
Regular Paper

A Reconfigurable, Pipeline Decimal Multiplier

Mahsa Rahmani Mehdi Sedighi

Computer Engineering and Information Technology Departement, Amirkabir University of Technology, Tehran, Iran

Abstract

After addition, multiplication is one of the most frequently used arithmetic operations. As such, numerous combinational or sequential multipliers have been introduced during the last few decades. Most of these multipliers are binary and only a small portion is decimal. However, decimal arithmetic in general and decimal multiplication in particular is gaining prominence. On the other hand, the ever-changing requirements of various applications have imposed a significant demand for reconfigurable designs. In this paper, a reconfigurable 16-digit by 16-digit decimal multiplier is proposed. Since the design is done hierarchically, a building block of a 4-digit by 4-digit combinational multiplier is proposed based on a discussion on the appropriate granularity of the building block. It will be shown that using 4221 coding (instead of BCD) and CSA adders in these blocks will lead to minimum delay. The optimized 4-digit multiplier blocks are devised in a pipeline structure that is inspired by lattice multipliers. For a given maximum latency, Pareto optimal points for a 16-digit by 16-digit decimal multiplier will be obtained. The proposed architecture is reconfigurable in terms of multiplicand and multiplier width. Synthesis results show that the reconfigurebility overhead in terms of area and delay is negligible.

Keywords: Decimal Multiplication, Pipeline Multiplier, 4221/ 5211 Coding, Lattice Multiplier.

1. Introduction

Since decimal arithmetic is preferred for use in financial and commercial applications, hardware support for decimal arithmetic has become a hot topic in computer architecture research [1]. Decimal arithmetic may be implemented both in software and hardware. Software implementations of decimal arithmetic are one or two orders of magnitude slower than the hardware implementations [2]. As such, software implementations are not practical in applications that require a high speed or throughput [2]. This has made hardware decimal arithmetic units a more attractive choice. This is demonstrated by the fact that today's modern computers such as IBM z9, z10 and POWER6 feature dedicated hardware decimal floating point units [3]. These computers are compliant with IEEE 754-2006 standard for decimal number representation.

Multiplication is a very important arithmetic operation that is needed in virtually any application [4]. Generally speaking, the approaches proposed for an implementation of decimal multiplication in hardware can be categorized into two groups: the ones that perform decimal multiplication sequentially and the ones that use a combinational circuit to do that. Many commercial implementations are sequential [5] [6] [7] [8]. These types of implementations are based on iterative algorithms and usually suffer from low performance even though some claim advanced algorithms and improved performance [9].

Combinational decimal multipliers, however, usually employ parallelism to have a better performance compared to sequential multipliers. This comes often as the cost of larger area. The typical decimal multiplier hardware designs proposed in the literature are highly customized to gain minimum area and delay. In other words, there is no flexibility in their structures as they are specific to the applications they were designed for. This rigidity may not be acceptable in applications that require some degree of flexibility such as DSPs [10]. Therefore; researchers in this field have started a trend to present reconfigurable

architectures for decimal multipliers that provide some degree of flexibility with a small overhead in terms of area, delay, and throughput [11]. To continue with this trend, a reconfigurable 16-digit by 16-digit decimal multiplier is proposed in this paper.

The proposed multiplier consists of some building blocks (smaller combinational multiplier blocks and carry-save adders) that are laid out to form a reconfigurable pipeline for a desired multiplier width. The effects of building block granularity as well as using non-BCD coding will be discussed in detail.

This paper is organized as follows: Section 2 contains a discussion on the previous works. The proposed architecture is described in detail in Section 3. The synthesis results are analyzed in Section 4. Section 5 concludes the paper.

2. Related Works

In general, fixed-point multiplication consists of three steps: 1) partial product generation, 2) partial product reduction, and 3) final addition. While these three steps are needed in both binary and decimal multiplications, decimal multiplication is usually more complicated than binary because BCD operations may require some form of correction hardware [12].

As mentioned earlier, decimal multiplication can be done using two general approaches of sequential and combinational. These two approaches will be discussed further in the next two subsections.

2.1. Sequential Decimal Multipliers

In general, sequential multipliers have lower area but higher delay than combinational multipliers [12]. Thus in some cases that low area is more important than delay, sequential multipliers are proposed.

One of the simplest methods to perform decimal multiplication is the iterative multiplication of the multiplier digits to multiplicand digits. The resulting partial products are stored in a register in each cycle. After each iteration, the register is shifted one digit to left and a partial product addition is performed [12]. One of the most important drawbacks of this method is its large critical path delay. To make the multiplier faster, a set of easy multiples is often generated and used. For example, easy multiples such as $2X, 5X, 4X, 8X$ are produced for an input X and other multiples are calculated using the easy multiples [12].

It has been reported that using digit-by-digit multipliers will result in a higher performance [13]. Three reasons have been given for this observation: 1) that this architecture reduces the number of cycles required to perform the multiplications 2) it requires a simpler interconnection network and 3) there is no need to use registers to store the multiples [13]. In this type of multiplier, unsigned binary multiplier units are used. These units take two 4-bit inputs (one decimal digit) and produce an 8-bit output (two decimal digits). It is noteworthy that in this design the output is binary and must be converted to BCD.

In some sequential designs such as [14], the partial product reduction section is divided into two parts that perform the additions in parallel. Even though each part is a sequential circuit but the overall critical path delay is reduced. Furthermore, BCD Carry-Save Adders (CSA) is used instead of carry ripple adders to further reduce the latency. Figure 1 shows this design.

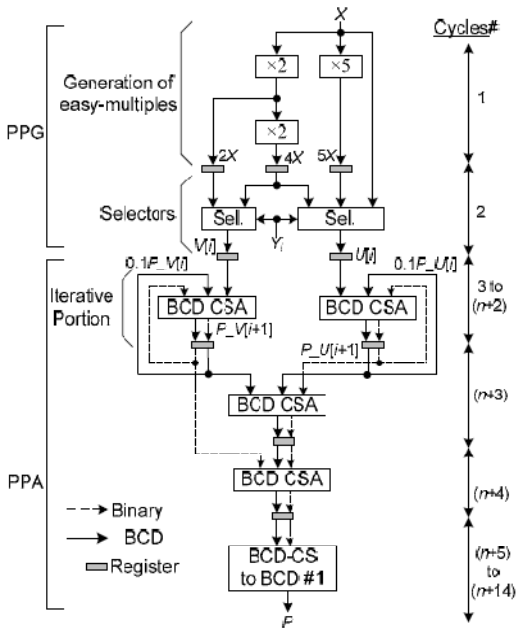

Figure 1. A sequential decimal multiplier proposed in [14]

In another research, the multiplier range has been changed to -5 to 5 and multi-operand redundant adders are employed to further reduce the critical path delay [15]. It has also been shown that changing the coding from BCD to other less common codes such as 4221 or 5211 may reduce the delay [16]. This is because all of the ten combinations of these codes are valid and their correction is less complex compared to 8421.For instance, by using 4221 coding in the partial product generation and reduction steps of figure 1, the overall delay of the multiplier has been reduced considerably [16].

2.2. Combinational Decimal Multipliers

Generally speaking, combinational decimal multipliers are faster than their sequential counterparts but they also take a larger space. This is because they usually use more logic for partial product generation and reduction. For the final addition stage, they often use tree adders. This type of adder typically consists of three stages. In the first stage, the carry propagate and generate signals (p and g, respectively) are produced. If the sum of two digits is 9, p will be 1 and if it is 10 or more, g will be 1. In the second stage, the result will be added with 1 and 0. In the final stage, if carry is propagated the “result +1” is selected; otherwise “result + 0” is selected [17].

Like the sequential multipliers, the notion of using multiple coding schemes has proven beneficial in speeding up combinational multipliers as well. As evidence, reference [18] proposes architecture in which after partial product generation in BCD format, they are converted into their 4221equivalents. Since 4221 addition does not require a complicated correction, this recoding saves the time that would have been otherwise wasted during correction. It is important to note that had the partial products been generated

using 4221 code at the first place, the anticipated delay reduction would have been more pronounced than the reduction reached now. But this idea has not been discussed in [18].

In addition to simpler correction, implementation of partial product generation using non-BCD codes has been reported to be simpler than using BCD [19]. For example, for 2X multiple in 4221 coding, the multiplicand is recoded to 5211 and then shifted to left by1 bit. The result is 2X in 4221 coding. For 5X multiple, the 4221 multiplicand is shifted 3 bits to the left and then recoded to 4221. This is because the internal result after 3-bit shift is valid in 5211 coding. Another important advantage of these codes is that, unlike BCD, they are self-complement.

As mentioned earlier, combinational multipliers are generally faster than the sequential ones and sequential multipliers take a smaller area compared to their combinational counterparts. In this paper, we propose a pipelined structure in which the advantages of both combinational and sequential multipliers are maintained in order to reach Pareto optimal solutions in terms of area and delay. To achieve this, the proposed architecture relies on 4221 coding and recoding as well. The proposed pipeline architecture is designed in such a way that it can take multipliers and multiplicands of different widths whereby featuring a certain degree of reconfigurability.

3. Proposed Architecture

The proposed architecture, which will be elaborated in this section, is designed to perform 16-digit by 16-digit decimal multiplication. However, it also possesses the flexibility to be configured to perform any other combinations of digits up to 64-digit by 4-digit decimal multiplication as well.

The proposed multiplier has a modular design, i.e., it is based on sixteen smaller 4-digit by 4-digit multipliers. The smaller multipliers are put together hierarchically along with appropriate adders to form the larger 16-digit by 16-digit multiplier.

When choosing the appropriate granularity for the building block multiplier (in this case, 4-digit by 4-digit), one has to consider the impact of their choice on the incurred overhead. For instance, if fine-grained building blocks (such as 1-digit by 1-digit multiplier) is used, the reconfigurability overhead due to extra steering logic and longer interconnects will be more significant. But using coarse-grained blocks instead (e.g., 8-digit by 8-digit multiplier) causes considerable wasted resources. Given this fact, our experiments showed that 4-digit by 4-digit multipliers provide a suitable tradeoff. Hence, they were used in the proposed architecture and will be explained in the next subsection.

3.1. The 4-Digit By 4-Digit Multiplier Block

As mentioned before, there is a tradeoff between area and speed when choosing fully combinational vs. sequential multiplier. Since the proposed architecture has a modular design, it is plausible to assume that if a fully combinational multiplier block is put together in a pipelined, i.e., sequential fashion, one can obtain the benefits of both schemes. As such, a fully-customized combinational 4-digit by 4-digit multiplier was designed in this research.

For the reasons explained earlier, the proposed decimal multiplier is based on 4221 and 5211 codes rather than 8421. This decision has an impact on the partial product generation circuitry that will be discussed in the next subsection.

3.1.1 Partial Product Generation in 4-Digit Multiplier

For partial product generation, two set of multiples are chosen as easy multiples, namely$P_1 = \{0, 5X, 10X\}$ and $P_2 = \{0, \pm X, \pm 2X\}$. These multiples were chosen because they have a rather straightforward implementation. Other multiples are generated by combining suitable members of each set. For instance, 3X is generated as $5X + (-2X)$. Table 1 shows the various combinations that form each multiple.

Table 1. The combinations that form multiples

Multiples	P_1	P_2
0	0	0
1X	0	1X
2X	0	2X
3X	5X	-2X
4X	5X	-X
5X	5X	0
6X	5X	1X
7X	5X	2X
8X	10X	-2X
9X	10X	-X

In order to generate the said easy multiples, the BCD input has to first be recoded to either 4211 code or 5211 code, depending on which code would require an easier circuit. The steps taken for each easy multiple generations is as follows:

2X Multiple: Each BCD digit of the multiplicand is recoded into 5211 coding and then shifted one bit to the left. The resulting number is 2X in 4221 coding [19]. Figure 2 illustrates an example for this procedure.

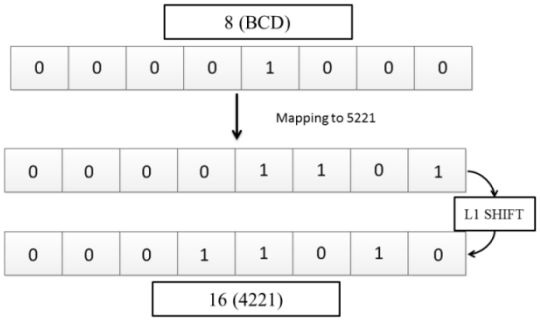

Figure 2. 2X multiple in 4221 coding

5X Multiple: The BCD input digits are recoded to 4221 coding and then shifted three bits to the left. The outcome number is 5X in 5211 coding. Therefore, a single-digit recoder will then be needed to recode from 5211 to 4221 [19]. Figure 3 shows how this multiple is generated.

10X Multiple: 4-bit (1-digit) left shift is performed on 4221 multiplicand. The resultant number is 10X multiple in 4221 coding.

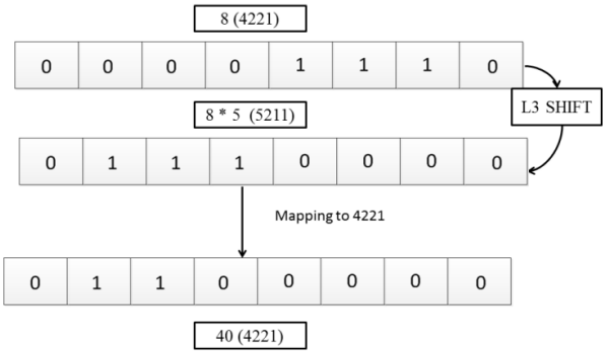

Figure 3. 5X multiple in 4221 coding

−X Multiple and −2X Multiple: Since 4221 code is self-complement, the 10's complement of a 4221 number can be obtained by inverting the bits of the multiplicand and adding 1 to it. Thus, these easy multiples are simply the 10's complement of X and 2X multiples, respectively.

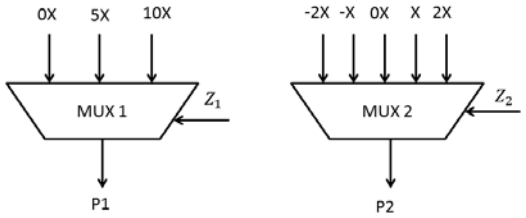

Figure 4. Partial products general structure

Figure 4 shows the two multiplexers that are used to pick the proper easy multiple (P_i) and generate the partial products. In this figure, the control signals of multiplexers are obtained directly from the multiplier bits ($y_3y_2y_1y_0$) using the following logical functions:

$$Z_1(5X) = \overline{((\overline{y_1y_0}) + y_3)} + y_2$$
$$Z_1(10X) = y_3y_2y_1$$
$$Z_1(0) = \overline{((y_2 + y_3) + (y_1y_0))}$$
$$Z_2(-2X) = \overline{((\overline{y_0 \oplus y_2}) + (y_2 \oplus y_3) + \overline{y_1})}$$
$$Z_2(-X) = \overline{((y_2 \oplus y_0) + (y_1 \oplus y_0) + \overline{y_3})}$$
$$Z_2(0) = \overline{((y_3 \oplus y_0) + y_2 + y_1)}$$
$$Z_2(X) = \overline{((\overline{y_0 \oplus y_1}) + (y_1 \oplus y_3) + y_2)}$$
$$Z_2(2X) = \overline{((y_3 \oplus y_0) + y_2 + \overline{y_1})}$$

To conserve area and reduce the delay, the multiplexers were designed using tri-state buffers. Once the single-digit multiplications are done, partial products for 4-digit by 4-digit multiplier are generated. The next step is partial product reduction and will be discussed below.

3.1.2. Partial Product Reduction in 4-Digit Multiplier

Figure 5 shows the partial products positions in the 4-digit multiplier. Each two parallel short lines represent P_1 and P_2 (the multiples generated by the two MUXes in figure 4) coded in 4221. In this figure, each n-digit column is reduced to two digits by means of a cascade of 3-digit: 2-digitCarry-Save Adders (CSA) for each column. Therefore, the columns are reduced in parallel. To avoid long carry propagation delay between columns, the carry produced from each column will not be propagated and will be kept until the final step.

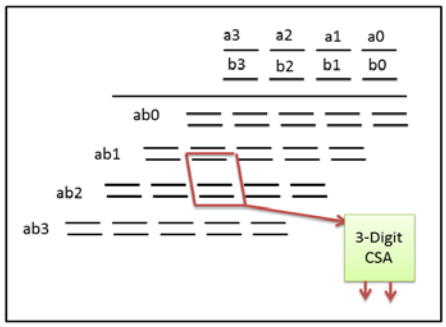

Figure 5. Partial product reduction scheme

The reason for using CSA adders in the proposed architecture is that this kind of adder matches 4221 code in a way that additions are performed without a need for correction. An efficient implementation of a 3-bit: 2-bit CSA adder is shown in figure 6.

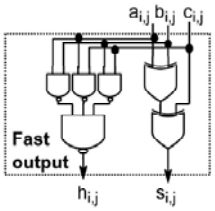

Figure 6. A 3-bit: 2-bit CSA adder [19]

The circuit that makes a 3-digit: 2-digit CSA adder out of 3-bit: 2-bit CSA adder is shown in figure 7.

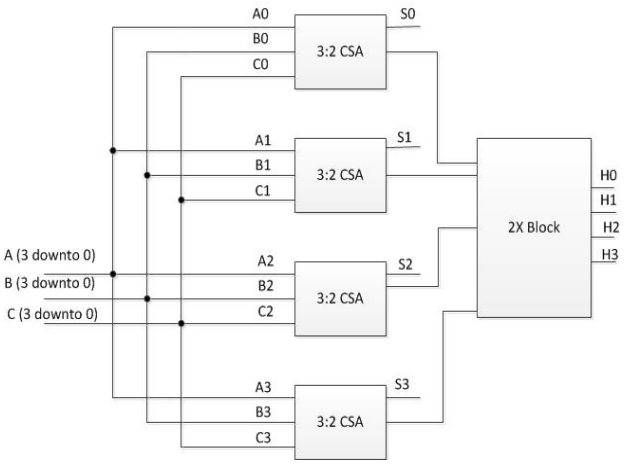

Figure 7. The 3:2 CSA structure for 3 digits

As will be explained later, the pipeline structure of the proposed 16-digit by 16-digit multiplier is inspired by the idea of lattice multipliers. The next section describes this idea and the proposed architecture.

3.2. Lattice Multiplier

Lattice multiplication is a method of multiplying large numbers using a grid. This method breaks the multiplication process into smaller steps. As shown in figure 8, in this method a lattice is formed with each cell divided into two parts diagonally. The multiplier (124576 in the figure) and multiplicand (3857 in the figure) are placed on the top and right side of the lattice, respectively. The most significant digit (MSD) of the multiplicand is on the top right hand side and its least significant digit (LSD) on the bottom right hand side. The content of each cell is the product of the digits that reside on its row and column. The final result from MSD to LSD is formed on the top left side to the bottom right side (480489632 in the figure).

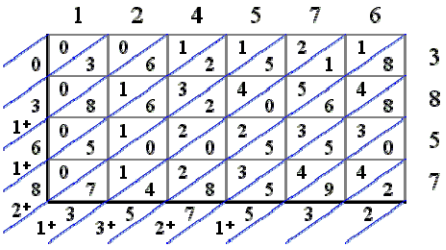

Figure 8. Lattice multiplier

The underlying concept of this research is to use the idea behind a lattice multiplier to perform 16-digit by 16-digit multiplication. To this end, several questions needed to be answered. The first question was how to map the cells of a lattice multiplier on to actual hardware blocks. To find the answer to this question, the lattice multiplier was analyzed with respect to the timing and data dependency of each product generation and the possibility of parallelism among them. Figure 9 illustrates the results. Each short line is a 4-digit partial product in 4221 coding. As this figure shows, 16-digit decimal multiplication requires sixteen 4-digit multiplications that could be executed in any order and then reduced by 24 additions to generate the final product. To explore the possibility of performing these multiplications in parallel, one needs to look at their data dependency and critical path. Figure 9 shows that theoretically the sixteen multiplications can be performed independently. Therefore, one could envision a fully combinational matrix of sixteen of the afore-mentioned 4-digit by 4-digit multiplier blocks connected together to produce sixteen partial products all at once. The problem with such architecture is the fact that the partial product reduction has to be done serially using several adders. So this setup would take a huge space with still a considerable latency.

A better approach is to perform the additions while some of the multiplications are taking place. In other words, strike a balance between the number of resources and the achieved latency by performing the partial product reduction during the multiplications. This notion will lead to the proposed pipelined architecture that delivers a tradeoff between area and latency.

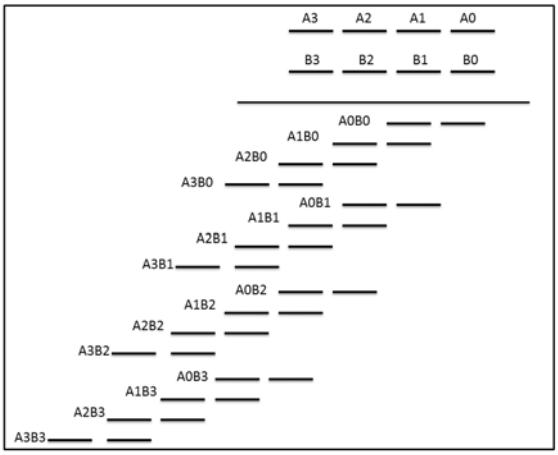

Figure 9. 16-digit multiplication scheme

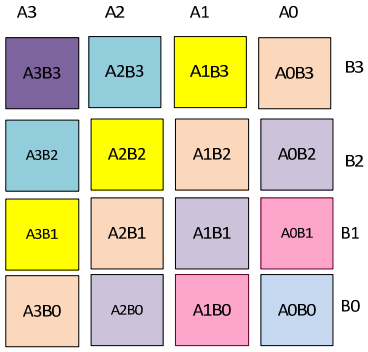

Figure 10. Timing characteristics of 4-digit multiplications in the 16-digit lattice multiplier

Figure 10 depicts the timing characteristics of the sixteen multipliers. The blocks with the same color in this figure have equal weights mathematically and have to be added together during partial product reduction. The longest chain of partial products resides in the main diameter and constitutes the critical path. The duration of this path dictates the minimum latency achieved by the proposed multiplier. The chains on the two sides of the critical path are shorter in length and may accommodate the necessary adders without incurring an extra delay. In other words, an architecture in which the blocks are laid out in a pipeline where combinational adders and 4-digit multipliers are interlaced in a carefully designed sequence provides an optimum structure to gain the least latency with the minimum number of adders and multiplier blocks.

The next question is how many 4-digit multiplier blocks and adders are needed and how should they be laid out to achieve the said minimum latency. These questions are answered in the next section.

3.3. Finding Pareto Optimal Number of 4-Digit Multiplier Blocks and Adders

In order to find the Pareto optimal number of 4-digit decimal multiplier blocks and adders, the extreme case with sixteen dedicated multipliers will have to be revisited. Since the 4-digit multiplier blocks are combinational, they generate their results in one cycle. These results must be added in the next cycles. Assuming no constraint on the number of adders, the partial product reduction will take at least 3 cycles as shown

in figure 11. In this figure, the numbers inside the colored boxes show the cycle in which a partial product is generated. The colors around each box represents which cycle (from T1 to T4) the additions (reductions) are performed. As is clear in the figure, the critical path is located in the middle columns. Overall, this approach takes 16 multiplier blocks, 24 adders, and takes 1 cycle for partial product generation and 3 cycles for reduction (total 4 cycles).

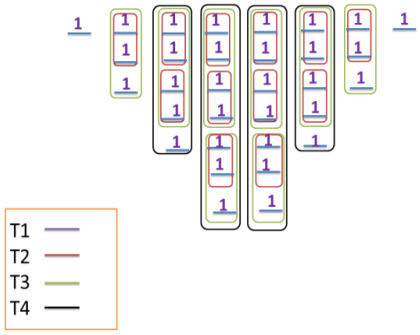

Figure 11. Partial product reduction scheme with no constraint on the number of multipliers and adders

Now that the solution for the unconstrained problem is known, we try to obtain the Pareto optimal solution by reducing the number of resources (multipliers and adders) while maintaining the minimum latency of 4 cycles.

Since the two multipliers that generate A0B0 and A3B3 are farthest from the critical path, it would be conceivable to remove a dedicated multiplier from them. This would reduce the number of multiplier blocks to 14. Figure 12 shows the partial product reduction scheme in this scenario. As can be seen, fewer multipliers and adders are used in this scheme while the latency is still 4 cycles.

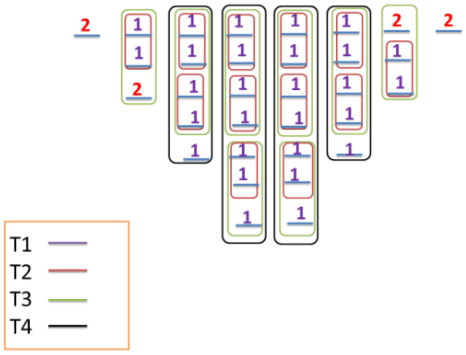

Figure 12. Partial product reduction scheme with 14 multipliers

Continuing this trend indicates that by using 8 multiplier blocks and 10 CSA adders, partial product reduction will still take 3 clock cycles. But removing one more multiplier block will inevitably increase the latency. So 8 multiplier blocks and 10 CSA adders is the Pareto optimal solution for hardware resources in this problem (total latency of 1+3=4 cycles). In this arrangement, the utilization factor for 4-digit multiplier blocks is 100%. Figure 13 shows the partial product scheme in this Pareto optimal point.

The same procedure can be repeated for other latency limits. For instance, five 4-digit multipliers allow for 5-cycle partial product reduction stage. But the utilization factor for multiplier blocks will not be 100% in all cycles. So, some of the hardware resources may be considered underused. With 4 multiplier blocks and 7 adders the latency of 6 cycles will be achieved. It should be noted that in this case the multiplications take 4 cycles (100% utilization factor) and the additions take 5 cycles. But the additions can start at the second cycle. Therefore, for the total latency of 6 cycles, the Pareto optimal point is obtained with 4 multiplier blocks and 7 adders. The pipeline structure for this Pareto optimal point is shown in figure 14.

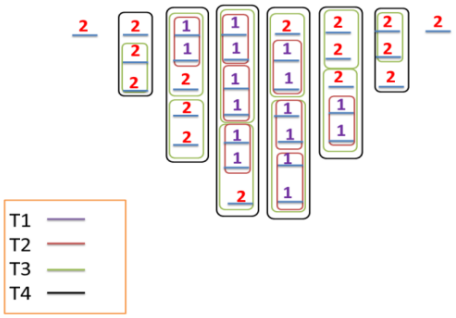

Figure 13. Partial product reduction scheme with 8 multipliers and 10 adders

Table 2. Comparison of two pare to optimal points

# Multipliers	# Adders	Total latency	Utilization
8	10	4	100%
4	7	6	100%

For a better comparison between the two Pareto optimal points discussed above, the results have been summarized in table 2. The latency in the second row is 50% more than the first row. However, the total area of the first row is more than twice the second row. So overall, the second row presents a better tradeoff between the area and delay and will be considered in the remainder of this paper.

3.4. Analytical Lower Bound for the Number of Clock Cycles

The number of partial products generation and reduction cycles can be estimated analytically. Clearly, a lower bound for the partial product generation cycles can be reached by dividing the total number of necessary 4-digit multiplications (in this case, 16) by the number of available 4-digit multipliers ($n_{avail.}$). The partial products that are generated in the last cycle require at least one more cycle to be added to the final product. Therefore, Equation (1) can be used to estimate a lower bound for the number of clock cycles. Depending on the number and arrangement of the adders used in the partial product reduction circuitry, the actual number of clocks may be a lot more than what is predicted by this equation.

$$n_{cycle} \geq \left\lceil \frac{16}{n_{avail.}} \right\rceil + 1 \qquad (1)$$

3.5. Final Addition Step for 16×16 Multiplier

After partial product reduction, the final addition with carry propagation has to be performed. As mentioned before, the

coding for internal operations are 4221. So the intermediate results must be converted back to BCD and then added. This operation is performed in a few levels of logic, as shown in figure 15.

Upon completion, the BCD sum may need correction. The correction unit is implemented as a tabular correction ROM. This approach was chosen to further reduce the delay.

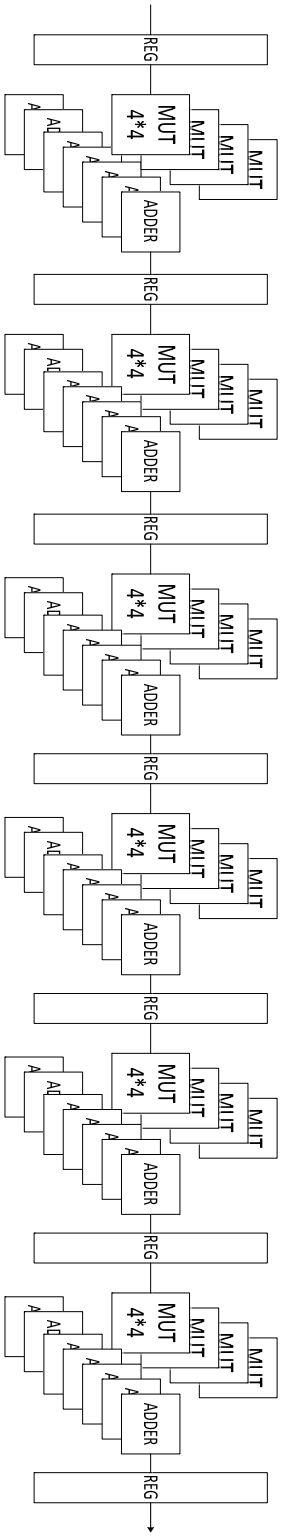

Figure 14. The 16-digit multiplier pipeline structure using 4 4-digit multiplier blocks

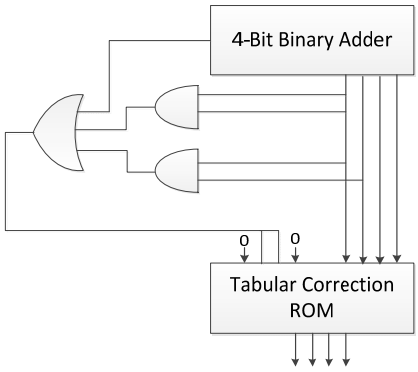

Figure 15. BCD correction scheme

3.6. Reconfigurability of the Proposed Multiplier

With the 4-digit by 4-digit multiplier as a building block, other multiplier widths are possible. For example, 16×4, 16×8, 16×12, 8×8, 8×4, 12×4, 12×8, 12×12, 32×8, 64×4 digits decimal multipliers can all be obtained.

The proposed architecture can complete a 32-digit by 8-digit multiplication using four 4-digit multiplier blocks and 6 (out of 7) CSA adders in 6 cycles. This is very similar to the original 16-digit by 16-digit multiplication and can be in one reconfigurable design. For 32×8 multiplication the structure of figure 16 is recommended. In order to integrate these two designs in one, appropriate steering logic (such as multiplexers and interconnections) should be added among the 4-digit multipliers and CSA adders. Figure 17 shows the steering logic which was added among multipliers and CSA adders to reach this goal.

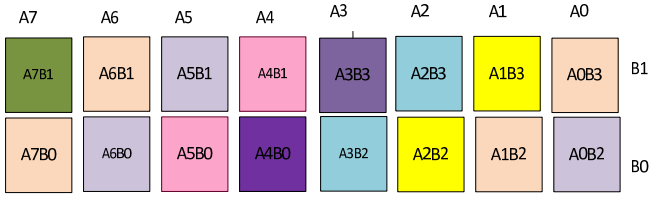

Figure 16. The 32-digit by 8-digitarrangement

A reconfigurable design has inevitably some area and delay overhead. In this design two levels of multiplexers are added among blocks in pipeline stages. So the area and delay of this design is increased. This increase is not too much. The synthesis results show this overhead in next section.

4. Synthesis Results and Comparison

The 4-digit multiplier blocks and the overall 16-digit multiplier have been synthesized using Synopsys Design Compiler and TSMC 90nm CMOS standard cell library under typical conditions (1V, 25℃). The FO4 delay is given as 45ps. The syntheses results are shown in table 3 and comparison with previous works are shown in table 4.

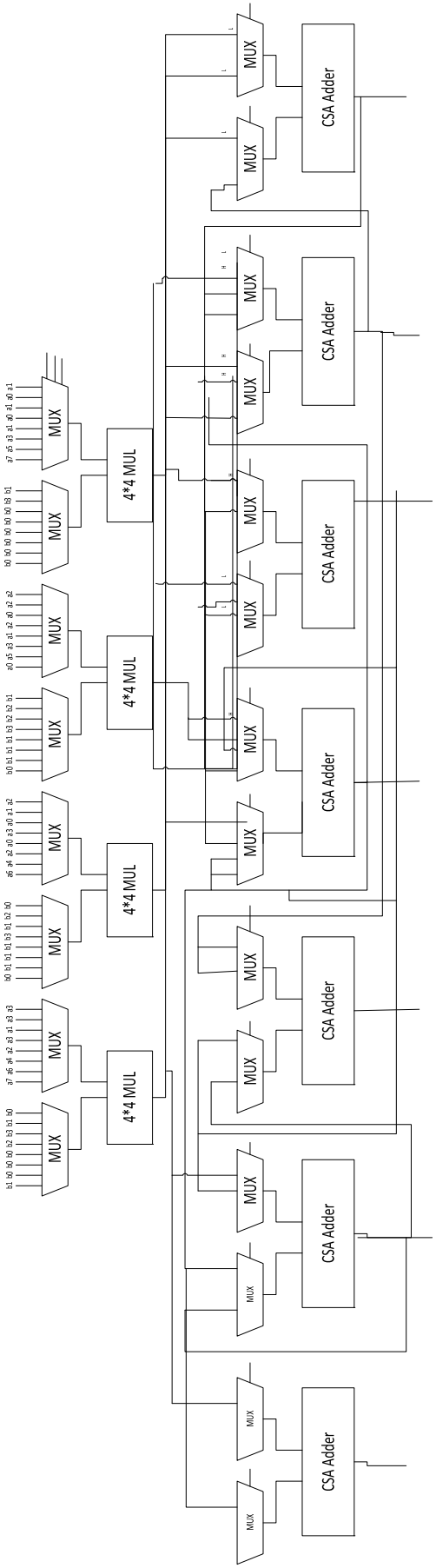

Figure 17. The steering l ogic that was added among multipliers and CSA adders

Table 3. Synthesis results

Design	Area (μm^2)	Delay (ns)
Partial Product Generation (4×4)	2108.33	0.66
Reduction (4×4)	11291	1.17
Multiplier (4×4)	13399.33	1.83
Reduction (16×16)	2557.09	0.78
Multiplier (16×16) Pipeline	89431.08	11.02
Final Addition	7329.77	1.04
Total	95147.33	12.27

As shown in table 3, the delay of 4-digit multiplier block is more than the delay of reduction step. So the 4-digit multiplier delay has to be used to determine the 16×16 pipelined multiplier clock period. However, considering the delay caused by the flip-flops used for pipelining, the 16-digit multiplier delay (11.02ns) is slightly larger than 6 times 1.83ns.

The top half of table 4 includes state of the art purely sequential, non-reconfigurable decimal multipliers found in literature. Like the proposed architecture, [16] employs 4221 coding but [12] and [14] use BCD. As the table clearly shows, the area of the proposed multiplier is smaller than all three designs. But its delay is slightly longer than the 4221-based design and shorter than the conventional BCD designs. This shows the impact of proper coding on the overall delay.

The bottom half of table 4 shows result comparison with fully combinational multipliers that use the same 4221 coding. As anticipated, purely combinational designs are considerably faster than the proposed architecture which is a hybrid of combinational and sequential. However, as also anticipated, the proposed pipeline architecture is 3 times smaller than the best combinational designs found in the literature.

Table 4. Comparison with previous works

Sequential Multipliers				
Design	Delay (FO4)	Ratio	Area (Nand2)	Ratio
Proposed	272.66	1	15875.88	1
[16]	230	0.84	22725	1.43
[14]	318	1.16	17638	1.11
[12]	320	1.17	16000	1.007
Combinational Multipliers				
Design	Delay (FO4)	Ratio	Area (Nand2)	Ratio
[19] SD radix-10	48	0.17	44500	2.80
[19] SD radix-5	46.5	0.17	49700	3.13

The synthesis results for reconfigurable multiplier are shown in table 5. As this table shows, the reconfigurability area overhead is about 10% of the total area and its delay overhead is approximately 2% of the total delay. These overhead are negligible compared to the added value of reconfigurability.

Table 5. Reconfigurability overhead results

	Area (μm^2)	Area (Nand2)	Delay (ns)	Delay (FO4)
Reconfigurability Overhead	10816	1638.78	0.27	6.7
Reconfigurable Multiplier	105963.33	17513.33	6×2.13 =12.78	284

5. Conclusion

In this paper, a reconfigurable 16-digit decimal multiplier was proposed. Since the design was done in a hierarchical manner, a building block of a 4-digit by 4-digit combinational multiplier was designed based on a discussion on the appropriate granularity of the building block. Using 4221 coding (instead of BCD) and CSA adders in these blocks led to minimum delay. The optimized 4-digit multiplier blocks were devised in a pipeline structure that was inspired by lattice multipliers. For a given maximum latency, Pareto optimal points for a 16-digit by 16-digitdecimal multiplier were obtained. To make the design reconfigurable, suitable steering logic was added among resources. Synthesis results show that the reconfigurability overhead in terms of area and delay is negligible. In comparison to pervious sequential multipliers, the 16-digit lattice multiplier takes about 30% smaller area but has 16% more delay than [16] (best case). Compared to previous combinational multipliers, the area of both architectures in [19] are 3 times larger than 16-digit lattice multiplier and their delay is 0.17 times the 16- digit lattice multiplier.

References

[1] M. F. Cowlishaw, "Decimal Floating-Point: Algorism for Computers," *Proceedings of the 16th IEEE Symposium on Computer Arithmetic*, pp. 104-111, 2003.

[2] M. F. Cowlishaw, "The 'Telco' benchmark," World-Wide Web document, Hurley, UK. Available at: http://speleotrove.com/decimal/telco.html, 2002, retrieved on October 15, 2013.

[3] A. Y. Duale, M. H. Decker, H. G. Zipperer, M. Aharoni, and T. J Bohizic, "Decimal Floating-Point in Z9: An Implementation and Testing Perspective," *IBM Journal Research and Development*, vol. 51, no. 1.2, pp. 217–227, 2007.

[4] L. Wang, S. Tsen, M. J. Schulte, and D. Jhalani, "Benchmarks and Performance Analysis of Decimal Floating-Point Applications," *25th IEEE International Conference on Computer Design*, pp. 164-170, 2007.

[5] F. Y. Busaba, C. A. Krygowski, W. H. Li, E. M. Schwarz, and S. R. Carlogh, "The IBM z900 Decimal Arithmetic Unit," *Proceedings of the Asilomar Conference on Signals, Systems and Computers*, vol. 2, pp. 1335-1339, November 2001.

[6] F. Y. Busaba, T. Slegel S. R. Carlogh, C. A. Krygowski, and J. G. Rell, "The Design of the Fixed Point Unit for the z900 Microprocessor," *Proceedings of 14th ACM* Great Lakes Sump. VLSI 2004, pp. 364-367, April 2004.

[7] L. Eisen, and et al., "IBM POWER6 Accelerators: VMX and DFU," IBM Research and Development, vol. 51, no. 6, pp. 663-684, November. 2007.

[8] T. Ohtsuki, and et al., "Apparatus for Decimal Multiplication," US Patent 4, 677, 583, June 1987.

[9] M. Erle, E. Schwarz, and M. Schulte, "Decimal Multiplication with Efficient Partial Product Generation," *Proceeding of 17th IEEE Symposium on Computer Arithmetic*, pp. 21-28, June 2005.

[10] S. Kim, and M. C. Papaefthymiou, "Reconfigurable Low Energy Multiplier for Multimedia System Design," *IEEE Computer Society Workshop on VLSI*, pp. 1-6, April 2000.

[11] S. A. Huack, "The Roles of FPGAs in Programmable Systems," *Proceedings of IEEE*, vol. 86, no. 4, pp. 615-638, 1998.

[12] M. Erle, and M. Schulte, "Decimal Multiplication via Carry Save Addition," *Proceedings IEEE International Conference on Application-Specific Systems, Architecture and Processors*, pp. 348-358, Jun 2003.

[13] G. Jaberipour, and A. Kaivani, "Binary Coded Decimal Digit Multipliers," *IET Computer and Techniques*, vol. 1, no. 4, pp. 377- 381, July 2007.

[14] A. Kaivani, A. Li Chen, and SB. Ko, "High-Frequency Sequential Decimal Multiplier," *IEEE International Symposium on Circuits and Systems (ISCAS 2012)*, pp. 3045-3048, May 2012.

[15] L. Han, A. Kaivani, and SB. Ko, "Area Efficient Sequential Decimal Fixed-Point Multiplier," *Springer Journal of Signal Processing Systems*, vol. 75, no. 1, pp. 39-46, 2014.

[16] A. Kaivani, L. Han, and SB. Ko, "Improved Design of High-Frequency Sequential Decimal Multipliers," Electronics Letters, vol. 50, Issue 7, pp. 558- 560, March 2014.

[17] T. Lang, and A. Nannarelli, "A Radix-10 Combinational Multiplier," *40th Asilomar Conference on Signals, Systems and Computers*, pp. 313-317, November. 2006.

[18] A. Vazquez, E. Antelo, and P. Montuschi, "A New Family of High Performance Parallel Decimal Multipliers," *18th IEEE Symposium on Computer Arithmetic*, pp. 902-916, Jun 2007.

[19] A. Vazquez, E. Antelo, and P. Montuschi, "Improved Design of High Performance Parallel Multiplier," *IEEE*

Transactions on Computers, vol. 59, no. 5, pp. 1902-1914, 2010.

Mahsa Rahmani received her B. S. degree in Computer Engineering from University of Guilin in 2012. She is currently pursuing her M. S. degree in Computer Systems Architecture at Computer Engineering and Information Technology Department, Amirkabir University of Technology (Tehran Polytechnic). Her research interests include decimal arithmetic, reconfigurable design and VLSI design.
E-mail: rahmani.m93@aut.ac.ir

Mehdi Sedighi received his B.S. in Electrical and Computer Engineering from Sharif University of Technology in 1990 and his M.S. and Ph.D. in the same field from University of Colorado at Boulder in 1994 and 1998, respectively. Since late 2001, he has been with the Department of Computer Engineering and Information Technology at Amirkabir University of Technology where he is currently an associate professor. His research interests include VLSI design, synthesis of arithmetic circuits, embedded systems and quantum computing.
E-mail: msedighi@aut.ac.ir

Paper Handling Data:

Submitted: 03.08.2016
Received in revised form: 12.10.2016
Accepted: 04.11.2016
Corresponding author: Dr. Mehdi Sadeghi,
Computer Engineering and Information Technology Department, Amirkabir University of Technology, Tehran, Iran.