

# A Faster Distributed Arithmetic Architecture for FPGAs

Radhika S. Grover, Weijia Shang and Qiang Li

Department of Computer Engineering, Santa Clara University, CA  
rgrover@scudc.scu.edu {wshang, qli}@sunrise.scu.edu

## ABSTRACT

Distributed Arithmetic (DA) is an important technique to implement digital signal processing (DSP) functions in FPGAs. However, traditional lookup table (LUT) based DA architectures contain one or more carry propagation chains in the critical path that dictates the fastest time at which an entire design can run. In this paper, we describe a novel technique that can reduce or eliminate the carry-propagate chain from the critical path in LUT based DA architectures on FPGAs. In the proposed scheme, the individual bits of a word do not have to be processed as a unit. Instead, the current iteration can start as soon as the least significant bit (LSB) of the previous iteration is available, without waiting for the entire word from the previous iteration to be fully computed. This technique has great potential in speeding up DSP applications based on DA. Designs are described for serial and parallel DALUT and accumulator structures in which an  $n$ -bit carry chain, where  $n$  is the word length, is broken into smaller  $r$ -bit chains,  $1 \leq r < n$ . A cost-performance analysis of the designs is presented. The analysis shows that the designs proposed in this paper have a lower cost-performance ratio (indicating better performance) than traditional DA designs. We also show that the 8-bit ( $r = 8$ ) designs offer a good compromise between cost and performance. The implementation is on a Xilinx chip XC4028XL-3-BG256 using Xilinx Foundation tools v 3.1i. The results show that the proposed designs can achieve speedup by a factor of at least 1.5 over traditional DA designs in some cases.

## Keywords

Distributed arithmetic, DALUT, XC4000, carry propagation, cost-performance analysis

## 1. INTRODUCTION

With improvements in capacity and performance and a decrease in cost, FPGAs have become a viable solution for making custom chips and programmable DSP devices. By mapping algorithms to FPGAs significant performance benefits can be achieved. *Distributed Arithmetic (DA)* [1,2] provides an approach for multiplier-less implementation of DSP systems. It is an algorithm that can perform multiplication with *lookup table (LUT)* based schemes (also called *DALUT*). DA specifically targets the sum of products (also referred to as the vector dot product) computation that is found in many of the important DSP filtering and frequency transforming functions. Combined with Xilinx

FPGA lookup table architecture, the DA algorithm was shown to produce very efficient filter designs [3].

Techniques for speeding up DALUT based applications have appeared in several papers, such as using matrix factorization methods and exploiting symmetry in matrices to speed up the computation procedure by reducing the number of addition and subtraction operations required. Image and signal processing techniques using DA have been discussed in [4,5,6,12] and others.

In this paper we present a novel architecture for speeding up DALUT applications. The distinguishing feature of the new design is that, unlike in traditional DALUT architectures, the carry chain delay is reduced or eliminated from the critical path. In the proposed scheme, the individual bits of a word do not have to be processed as a unit. Instead, the current iteration can start as soon as the LSB of the previous iteration is available, without waiting for the whole word from the previous computation. Designs in which an  $n$ -bit carry chain, where  $n$  is the word length, is broken into smaller  $r$ -bit chains,  $1 \leq r < n$ , are described in this paper. Techniques to eliminate carry chain delay have appeared in other papers [7,8,9]. In this paper, we apply a similar idea to DALUT-based structures on FPGAs to gain performance benefits. New designs for parallel and serial DALUT structures are proposed. A cost-performance analysis of the designs shows that the proposed designs with no carry chain delay have a lower cost-performance ratio than traditional DALUT designs as well as designs with  $r$ -bit carry chains. Also, it is shown that designs with 8-bit carry chains provide a good compromise between performance and cost. The designs are implemented on a Xilinx XC4028XL-3-BG256 using Xilinx Foundation tools v 3.1i. The results show that the new designs achieve speedup by a factor of at least 1.5 over traditional designs. The technique is general enough to be applicable to other families of chips as well.

The rest of the paper is organized as follows: In Section 2 a brief overview of traditional DA background is presented. Section 3 describes the new designs for parallel and serial DA and gives a cost-performance analysis of these designs. In section 4 the results of comparing the traditional and proposed designs are presented. Concluding remarks are given in section 5.

## 2. TRADITIONAL DALUT ARCHITECTURE

A brief overview of traditional DALUT architecture is presented in this section. Let the input data and the transformed data be represented by two vectors  $X$  and  $Y$  of size  $N$ , respectively. Then  $Y$  can be written as follows:

$$Y = \sum_{k=0}^{N-1} A_k x_k \quad (1)$$

where  $A_k$  are constant coefficients.  $x_k$  is written in weighted format as shown in the equation (2).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

FPGA '02, February 24-26, 2002, Monterey, California, USA.  
Copyright 2002 ACM 1-58113-452-5/02/0002...\$5.00.

$$x_k = -x_{k,n-1} + \sum_{m=1}^{n-1} x_{k,n-1-m} 2^{-m} \quad (2)$$

where,  $x_{k,m}$  is the  $m$ th bit of  $x_k$  (which can be a zero or one),  $x_{k,n-1}$  is the sign bit and  $n$  is the word size. Substituting equation (2) in (1),

$$\begin{aligned} Y &= \sum_{k=0}^{N-1} A_k \left( -x_{k,n-1} + \sum_{m=1}^{n-1} x_{k,n-1-m} 2^{-m} \right) \\ &= -\sum_{k=0}^{N-1} A_k x_{k,n-1} + \sum_{m=1}^{n-1} \left( \sum_{k=0}^{N-1} A_k x_{k,n-1-m} \right) 2^{-m} \end{aligned}$$

Define,

$$Z_{n-1-m} = \sum_{k=0}^{N-1} A_k x_{k,n-1-m} \quad (m \neq 0)$$

and,

$$Z_{n-1} = -\sum_{k=0}^{N-1} A_k x_{k,n-1} \quad (m = 0)$$

The output result is given by,

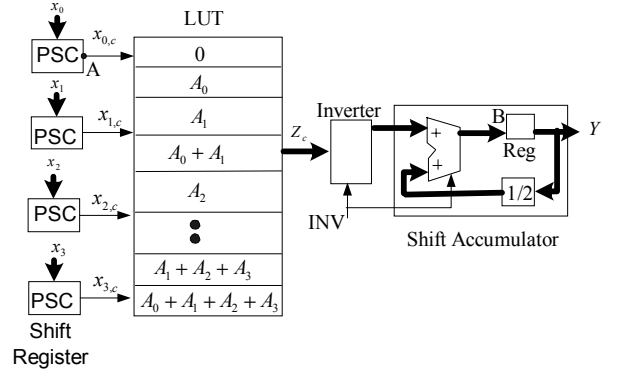
$$Y = \sum_{m=0}^{n-1} Z_{n-1-m} 2^{-m} \quad (3)$$

Let  $c = n-1-m$  for convenience. Since  $Z_c$  is a function of  $x_{k,c}$ ,  $k = 0, N-1$ , it has only  $2^N$  possible values which can be precomputed and stored in a LUT. Figure 1 shows the traditional LUT-based serial DA design for a *four-product* ( $N = 4$ ) multiplier-accumulator (MAC), which multiplies four pairs of numbers and sums the results. The LUT data is composed of all partial sums of the coefficients ( $A_0, A_1, A_2$  and  $A_3$ ). The LSB (output from each parallel to serial converter PSC register) of the four data samples ( $x_0, x_1, x_2$  and  $x_3$ ) addresses the LUT. The output of the LUT depends on the address formed by the combinations of these input bits. For example, if all four data bits are 1, then the output from the LUT is the sum of all four coefficients. Further details on DA can be found in [2,3]. In the next section, new designs for serial and parallel DALUT structures are presented.

### 3. PROPOSED DESIGNS

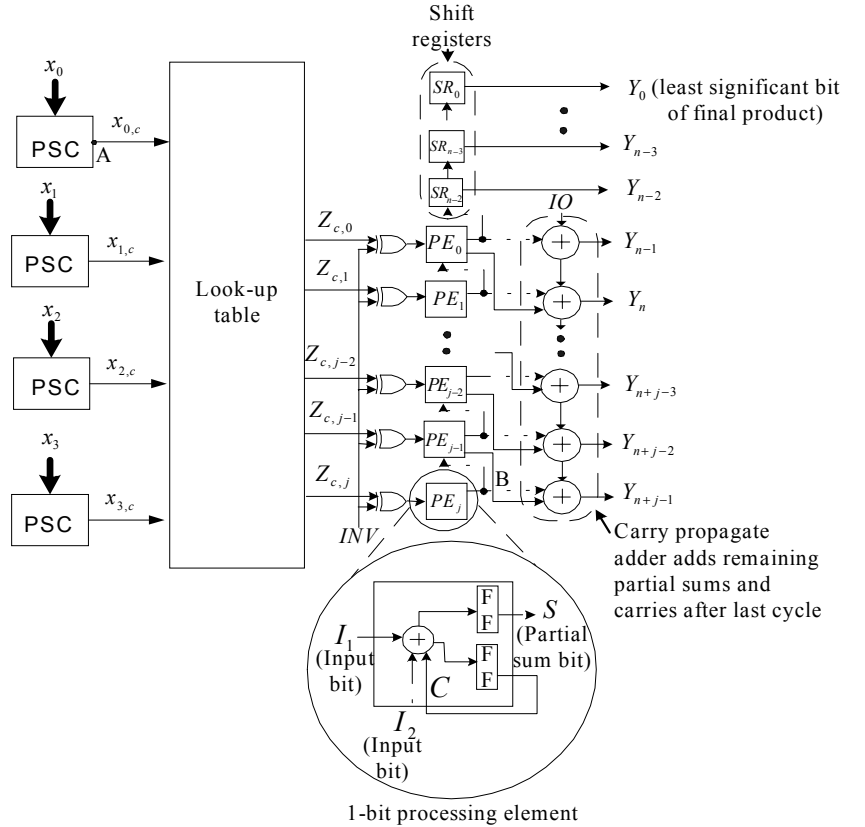
In Figure 1, the carry chain is present in the shift-accumulator. Figure 2 shows how the carry chain can be removed from the critical path in this design. The critical path is the delay between points A and B in both designs.

The accumulator is composed of a series of *bit-level processing elements* (PEs). Each bit-level PE takes three input bits that are added to produce a sum bit and a carry bit that are registered in two flip-flops (FF) at each clock.



**Figure 1. Traditional LUT-based serial DA for a four-product MAC**

Figure 2 shows a PE configured as a serial adder. The output of the LUT is connected to input  $I_1$  of the PEs. The partial sum bit in the  $j$ th PE is connected to the input  $I_2$  of the  $(j-1)$ th PE. The carry bit  $C$ , generated from adding three bits in each PE, is not propagated across the PEs; instead, it is registered within the PE at each clock, and added to the input bits in the next clock. One bit of final product is shifted out serially from the bit-level shift-accumulator at each clock into a set of shift registers. After the  $n$ th clock, (where  $n$  is the word size), the *residual* partial sum and carry bits are summed in a carry-propagate adder to form the  $n-1$  higher-order bits of the final product which has  $2n-1$  bits.



**Figure 2. Proposed (1-bit) serial DA one-LUT design for a four-product MAC.**

The residual partial sum and carry bits are the bits left over in the PEs after  $n$  clocks, when no new outputs from the LUT are required to be added. When the sign bits arrive, a subtraction instead of an addition is done in the shift-accumulator. The bits of  $Z_c$  are inverted in the last cycle by setting  $INV$  to '1' (corresponding to multiplication by the sign bit); however, the addition of a '1' is delayed (in order to move the carry-propagate adder out of the critical path), and takes place, after the last clock cycle, by setting the  $IO$  bit to a '1'. The  $IO$  bit is called a *compensating one*, since it adjusts the final result to the correct value by compensating for the addition of '1' that was omitted earlier in the computation. The resulting design, in Figure 2, takes the same number of clocks as the design in Figure 1, but runs much faster because the carry propagate adder has been moved out of the critical path. The clock time of design in Figure 1 has to be long enough to accommodate the carry chain; the clock time of

design in Figure 2 is the time needed to generate the partial carry and sum. Figure 3 shows how this can be extended to the  $r$ -bit case by using PEs with carry chain lengths of  $r$ -bits. The last cell in the  $r$ -bit PE sums 4 bits and generates two carries –  $C$  (lower order carry) and  $C1$  (higher order carry). All the other cells are similar to the bit-level PE. The carries,  $C$ , between the cells inside the  $r$ -bit PE are not registered; instead, they propagate across the  $r$ -bit PE, with the exception of the two carries in the last cell. Thus, the carry chain is broken across PEs. Note however, that the last PE may have less than  $r$  cells, depending on whether  $n$  is a multiple of  $r$ . The above scheme is extended to the multiple-product case. As the number of coefficients increase, the size of the LUT grows exponentially.

A large LUT is avoided by partitioning the circuit into smaller groups and combining the LUT outputs with adders, as shown in Figure 4. The adders are less costly than the larger LUT. However,

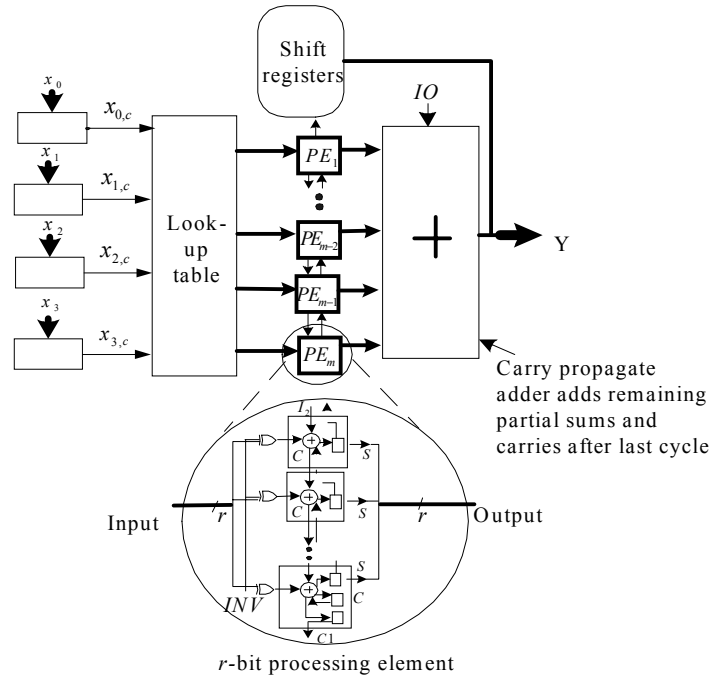


Figure 3. Proposed ( $r$ -bit) serial DA one-LUT design for a four-product MAC.

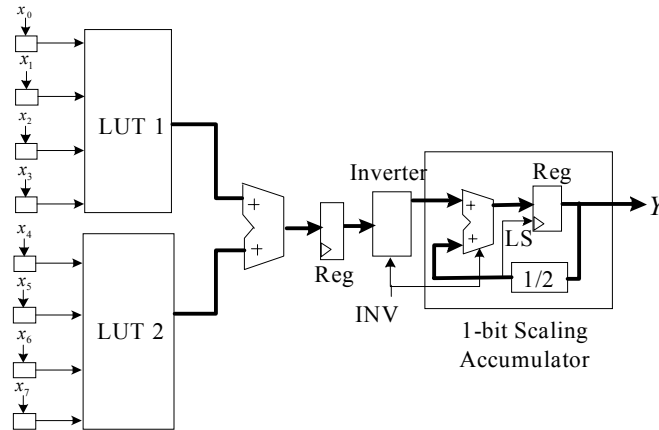


Figure 4. Traditional serial DA two LUT-based design for an eight-product MAC.

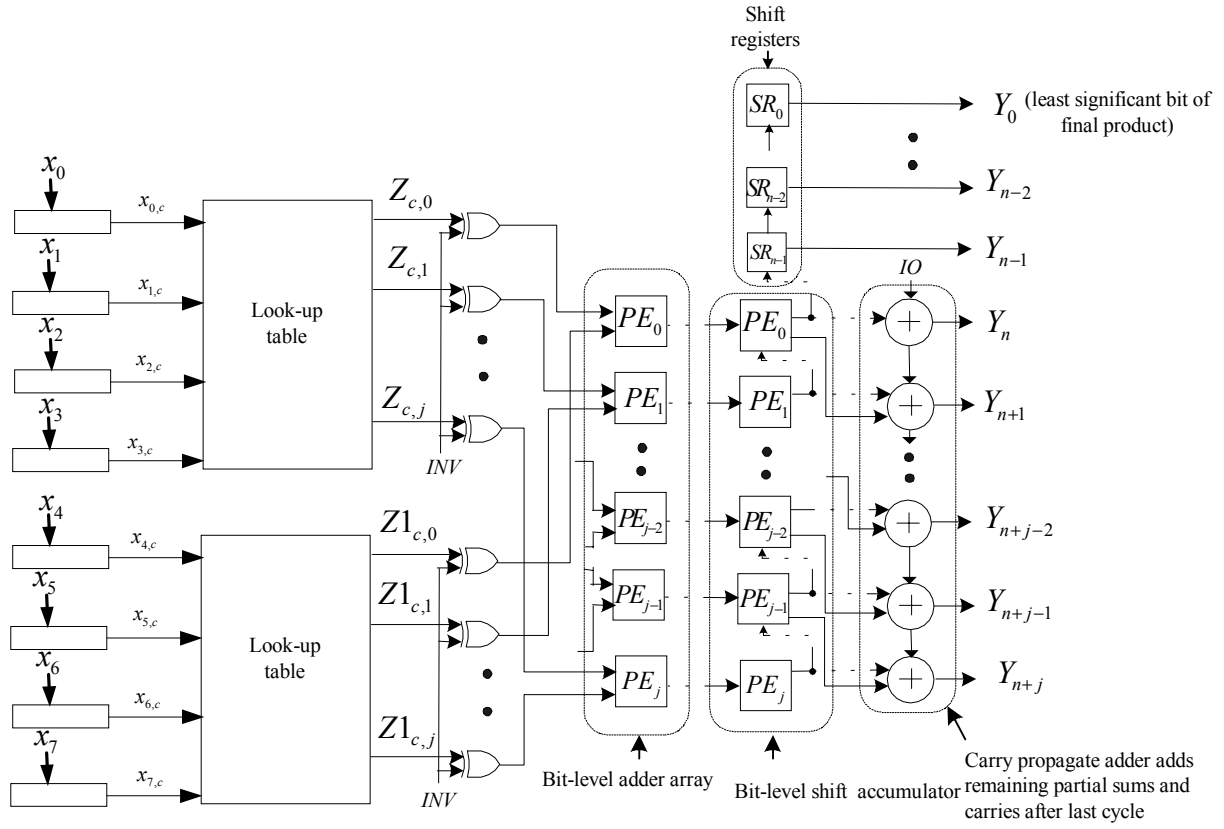


Figure 5. Proposed serial DA two-LUT design for an eight product MAC.

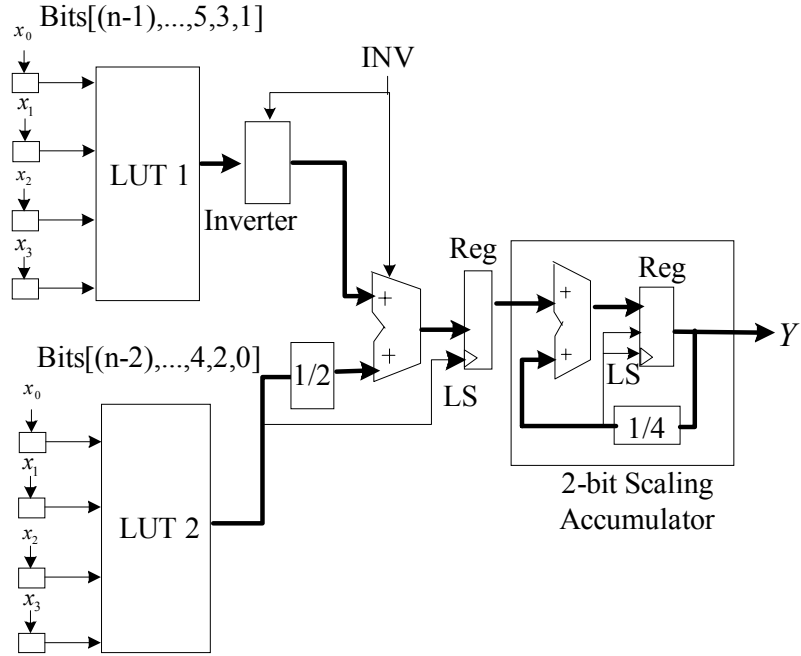


Figure 6. Traditional 2-bit parallel DA for a four product MAC.

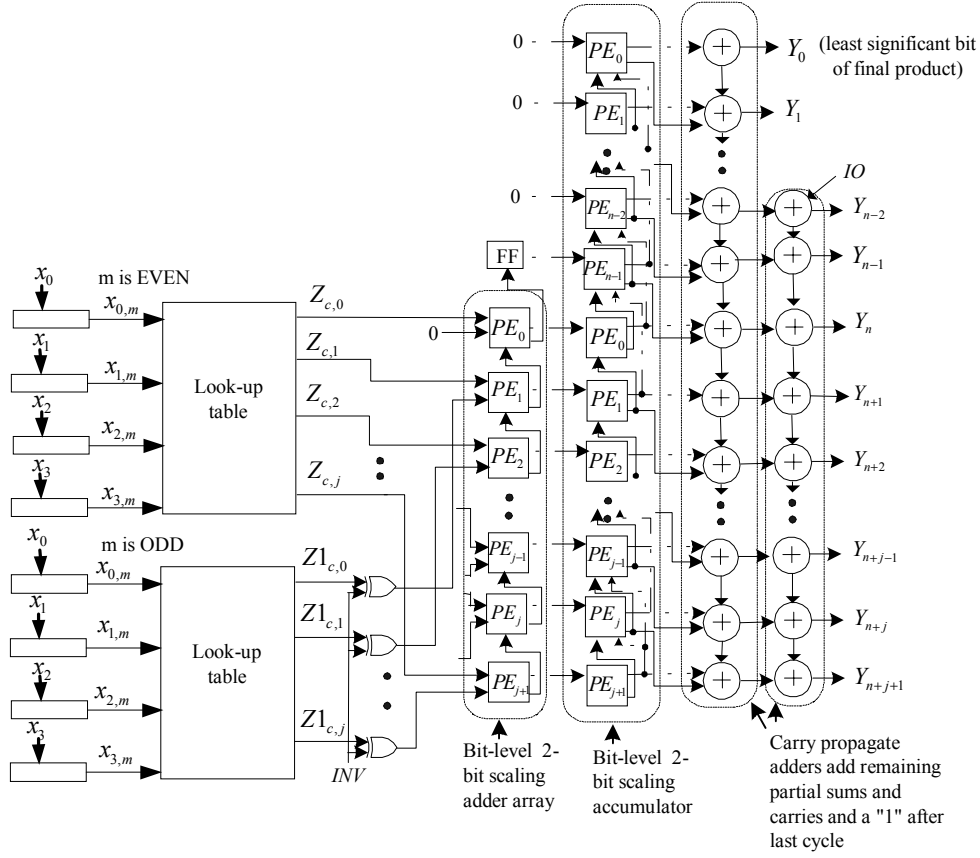


Figure 7. Proposed 2-bit parallel DA for a four product MAC.

each adder introduces a carry chain delay in the critical path. Figure 5 shows how to remove the carry chain for an eight-product MAC, implemented with two LUTs, by using a simple bit-level adder array. The adder array is composed of bit-level PEs shown in Figure 2. The partial sum output bit of each PE in the bit-level adder array is connected to a PE input in the bit-level shift-accumulator. The partial sums and carries are stored separately in this adder array, and the carries are not propagated across the PEs of the adder array. When the sign bits are input to the LUT, the outputs of both LUTs must be inverted and a '1' added to each of the outputs. Instead, as in the one-LUT case, the addition of the two '1's is delayed and takes place, after the last clock, in the carry-propagate adder. The design in Figure 5 takes one *extra* clock compared to the traditional serial DA for two-LUT case so that the residual carries in the bit-level adder array are added in the bit-level shift-accumulator. After the last clock, the partial sums and carries in the shift-accumulator are added in the carry-propagate adder to form the final product. Again, the clock cycle time is reduced since the carry chains are removed from the critical path, at the cost of a single extra clock. To compensate for the two '1's required to be added earlier, a suitably weighted compensating one, (i.e., in this case a '10' string) is added in the carry-propagate adder at the appropriate position (by setting *IO* to a '1' after the last clock as in Figure 5). The above idea can be extended similarly for serial DA many-product cases and parallel DA. Figure 6 shows the traditional LUT-based scheme for 2-bit parallel DA. Two LUTs are needed, one for

the even order coefficients and the other for the odd order coefficients. Figure 7 shows the scheme for proposed 2-bit parallel DA for a four product MAC. The bit-level 2-bit scaling adder array and accumulator are used here, and there is no carry propagation across these components. The carry out  $C$  from the  $j$ th PE is registered before it is sent to the  $(j-1)$ th PE, where it is added to input bits of the same weight. Unlike in SDA designs, the carryout in the bit-level PDA PE is not fed back into the same PE where it was generated. The bit-level 2-bit scaling accumulator shifts two bits at a time. As in the previous designs, carry propagation takes place after the last clock in the two carry propagate adders shown in Figure 7. The *IO* bit is set to a '1' after the last clock to compensate for the '1' that was required to be added earlier. The  $r$ -bit designs can be obtained directly by replacing  $r$  bit-level PEs with one  $r$ -bit PE in Figures 5 and 7.

### 3.1 Cost and performance analysis

Gate count and gate-delay unit models used in VLSI designs are not useful for evaluating costs and performance of FPGA designs. In our study, instead of gate counts, cost is measured as the number of configurable logic blocks (CLBs) used. Performance is calculated as the inverse of the computation time (time to complete the DA operation). The computation time is estimated using a simple model that estimates the total delay

**Table 1. CLB counts and logic delays in various components, as a function of word size  $n$ , on the XC4000 series.**

Component	CLB count ( $C$ )	Delays ( $T$ )
$n$ -bit full adder (non-registered)	$C_{FA}(n) = \frac{n}{2} + 2$	$T_{FA}(n) = T_{INCY} + \frac{(n-2)}{2}T_{BYP} + T_{SUM}$
$n$ -bit shift accumulator (registered)	$C_{ACC}(n) = \frac{n}{2} + 2$	$T_{ACC}(n) = T_{INCY} + \frac{(n-2)}{2}T_{BYP} + T_{SUMC} + T_{ICK}$
$n$ -bit inverter	$C_{INV}(n) = \frac{n}{2}$	$T_{INV} = T_{ILO}$
$n$ -bit ROM-based LUT (depth $\leq 16$ )	$C_{LUT}(n) = \frac{n}{2}$	$T_{LUT} = T_{ILO}$
$n$ -bit Shift register	$C_{SR}(n) = C_{PSC}(n) = \frac{n}{2}$	$T_{PSC} = T_{CKO}$
1-bit Serial adder	1	$T_{SA} = T_{ICK}$

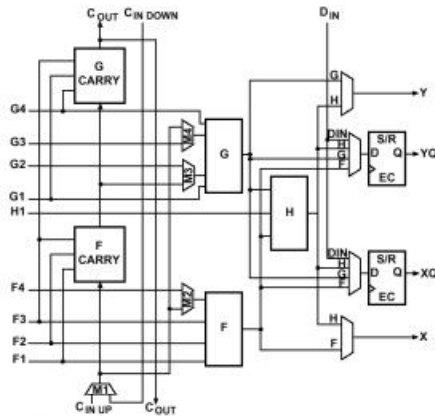
in the critical path as a sum of the *logic* and *routing* delays. The number of logic levels in the design determines the logic delay, and the routing delay is estimated to be 50-100% of the logic delay. The actual values of the computation times are then obtained from Xilinx timing-simulation software instead of from gate-delay models used for VLSI designs. The cost-performance ratio (CT) is calculated as cost multiplied by the computation time. In making comparisons, a DA design with a lower CT ratio has better performance. Table 1 shows the CLB count and logic delays for various components in the traditional and proposed designs as a function of the word size  $n$ , on the XC4000 chip. In the XC4000 series ripple-carry outputs are routed between CLBs on high-speed dedicated paths (Figure 8). This dedicated carry logic provides fast arithmetic carry capability for high-speed arithmetic functions. In adders using XC4000 dedicated carry logic delay estimation is possible [10]. The carry path in an adder uses dedicated interconnects between CLBs. These interconnects introduce a fixed delay, even when the carry passes from one CLB column to the next

at the top or bottom of the array. This permits the routing delay to be incorporated into the CLB specifications published in the data book. Consequently, the propagation delay through an adder can be calculated directly from the data book specifications. The values of the delay parameters for the XC4000 (-3) family (shown in Table 2) are obtained from the Xilinx data sheet [11]. The total computation time to calculate the result is given by the product of the number of clocks and the clock cycle time. In Figure 1, the critical path is the delay through points A and B. The total delay is the sum of the logic delays from the output of the shift register, through the LUT, inverter and the  $(n+2)$ -bit shift accumulator including the routing delay between these components. The cost is the sum of the number of CLBs in the shift registers, LUT, inverter and the  $(n+2)$ -bit (signed) shift accumulator.

The computation time of the one LUT-based traditional design (Figure 1) is

$$= (n+1) * (T_{PSC} + T_{LUT} + T_{INV} + T_{ACC}(n+2) + T_R)$$

$$= (n+1) * (T_{CKO} + 2T_{ILO} + T_{INCY} + \frac{n}{2}T_{BYP} + T_{SUMC} + T_{ICK} + T_R)$$



**Figure 8. XC4000 dedicated carry logic.**

**Table 2. Delay parameters for XC4000 (-3) series**

$T_{ILO}$	Delay from F/G inputs to X/Y outputs	1.6 ns
$T_{INCY}$	Fast carry logic (F1/F3 to Cout)	2.0 ns
$T_{SUM}$	Fast carry logic (Cin through function generators to X/Y outputs)	2.8 ns
$T_{BYP}$	Cin to Cout (bypass function generators)	0.26 ns
$T_{CKO}$	Clock to flip-flop output Q	2.1 ns
$T_{ICK}$	Setup time (F/G inputs)	1.1 ns
$T_{SUMC} + T_{ICK}$	Fast carry logic and setup time	2.93 ns <sup>1</sup>

<sup>1</sup> Obtained from the Xilinx timing analyzer.

where,  $T_R$  is the total routing delay between components in the critical path.

The cost of the traditional design (Figure 1) is

$$= C_{LUT}(n+2) + C_{ACC}(n+2) + C_{SR}(n) + C_{PSC}(n) * 4$$

Substituting the values of the parameters from Table 1 and Table 2 gives the total computation time to be  $(n+1)(10.23 + 0.13n + T_R)$  and the cost to be  $3.5n + 4$ .

Therefore, the cost-performance metric is of the traditional design (Figure 1) is

$$= (3.5n + 4)(n+1)(10.23 + 0.13n + T_R). \quad (4)$$

The computation time of the one LUT-based proposed design (Figure 2) is  $= (n+1) * (T_{PSC} + T_{LUT} + T_{INV} + T_{SA} + T_R) + T_{FA}(n)$

$$= (n+1) * (T_{CKO} + 2T_{ILO} + T_{ICK} + T_R) + T_{INCY} + \frac{(n-2)}{2} T_{BYP}$$

$$+ T_{SUM}$$

The cost of the proposed design (Figure 2) is

$$= C_{LUT}(n+2) + (n+2) * C_{SA}(1) + C_{INV}(n+2) + C_{SR}(n) + C_{PSC}(n) * 4$$

Substituting the values of the parameters from Table 1 and Table 2 gives the total computation time to be  $6.53n + 10.94 + (n+1)T_R$  and the cost to be  $5n + 6$ .

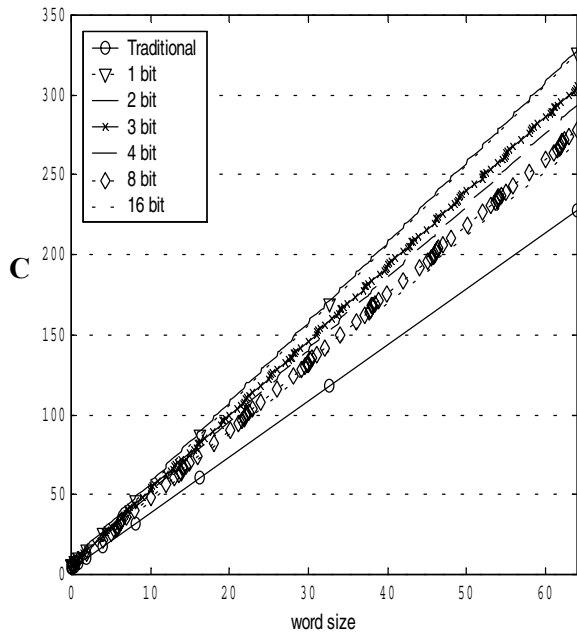
The cost-performance metric of the one LUT-based design (Figure 2) is

$$= (5n+6)(6.53n+10.94+(n+1)T_R) \quad (5)$$

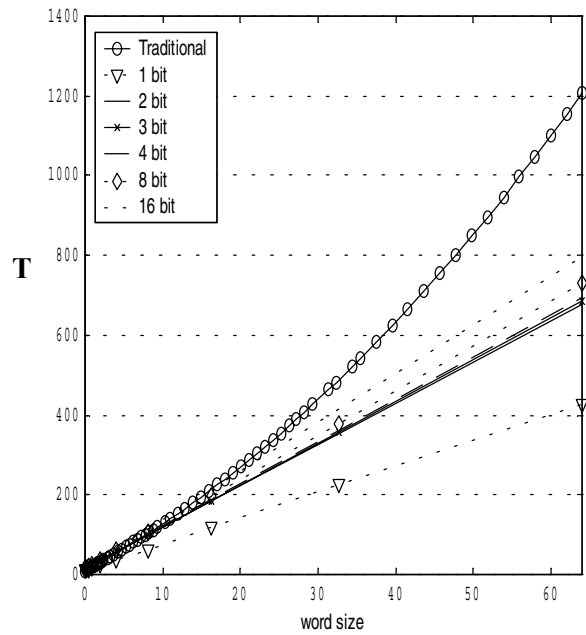
The cost and computation time of these designs with  $r$ -bit wide carry chains, where  $1 < r < n$ , can be calculated similarly.

The cost-performance metrics are plotted for different values of  $r$ , in the range  $1 \leq r \leq n$ , for different values of  $n$  and different routing delays in Figure 9. The traditional design ( $r = n$ ) has the lowest cost (Figure 9 (a)), but a higher computation time (Figure 9 (b)) as compared to the proposed designs. The proposed design ( $r = 1$ ) has the fastest computation time and the best cost-performance metric (Figures 9(c) and 9(d)) as compared to the others. The design with  $r = 8$  provides a good compromise between cost and performance. Figures 9 (c) and (d) show the variation in the cost-performance curves with zero routing delay, and routing delay equal to the logic delay, respectively. The performance improvement in designs with  $r < n$  becomes more pronounced as the word size increases. The analysis given above can be similarly extended to other cases such as PDA and SDA with more than one lookup table. For lack of space we do not present other cases, but in general, the proposed designs outperform the traditional designs.

The bit-level designs are implemented on an XC4000xl chip, and compared with traditional DA designs. The implementation results are presented in the next section. It will be shown that the results of the simulation are consistent with the above discussion.

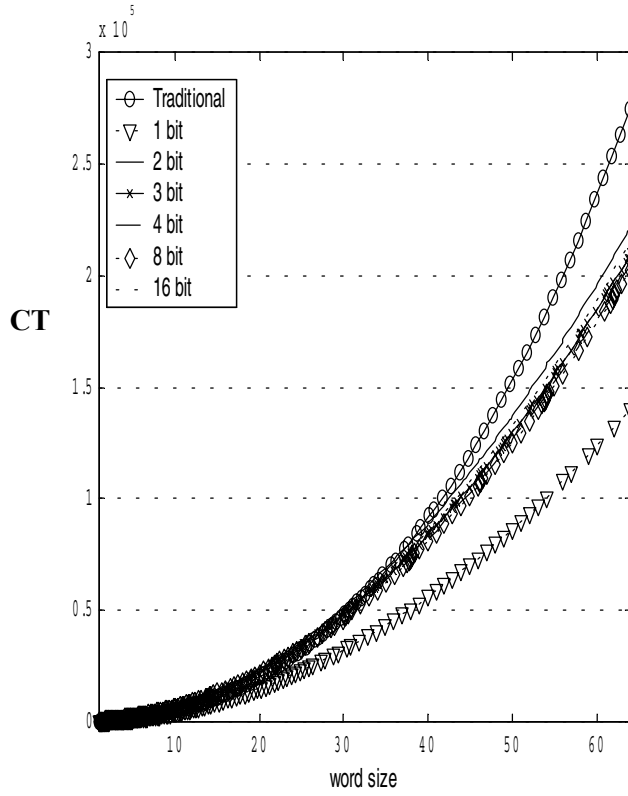


(a)

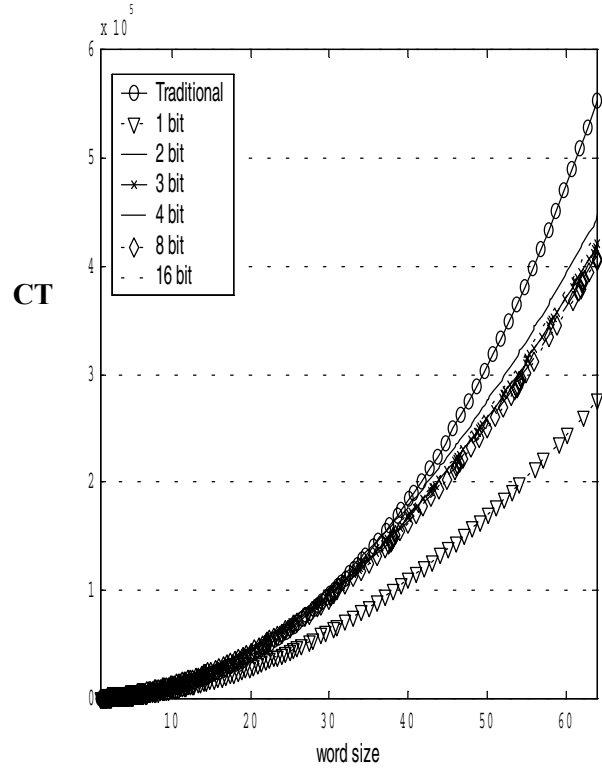


(b)

Figure 9. Cost-performance analysis of SDA for a four product MAC with different word sizes and carry chain lengths (a) Cost vs. word size (b) performance vs. word size (0 routing delay assumed) (figure continued)



(c)



(d)

Figure 9 (continued). Cost-performance analysis of SDA for a four product MAC with different word sizes and carry chain lengths (c) cost-performance ratio vs. word size (0 routing delay) (d) cost-performance ratio vs. word size (routing delay=logic delay)

#### 4. IMPLEMENTATION RESULTS

The designs described in the earlier section were coded using Verilog HDL and implemented on a Xilinx XC4820-4- BG256 chip using Xilinx Foundation Series 3.1i. The accumulator and adder for traditional designs were implemented from the Core Generator library, and contain dedicated carry logic. The designs presented in the paper were verified through extensive simulations. Constraints were entered in the user constraints file (ucf). After the place-and-route (PAR) tool completed the placement and routing, the timing for the design was obtained using the Timing Analyzer. The resources required for each design are given in terms of the total number of CLBs (*Configurable Logic Blocks*) that are used. The word size used for all the designs is 32 bits. Table 3 compares the resources and computation time required by the traditional and proposed one LUT-based serial DA designs for four-product MAC. The overall path delay has two components: delay through the logic gates (logic delay) and the route delay. After the designs were placed and routed by the place-and-route tool, they were examined

using the Timing Analyzer tool. The tool showed that the traditional design could be clocked at a period of 27ns, of which 13.4ns is logic delay and 13.6ns is routing delay. The logic delays and the number of CLBs used correspond closely with those predicted by the model in the previous section. The traditional design has a larger route delay because of the additional overhead of routing delay between the stages in a 32-bit accumulator or adder. The proposed design has a period of 15ns, of which 6.4ns is logic delay and 8.6ns is routing delay. This design has fewer logic levels, since there is no carry propagation, than the traditional design resulting in lower logic delay. However, it consumes more resources because extra registers are required to hold the carry bits in the adder and accumulator, as shown earlier. Table 4 shows the results for serial DA for eight-product MAC. This uses more CLBs than the serial DA with one LUT because of four extra PSC shift registers and a lookup table. However, the logic and routing delays are almost the same as in the previous case. The final addition introduces a combinational delay of 29.4 ns, through the adder, before the final

Table 3. One LUT-based serial DA design for four-product MAC

Design	CLBs	Clock cycles	Logic delay (ns)	Route delay (ns)	Computation Time (ns)
Traditional	114	33	13.4	13.6	891.0
Proposed (1-bit)	165	33	6.4	8.6	527.0



**Table 4. Two LUT-based serial DA design for eight-product MAC**

Design	CLBs	Clock cycles	Logic delay (ns)	Route delay (ns)	Computation Time (ns)
Traditional	231	34	13.2	13.3	902.3
Proposed (1-bit)	314	35	6.4	9.6	589.4

**Table 5. Two LUT-based parallel DA design for four-product MAC**

Design	CLBs	Clock cycles	Logic delay (ns)	Route delay (ns)	Computation Time (ns)
Traditional	151	17	13.2	13.5	480.6
Proposed (1-bit)	270	18	6.4	9.6	346.8

result is available in the proposed design. The traditional design has a larger routing delay because of the additional routing overhead between stages. As the two tables above show, the (1-bit) proposed designs run faster by a factor of over 1.5 over traditional design.

In Table 5 the traditional and proposed designs for 2-bit parallel DA for four-product MAC are compared. As before, the proposed design with carry chain length of 1-bit has a smaller computation time than traditional.

## 5 CONCLUSION

In this paper a novel architecture for serial and parallel DALUT and accumulator structures is proposed that reduces or eliminates the carry chain delay from the critical path. New designs for serial and parallel DA are presented. Also, a cost-performance analysis of the SDA for a four-product MAC is performed. Instead of using gate counts and gate delays as in VLSI, the cost is measured as the CLB count and the performance measured as the inverse of the total computation time (dependent on the number of CLB logic levels and routing delays) to complete the DA operation. The cost-performance analysis shows that proposed  $r$ -bit designs,  $r < n$ , have a better performance than traditional DA designs on XC4000 series. The 1-bit designs have the best performance, but a high cost. The 8-bit designs provide a good compromise between cost and performance. Other families of chips can also be analyzed similarly. The designs are implemented using Xilinx Foundation series 3.1i on a Xilinx XC4028-3-BG256 chip. The results show that the new designs proposed in this paper show speedup by a factor of at least 1.5 over traditional DA designs. This can pave the way for faster DSP chips since DA is used in many signal and image processing applications. Our future work consists of using the DA architecture described in this paper in DSP.

## REFERENCES

- [1] S.A. White. Applications of Distributed Arithmetic to Digital Signal Processing: A Tutorial Review. *IEEE ASSP Magazine*, Vol. 6, No. 3, pp. 4-19.
- [2] B. New, "A Distributed Arithmetic Approach to Designing Scalable DSP Chips", *Electronic Design News*, August 17, 1995.
- [3] Mintzer, L. FIR filters with the Xilinx FPGA. *FPGA '92 ACM/SIGDA, Workshop on FPGAs*. pp. 129-134.
- [4] J. Valls, M. Martinez-Peiro, T. Sansaloni, and E. Boemo. Design and FPGA Implementation of Digit-Serial FIR Filters. *Proceedings of the 1998 IEEE ICECS'98 (5th IEEE International Conference on Electronics, Circuits and Systems)*, Vol.2, pp.191-194, Lisboa, 7-10 Sept. 1998.
- [5] N.W. Bergman, Y.Y. Chung, B.K. Gunther. Efficient Implementation of the DCT on Custom Computers. In Kenneth L. Pocek and Jeffrey Arnold, editors, *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 244-245, Los Alamitos, CA, April 1997.
- [6] R. Woods, D. Trainor, and J.-P. Heron. Applying an XC6200 to Real-Time Image Processing. *IEEE Design and Test of Computers*, Vol. 15, No. 1, January/March 1998.
- [7] R. Grover, W. Shang, Q. Li. A Comparison of FPGA Implementations of Bit-level and Word-level Matrix Multipliers. *Proc. 10th Intl. Conf. on field-programmable logic and applications, FPL 2000*, Villach, Austria, August 27-30, 2000, pp. 422-431.
- [8] W. Shang, B. W. Wah. Dependence Analysis and Architecture Design for Bit-Level Algorithms. *Intl. Conf. On Parallel Process*, vol. I, pp. 30-38, 1993.
- [9] Zhen Luo and Margaret Martonosi. Accelerating Pipelined Integer and Floating-Point Accumulations in Configurable Hardware with Delayed Addition Techniques. *IEEE Transactions on Computers*, Vol. 49, No. 3, March 2000.
- [10] Xilinx Inc. Estimating the Performance of XC4000E Adders and Counters, v.2.0, July 1996. Available from: <http://www.xilinx.com/xapp/xapp018.pdf>
- [11] Xilinx Inc. XC4000 XL Electrical characteristics, v1.7, October 1999.
- [12] J. Valls, M. Martinez, T. Sansaloni, and E. Boemo. A Study about FPGA-based Digital Filters. *Proc. 1998 IEEE SIPS, IEEE Workshop on VLSI Signal Processing: Design and Implementation*, pp.191-201, Boston, Oct.1998.