International Scholarly Research Network
ISRN Electronics
Volume 2012, Article ID 253742, 9 pages
doi:10.5402/2012/253742

Research Article

Design of High-Speed Adders for Efficient Digital Design Blocks

Deepa Yagain, Vijaya Krishna A, and Akansha Baliga

Department of Electronics and Communication, People's Education Society Institute of Technology, Karnataka Bangalore 560 085, India

Correspondence should be addressed to Deepa Yagain, deepa.yagain@gmail.com

Received 19 June 2012; Accepted 22 July 2012

Academic Editors: J. Solsona and Y. Takahashi

Copyright © 2012 Deepa Yagain et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The core of every microprocessor and digital signal processor is its data path. The heart of data-path and addressing units in turn are arithmetic units which include adders. Parallel-prefix adders offer a highly efficient solution to the binary addition problem and are well suited for VLSI implementations. This paper involves the design and comparison of high-speed, parallel-prefix adders such as Kogge-Stone, Brent-Kung, Sklansky, and Kogge-Stone Ling adders. It is found that Kogge-Stone Ling adder performs much efficiently when compared to the other adders. Here, Kogge-Stone Ling adders and ripple adders are incorporated as a part of a lattice filter in order to prove their functionalities. It is seen that the operating frequency of lattice filter increases if parallel prefix Kogge-Stone Ling adder is used instead of ripple adders since the combinational delay of Kogge-Stone Ling adder is less. Further, design and comparison of different tree adder structures are performed using both CMOS logic and transmission gate logic. Using these adders, unsigned and signed comparators are designed as an application example and compared with their performance parameters such as area, delay, and power consumed. The design and simulations are done using 65 nm CMOS design library.

1. Introduction

Binary addition is one of the most primitive and most commonly used applications in computer arithmetic. A large variety of algorithms and implementations have been proposed for binary addition [1–3]. Parallel-prefix adder tree structures such as Kogge-Stone [4], Sklansky [5], Brent-Kung [6], Han-Carlson [7], and Kogge-Stone using Ling adders [8, 9] can be used to obtain higher operating speeds. Parallel-prefix adders are suitable for VLSI implementation since they rely on the use of simple cells and maintain regular connections between them. VLSI integer adders are critical elements in general purpose and digital-signal processors since they are employed in the design of Arithmetic-Logic Units, floating-point arithmetic data paths, and in address generation units. Moreover, digital signal processing makes extensive use of addition in the implementation of digital filters, either directly in hardware or in specialized digital signal processors (DSPs). In integer addition, any decrease in delay will directly relate to an increase in throughput. In nanometer range, it is very important to develop addition algorithms that provide high performance while reducing power consumption. The requirements of the adder are that it should be primarily fast and secondarily efficient in terms of power consumption and chip area. For wide adders ($N > 16$), the delay of carry look-ahead adders becomes dominated by the delay of passing the carry through the look-ahead stages. This delay can be reduced by looking ahead across the look-ahead blocks. In general, we can construct a multilevel tree of look-ahead structures to achieve delay that grows with $\log N$. Such adders are variously referred to as tree adders or parallel prefix adders. Many parallel prefix networks have been described in the literature, especially in the context of addition. The classic networks include Brent-Kung, Sklansky, Kogge-Stone, and Han-Carlson adders. The basic components of adders can be designed in many ways. Initially, the combinational delay and functionality can be verified using HDLs, and optimization can be seen at architecture level. At second level, optimization can also be achieved by using specific logic families in the design. In this paper, adder components are designed, analyzed, and compared using CMOS gates

and transmission gates using 130 nm technology file. This is a deep submicron technology file. Several variants of the carry look-ahead equations, like Ling carries [9], have been presented that simplify carry computation and can lead to faster structures. Most high speed adders depend on the previous carry to generate the present sum. Ling adders [8, 9], on the other hand, make use of Ling carry and propagate bits, in order to calculate the sum bit. As a result, dependency on the previous bit addition is reduced; that is, ripple effect is lowered. This paper provides a comparative study on the implementation of the abovementioned high-speed adders. By designing and implementing high-speed adders, we found that the power consumption and area reduced drastically when the gates were implemented using transmission gates. This is found to happen without compromising on the speed. Later as an application example such as magnitude comparator is designed using Kogge-Stone Ling adder to verify the efficiency.

2. Adders

2.1. Carry Look Ahead Adders. Consider the n-bit addition of two numbers: $A = a_{n-1}, a_{n-2}, \ldots, a_0$ and $B = b_{n-1}, b_{n-2}, \ldots, b_0$ resulting in the sum, $S = s_{n-1}, s_{n-2}, \ldots, s_0$ and a carry, C_{out}. The first stage in CLA computes the bit generate and bit propagate as follows:

$$\begin{aligned} g_i &= a_i \cdot b_i \\ p_i &= a_i + b_i, \end{aligned} \tag{1}$$

where g_i is the bit generate and p_i is the bit propagate. The schematic of g_i and p_i using CMOS and transmission gates design style is as shown in Figure 1.

These are then utilized to compute the final sum and carry bits, in the last stage as follows:

$$\begin{aligned} s_i &= p_i \oplus c_i, \\ c_{i+1} &= g_i + p_i \cdot c_i, \end{aligned} \tag{2}$$

where $\cdot$, $+$ and $\oplus$ represent AND, OR, and XOR operations. It is seen from (2) that the first and last stages are intrinsically fast because they involve only simple operations on signals local to each bit position. However, intermediate stages embody the long-distance propagation of carries, as a result of which the performance of the adder hinges on this part [10]. These intermediate stages calculate group generate and group propagate to avoid waiting for a ripple which, in turn, reduces the delay. These group generate and propagates are given by

$$\begin{aligned} P_{i:j} &= P_{i:k} \cdot P_{k-1:j}, \\ G_{i:j} &= G_{i:k} + G_{k-1:j} \cdot P_{i:k}. \end{aligned} \tag{3}$$

There are many ways to develop these intermediate stages, the most common being parallel prefix. Many parallel prefix networks have been described in the literature, especially in the context of addition. In this paper, we have used the Kogge-Stone implementation, Hans-Carlson, Sklansky, Brent-Kung implementation of CLA, and Kogge-Stone implementation of Ling adder. PG logic in all adders is generally represented in the form of cells. These diagrams known as cell diagrams will be used to compare a variety of adder architectures in the following sections. Here two cells are used for implementation of all the adders: grey cell and the black cell. The basic block diagrams are as shown in Figure 2.

3. Analysis of Adders

In this paper, mathematical analysis is given for Ling adders. Similar analysis can be given for all other adders as well.

3.1. Brent-Kung Implementation. The Brent-Kung tree computes prefixes for 2-bit groups. These are used to find prefixes for 4-bit groups, which in turn are used to find prefixes for 8-bit groups, and so forth. The prefixes then fan back down to compute the carries-in to each bit. The tree requires $2\log_2 N - 1$ stages. The fanout is limited to 2 at each stage. The diagram shows buffers used to minimize the fanout and loading on the gates, but, in practice, the buffers are generally omitted. The basic blocks used in this case are gray and black cells which are explained in Section 2. This adder is implemented for 8, 16, and 32 bits using CMOS logic and transmission gate logic.

3.2. Sklansky Implementation. The *Sklansky* or *divide-and-conquer* tree reduces the delay to $\log_2 N$ stages by computing intermediate prefixes along with the large group prefixes. This comes at the expense of fanouts that double at each level. The gates fanout to $(8, 4, 2, 1)$, respectively. These high fanouts cause poor performance on wide adders unless the high fanout gates are appropriately sized, or the critical signals are buffered before being used for the intermediate prefixes. Transistor sizing can cut into the regularity of the layout because multiple sizes of each cell are required although the larger gates can spread into adjacent columns.

3.3. Han-Carlson Adder. The *Han-Carlson* trees are a family of networks between Kogge-Stone and Brent-Kung. The logic performs Kogge-Stone on the odd numbered bits and then uses one more stage to ripple into the even positions.

3.4. Kogge-Stone Adders. The main difference between Kogge-Stone adders and other adders is its high performance. It calculates carries corresponding to every bit with the help of group generate and group propagate. In this adder the logic levels are given by $\log_2 N$, and fanout is 2.

3.5. Ling Adders. Ling [8] proposed a simpler form of CLA equations which rely on adjacent pair bits (a_i, b_i) and (a_{i-1}, b_{i-1}). Along with bit generate and bit propagate, we introduce another prefix bit, the half sum bit given by

$$d_i = a_i \oplus b_i. \tag{4}$$

Table 1: Delay, power and area consumed for different adders: a comparision.

Adder	Number of bits	CMOS logic			Transmission gate logic		
		Area (no of transistors)	Power in W	Delay in sec	Area (no of transistors)	Power in W	Delay in sec
Kogge-Stone	8	486	4.13 m	$2.18e-11$	432	1.8799 m	$2.33e-10$
	16	1140	7.694 m	$2.87e-11$	1056	5.2718 m	$2.38e-10$
	32	2658	13.648 m	$3.01e-11$	2345	10.314 m	$2.70e-10$
Sklansky	8	415	17.88 m	$8.08e-10$	323	8.92 m	$7.75e-10$
	16	1047	36.34 m	$11.15e-10$	763	18.73 m	$10.95e-10$
	32	2199	65.13 m	$22.03e-10$	1659	40.2 m	$21.2e-10$
Brent-Kung	8	598	0.18 μ	$7.08e-10$	470	0.13 μ	$7.03e-10$
	16	1268	0.4 μ	$9.24e-10$	1012	0.3 μ	$9.03e-10$
	32	2494	12.5 μ	$11.25e-10$	1982	0.614 μ	$10.4e-10$
Han-Carlson	8	440	10.81 m	$61.66e-09$	312	1.9178 m	$60.18e-09$
	16	992	13.54 m	$82.21e-09$	736	6.411 m	$81.33e-09$
	32	2208	13.99 m	$104.8e-09$	1696	9.825 m	$100.3e-09$
Ling	8	742	0.313 μ	$23.1e-10$	530	0.139 μ	$19.3e-10$
	16	1655	0.6 μ	$35.2e-10$	1250	0.3104 μ	$28.5e-10$
	32	3382	13.3 m	$42.8e-10$	2690	0.4105 μ	$37.4e-10$

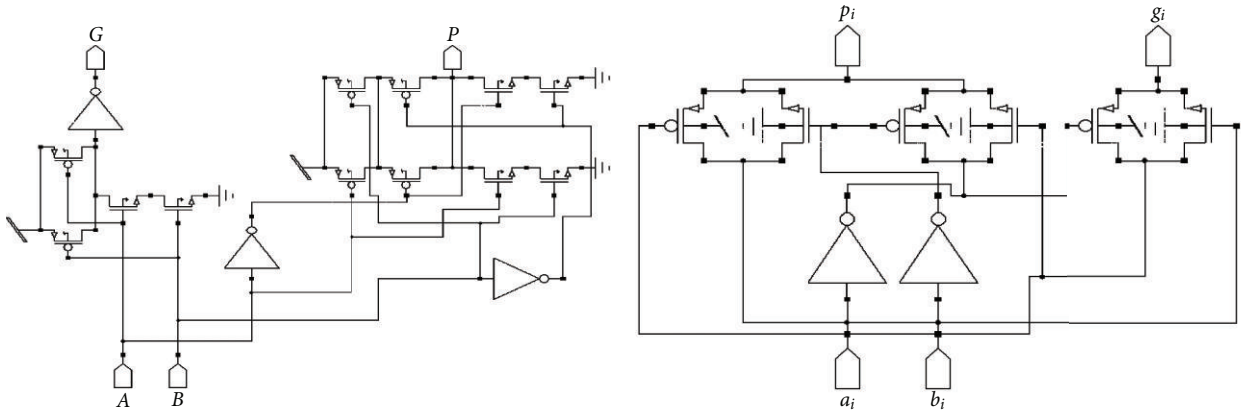


Figure 1: Schematic of bit generate circuit using CMOS and transmission design style.

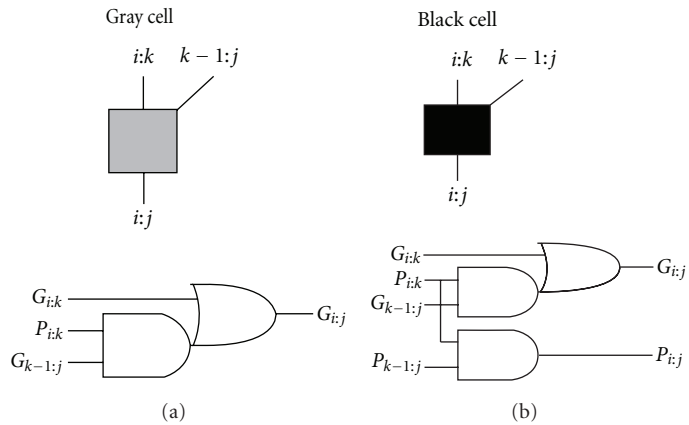


Figure 2: Block diagram of grey cell and black cell.

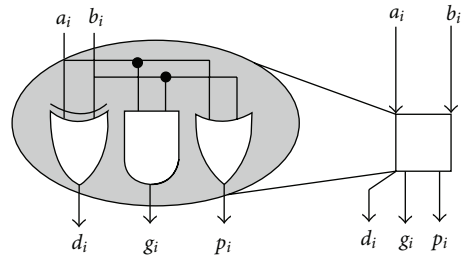


Figure 3: Bit generate and propagate in Ling CLA.

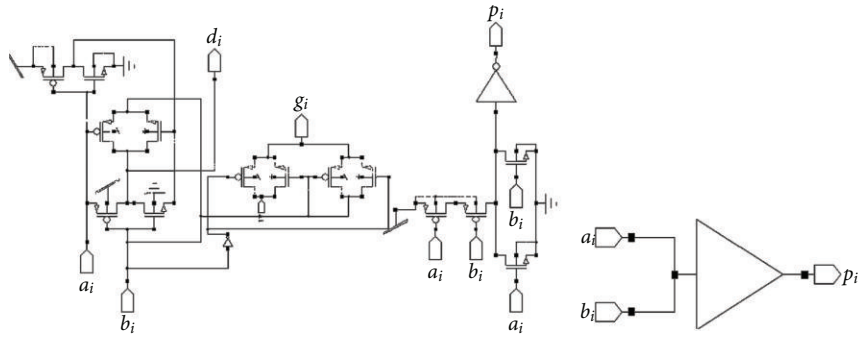


Figure 4: Bit generate, propagate, and half-sum bits using transmission gates.

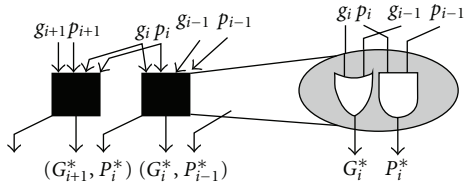


Figure 5: Ling generate and propagate in Ling CLA.

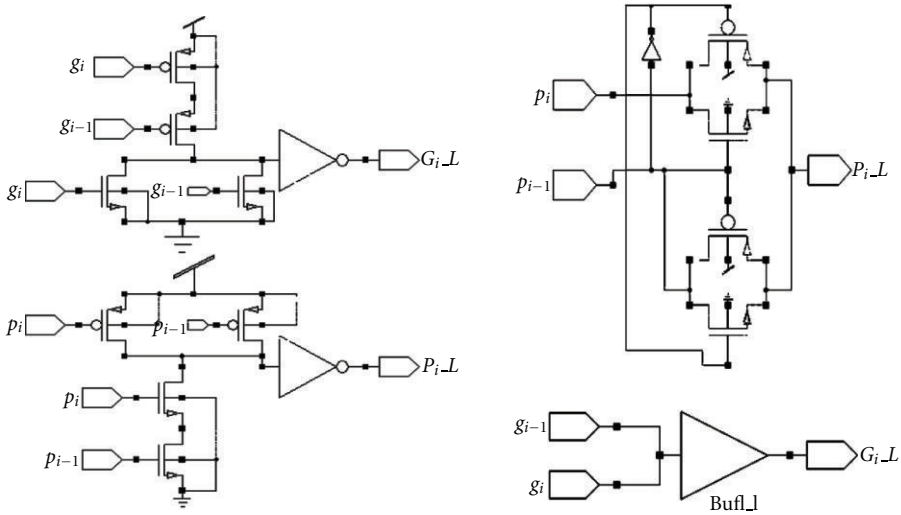


Figure 6: Block generate and propagate (Ling carry) using CMOS and transmission gate.

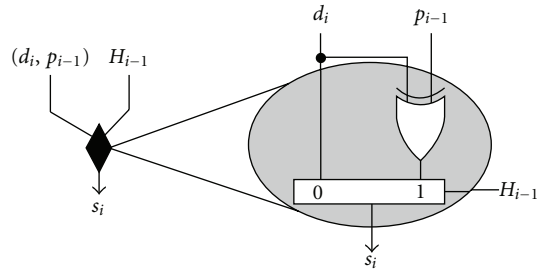


Figure 7: Sum in Ling CLA.

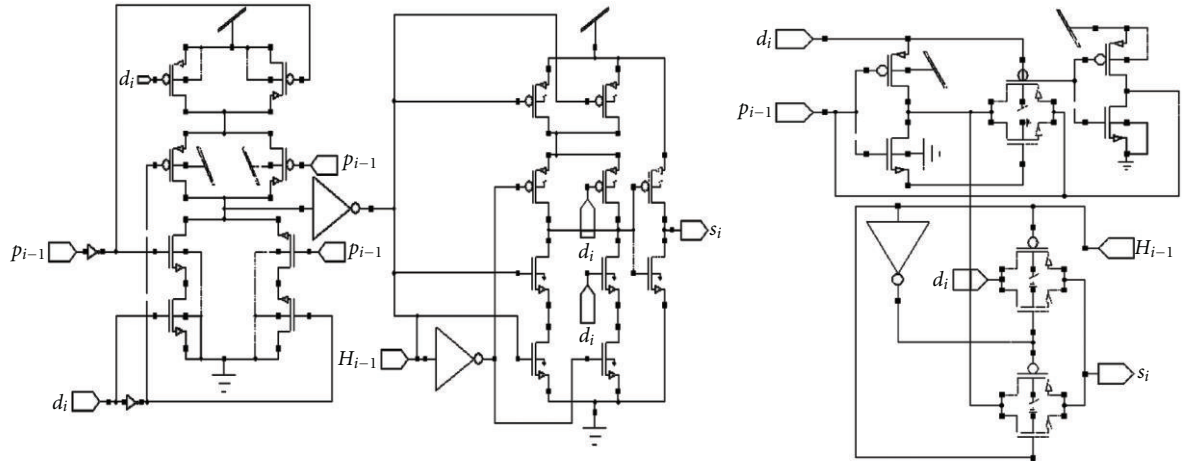


Figure 8: Sum block in Ling adder using CMOS and transmission gates.

Now, instead of utilizing traditional carries, a new type of carry, known as Ling carries, is produced where the ith Ling carry in [11] is defined to be

$$c_i = H_i \cdot p_i, \tag{5}$$

where

$$H_i = c_i + c_{i-1}. \tag{6}$$

In this way, each H_i can be in turn represented by

$$\begin{aligned} H_i = {} & g_i + g_{i-1} + p_{i-1} \cdot g_{i-2} \\ & + \cdots + p_{i-1} \cdot p_{i-2} \cdot p_{i-3} \cdot \ldots p_1 g_0. \end{aligned} \tag{7}$$

We can see from (5) that Ling carries can be calculated much faster than Boolean carry. Consider the case of c_4 and H_4

$$\begin{aligned} c_4 &= g_4 + p_4 \cdot g_3 + p_4 \cdot p_3 \cdot g_2 \\ &\quad + p_4 \cdot p_3 \cdot p_2 \cdot g_1 + p_4 \cdot p_3 \cdot p_2 \cdot p_1 \cdot g_0, \\ H_4 &= g_4 + g_3 + p_3 \cdot g_2 + p_3 \cdot p_2 \cdot g_1 + p_3 \cdot p_2 \cdot p_1 \cdot g_0. \end{aligned} \tag{8}$$

If we assume that all input gates have only two inputs, we can see that calculation of c_4 requires 5 logic levels, whereas that for H_4 requires only four. Although the computation of carry is simplified, calculation of the sum bits using Ling carries is much more complicated. The sum bit, when calculated by using traditional carry, is given to be

$$s_i = d_i \oplus c_{i-1}. \tag{9}$$

Substituting (5) into (9), we get that

$$s_i = d_i \oplus p_{i-1} \cdot H_{i-1}. \tag{10}$$

However, according to [12] the computation of the bits s_i can be transformed as follows:

$$s_i = \overline{H}_{i-1} \cdot d_i + H_{i-1}(d_i \oplus p_{i-1}). \tag{11}$$

Equation (11) can be implemented using a multiplexer with H_{i-1} as the select line, which selects either d_i or $(d_i \oplus p_{i-1})$. No extra delay is added by Ling carries to compute the sum since the delay generated by the XOR gate is almost equal to that generated by the multiplexer and that the time taken to compute the inputs to the multiplexer is lesser than that taken to compute the Ling carry. In [9], a methodology to develop parallel prefix Ling adders using Kogge-Stone [4] and Knowles [8] algorithm was developed. Here, for n-bit addition, Ling carry H_i and H_{i+1} is given by

$$\begin{aligned} H_i &= (G_i^*, P_{i-1}^*) \circ (G_{i-2}^*, P_{i-3}^*) \circ \cdots \circ (G_0^*, P_{-1}^*), \\ H_{i+1} &= (G_{i+1}^*, P_i^*) \circ (G_{i-1}^*, P_{i-2}^*) \circ \cdots \circ (G_1^*, P_0^*), \end{aligned} \tag{12}$$

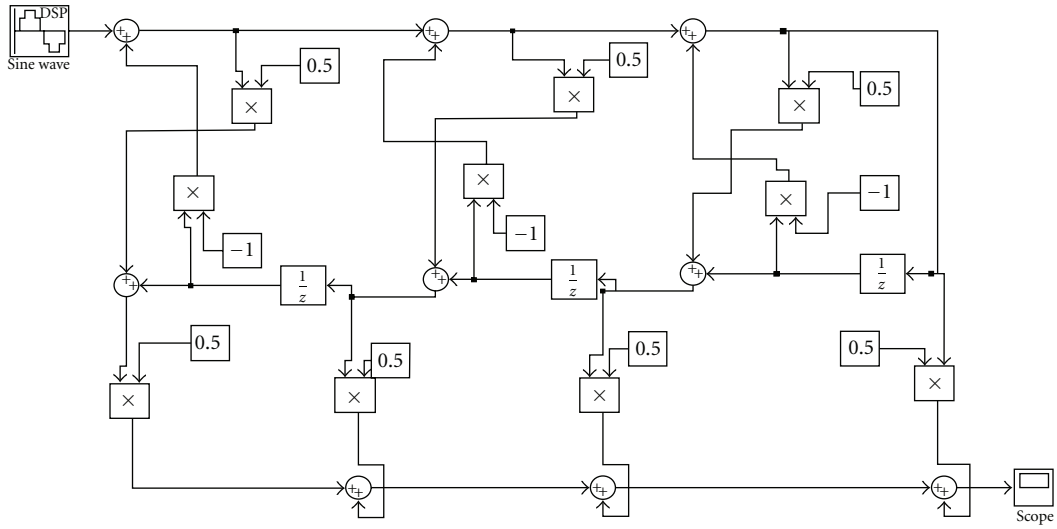


Figure 9: Third-order cascaded IIR lattice filter structure.

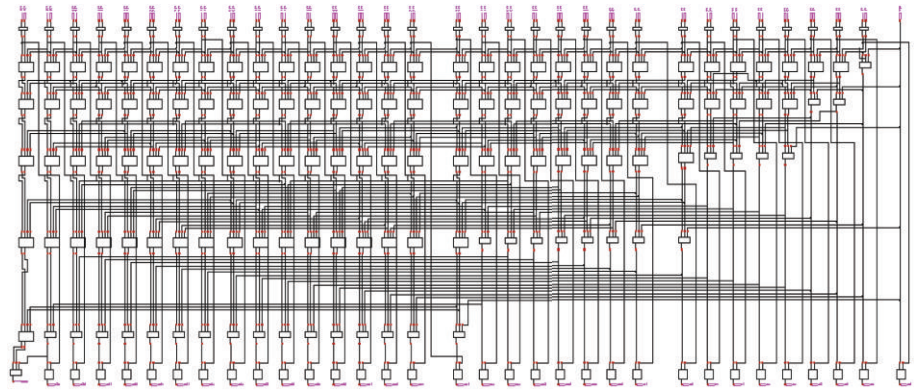


Figure 10: Schematic of 16-bit Kogge-Stone adder using transmission gates.

where

$$G_i^* = g_i + g_{i-1},$$
$$P_i^* = p_i \cdot p_{i-1}. \tag{13}$$

To explain the above equations, consider the 3rd and 4th Ling carry, given by

$$H_3 = g_3 + g_2 + p_2 \cdot g_1 + p_2 \cdot p_1 \cdot g_0,$$
$$H_4 = g_4 + g_3 + p_3 \cdot g_2 + p_3 \cdot p_2 \cdot g_1 + p_3 \cdot p_2 \cdot p_1 \cdot g_0. \tag{14}$$

This can be further reduced by using (13) to

$$H_3 = G_3^* + P_2^* \cdot G_1^*,$$
$$H_4 = G_4^* + P_3^* \cdot G_2^* + P_3^* \cdot P_1^* \cdot G_0^*. \tag{15}$$

This can be then further reduced by using the "$\circ$" operator to

$$H_3 = (G_3^*, P_2^*) \circ (G_1^*, P_0^*),$$
$$H_4 = (G_4^*, P_3^*) \circ (G_2^*, P_1^*) \circ (G_0^*, P_{-1}^*). \tag{16}$$

This allows the parallel prefix computation of Ling adders using a separate tree [9] for even and odd indexed positions. Using this methodology, we implemented a 16-bit adder using the Kogge-Stone tree and then utilized that block to develop 32 and 64-bit adders. The gates and blocks used for this implementation were then modified using transmission gates. Cells other than gray and black cell that are used as components in Ling adder, and they are as explained, in Figures 3 and 4.

Figure 3 forms the first stage in the adder. It generates the bit generate, bit propagate, and half sum bits (for Ling adders) that is g_i, p_i, and d_i, respectively, which are used extensively in the next stages to generate block generate and propagate.

Figure 5 is used to generate the Ling carry H which is nothing but the block generate. This is then used to find subsequent group generate and propagate with the block shown in Figure 6.

Finally the block generates are used to calculate the final sum along with the bit propagate half-sum bits to calculate the sum as in Figures 7 and 8.

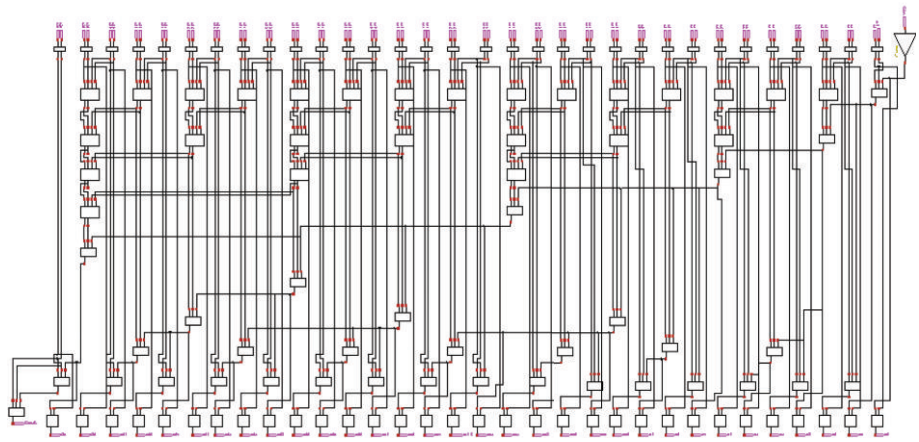


Figure 11: Schematic of 16-bit Brent-Kung adder using transmission gates.

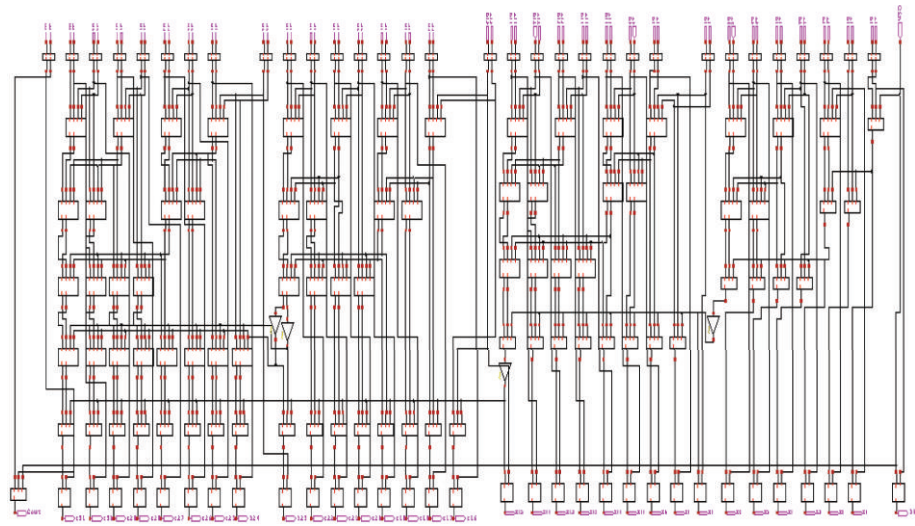


Figure 12: Schematic of 16-bit Sklansky adder using transmission gates.

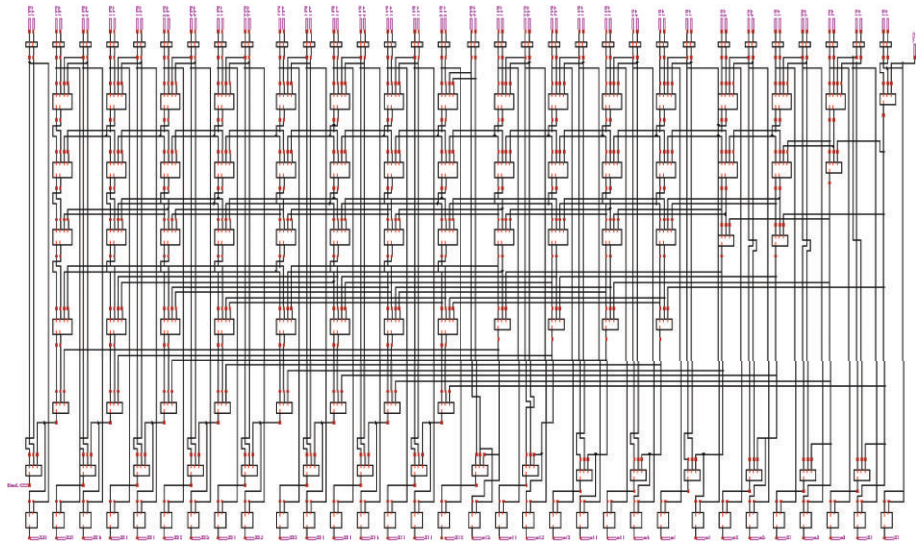


Figure 13: Schematic of 16-bit Han-Carlson adder using transmission gates.

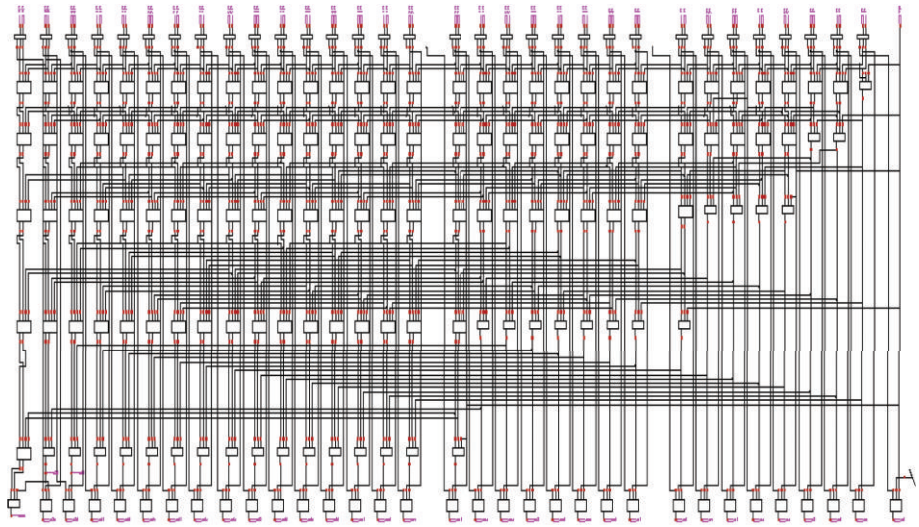


Figure 14: Schematic of 16-bit Kogge-Stone Ling adder using transmission gates.

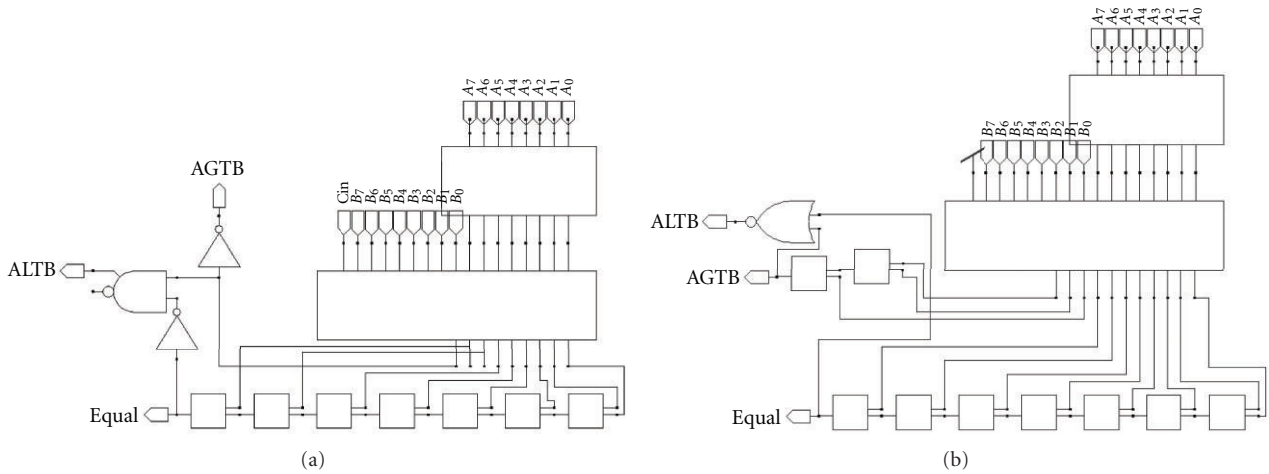


Figure 15: (a) Schematic of 16-bit unsigned comparator, (b) Schematic of 16-bit signed comparator.

Adders are extensively used as a part of filters. Lattice filter structures are used in various signal processing applications, and they are internally considered in the present work. The block diagram of third-order lattice filter is shown in Figure 9. The ripple adders in Lattice filter are replaced with Kogge-Stone Ling adder using component instantiation in VHDL. Here initially, Kogge-Stone Ling adder is implemented in VHDL to observe the functionality and combination delay. It is found that combination delay of 32 Kogge-Stone Ling adder is 12.492 ns which is much less when compared to the ripple adder of 15.504 ns. If components with lesser combinational delay are used in sequential circuits, the clock period will be reduced which internally increases the clock frequency. It is found that the implementation of Ling adder resulted in a 15% less delay when compared to the ripple adders after synthesis.

For cascaded lattice filter shown in Figure 9, with ripple adder, we get the below results after synthesis:

(i) minimum period: 13.058 ns (maximum frequency: 76.579 MHz),

(ii) minimum input arrival time before clock: 2.680 ns, and

(iii) maximum output required time after clock: 21.707 ns.

Similarly for lattice filter with Kogge-Stone Ling adder the postsynthesis results are as follows:

(i) minimum period: 11.097 ns (maximum frequency: 90.112 MHz),

(ii) minimum input arrival time before clock: 2.697 ns, and

(iii) maximum output required time after clock: 13.476 ns.

Hence the clock frequency of any digital filter blocks is found to increase if Kogge-Stone Ling adder is used. This can be used for any digital blocks where operation speed needs to be high.

4. Simulations and Results

Schematic is constructed for 8 bit and 32 adders using CMOS and transmission gates as given in Figures 10, 11, 12, 13, and 14. In each circuit, measurement of power, area, and delay is done. This can be done by designing the basic components such as black and grey cells using CMOS and transmission gates. The performance parameters are obtained for all these using 65 nm technology file, and the different performance parameters are compared for adders using CMOS gates and adders using transmission gates. The result summary of all the adders is given in Table 1.

5. Application Example

Here signed and unsigned magnitude comparator [13, 14] is designed using Kogge-Stone Ling adder. A magnitude comparator determines the larger of two binary numbers. To compare two unsigned numbers A and B, compute $B - A = B + A + 1$. If there is a carryout, $A \leq B$; otherwise, $A > B$. A zero detector indicates that the numbers are equal. Figure 15(a) shows a 8-bit unsigned comparator built from a carry-ripple adder and two complement units. The relative magnitude is determined from the carryout (C) and zero (Z) signals. For wider inputs, any of the faster adder architectures can be used. Figure 15(b) shows 8-bit signed comparator. Comparing signed two's complement numbers is slightly more complicated because of the possibility of overflow when subtracting two numbers with different signs. Instead of simply examining the carry-out, we must determine if the result is negative (N, indicated by the most significant bit of the result) and if it overflows the range of possible signed numbers. The overflow signal V is true if the inputs had different signs (most significant bits), and the output sign is different from the sign of B. The actual sign of the difference $B - A$ is $S = \text{N XOR V}$ because overflow flips the sign. If this corrected sign is negative ($S = 1$), we know that $A > B$. Again, the other relations can be derived from the corrected sign and the Z signal. Carry signal is used here as well for comparison purpose. Kogge-Stone Ling adder as a basic block for comparator design performs much better since its combinational delay is less.

6. Conclusions

From the above work, it was seen that the clock frequency for the IIR filter using Ling adder was more than the clock frequency for the same IIR filter using simple ripple adder. The combinational path delay for the Ling adder was found to be 15% lesser than that for the ripple adder. Using transmission gates reduced the area of the adder and hence the comparator built using the adder, as compared to the area consumed when CMOS logic was used for implementation. Using transmission gate logic reduced the delay and power consumption of the adder, and hence the comparator using these adders, as compared to the delay and power consumed when CMOS logic was used for implementation. The power consumed by the comparator using Ling adder is lesser than the power consumed by comparator designed using other normal tree adders.

References

[1] I. Koren, *Computer Arithmetic Algorithms*, A. K. Peters, 2002.

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

[3] M. Ergecovac and T. Lang, *Digital Arithmetic*, Morgan-Kauffman, 2003.

[4] P. M. Kogge and H. S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations," *IEEE Transactions on Computers*, vol. C-22, no. 8, pp. 786–793, 1973.

[5] J. Sklansky, "Conditional-sum addition logic," *IRE Transactions on Electronic Computers*, vol. 9, pp. 226–231, 1960.

[6] R. P. Brent and H. T. Kung, "A Regular Layout for Parallel Adders," *IEEE Transactions on Computers*, vol. C-31, no. 3, pp. 260–264, 1982.

[7] T. Han and D. Carlson, "Fast area-efficient VLSI adders," in *Proceedings of IEEE Symposium on Computer Arithmetic*, pp. 49–56, May 1987.

[8] H. Ling, "High-speed binary adder," *IBM Journal of Research and Development*, vol. 25, pp. 156–166, 1981.

[9] A. Baliga and D. Yagain, "Design of High speed adders using CMOS and Transmission gates in Submicron Technology: a Comparative Study," in *Proceedings of the 4th International Conference on Emerging Trends in Engineering and Technology (ICETET '11)*, pp. 284–289, November 2011.

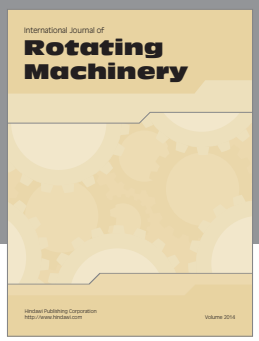
[10] S. Knowles, "A family of adders," in *Proceedings of the 15th IEEE Symposium on Computer Arithmetic*, pp. 277–281, June 2001.

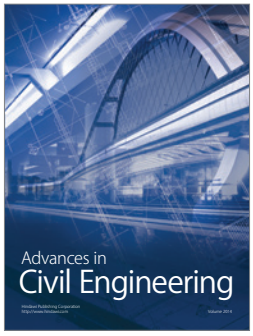
[11] S. Knowles, "A family of adders," in *Proceedings of the 14th IEEE Symposium on Computer Arithmetic*, pp. 30–34, April 1999.

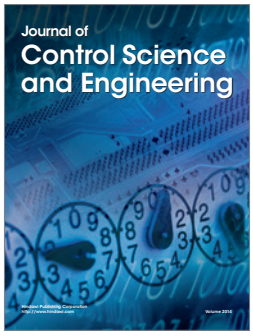
[12] R. E. Ladner and M. J. Fischer, "Parallel prefix computation," *Journal of the ACM*, vol. 27, no. 4, pp. 831–838, 1980.

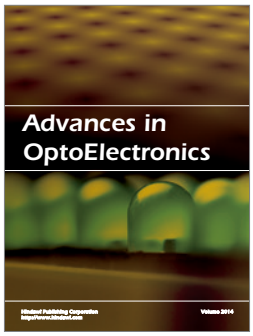
[13] S. Veeramachaneni, M. K. Krishna, L. Avinash, P. Sreekanth Reddy, and M. B. Srinivas, "Efficient design of 32-bit comparator using carry look-ahead logic," in *Proceedings of the IEEE North-East Workshop on Circuits and Systems (NEWCAS '07)*, pp. 867–870, August 2007.

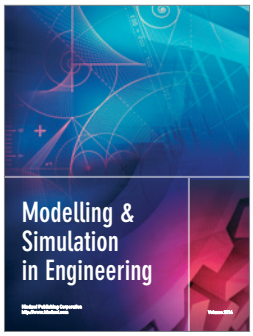
[14] M. Nesenbergs and V. O. Mowery, "Logic synthesis of high speed digital comparators," *Bell System Technical Journal*, vol. 38, pp. 19–44, 1959.

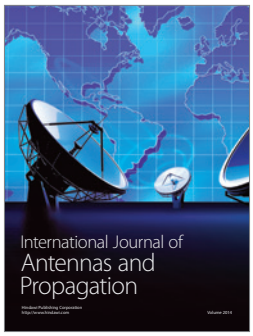


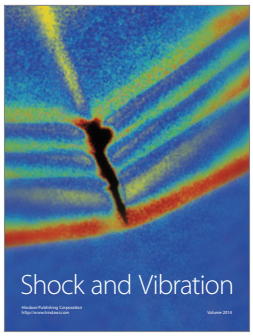


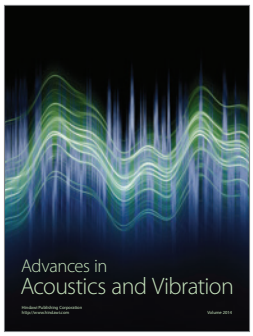












Hindawi

Submit your manuscripts at
http://www.hindawi.com