Design of High speed adders using CMOS and Transmission gates in Submicron Technology: A Comparative Study

Akansha Baliga [#1], Deepa Yagain[#2]
Department of Electronics and Communication (VLSI Design & Embedded Systems)
People's Education Society Institute of Technology
Bangalore-560 085, Karnataka, INDIA
abaliga1990@gmail.com, deepa.yagain@gmail.com

Abstract— **The core of every microprocessor, digital signal processor (DSP), and data-processing application-specific integrated circuit (ASIC) is its data path. At the heart of data-paths and addressing units are arithmetic units, such as comparators, adders, and multipliers and at the heart of arithmetic circuits are adders. The main constraints of all adders are their speed, performance, power consumption and die area. 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 of high speed, parallel-prefix adders such as Brent-Kung ,Sklansky, Kogge-Stone and Ling adders, by Kogge-Stone implementation, using CMOS logic and transmission gate logic. The design and simulations are done using deep submicron technology file. The power, area and delay for the two implementations are compared and it is found that the power, area and delay in the transmission gate logic is much lower than those in CMOS logic. This is done for 8, 16 and 32 bit adders. All the circuits are implemented using Tanner EDA and simulated in 130nm using TSMC MOSIS Level -49 model in TSPICE simulator.**

Keywords- parallel-prefix adders, Brent-Kung , Sklansky, Kogge-Stone , Ling adders by Kogge-Stone, Grey cell, Black cell, Generate and propagate block.

I. 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] [2] [3]. When high operation speed is required, tree structures, like parallel-prefix adders, such as Kogge-Stone[4],Sklansky[5],Brent-Kung[6],Han-Carlson[7] and Kogge-stone using Ling adders[8][17] are utilised. 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.

In integer addition, any decrease in delay will directly relate to an increase in throughput. In nano scale, 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 is 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. The classic parallel prefix networks include Brent-Kung, Sklansky, Kogge-Stone and Han-Carlson adders. The basic components of adders can be designed in many ways. In this paper, adder components are designed, analyzed and compared using CMOS and Transmission gates using 130nm technology file. This is a deep submicron technology file.

Several variants of the carry look-ahead equations, like Ling carries [17], 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] [17] 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. i.e., ripple effect is lowered. This paper provides a comparative study on the implementation of the above mentioned 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. It's found that the delay also reduces in transmission gate high speed adders.

II. ADDERS

A. Carry look ahead adders

Consider the n-bit addition of two numbers, $A=a_{n-1}\ a_{n-2}\ldots\ldots a_0$ 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 is to compute the bit-generate and bit propagate as:

$$g_i = a_i . b_i \quad (1)$$

$$p_i = a_i + b_i \quad (2)$$

Where, g_i is the bit-generate and pi is the bit-propagate. The schematic of g_i and p_i using CMOS and transmission gate logic is shown in Fig.1 and Fig.2.These

978-0-7695-4561-5/11 $26.00 © 2011 IEEE
DOI 10.1109/ICETET.2011.25

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

$$s_i = p_i \oplus c_i \quad (3)$$

$$c_{i+1} = g_i + p_i . c_i \quad (4)$$

Where ·, + and ⊕ represent AND, OR and XOR operations. It is seen from (3) and (4) 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 [13]. These intermediate stages calculate group generate and group propagate to avoid waiting for a ripple which in turn reduces the delay. They are given by,

$$P_{i:j} = P_{i:k} \cdot P_{k-1:j} \quad (5)$$

$$G_{i:j} = G_{i:k} + G_{k-1:j} \cdot P_{i:k} \quad (6)$$

There are many ways to develop these intermediate stages, the most common being parallel prefix. In this paper we have used the Kogge-Stone, Hans-Carlson, Sklansky, Brent-Kung 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. We make use of two cells for implementation of all the adders: Grey Cell and the Black Cell. The schematics and basic block diagrams are as shown in Fig.3and Fig.4.

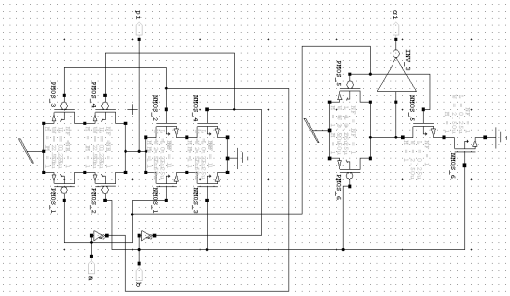

Figure 1: Schematic of Bit generate circuit using CMOS design style

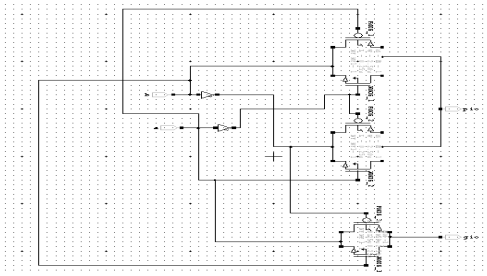

Figure 2: Bit generate circuit using Transmission design style

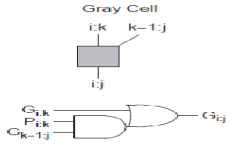

Figure 3: Schematic of Grey cell

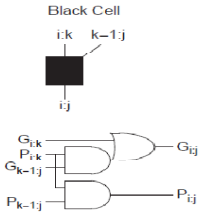

Figure 4: Schematic Of Black Cell

III. Analysis Of Adders

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

A. *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 for 8-bit groups and so on. The prefixes then fan back down to compute the carries-in to each bit. The tree requires 2log2 N – 1 stages. The fan out is limited to 2 at each stage. The diagram shows buffers used to minimize the fan out 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 as follows. This adder is implemented for 8, 16 and 32 bits using CMOS logic and Transmission gate logic.

B. *Sklansky implementation*

The *Sklansky* or *divide-and-conquer* tree reduces the delay to log2 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] other columns. 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.

C. *Hans-Carlson Adder*

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

D. *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.

E. *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

$$d_i = a_i \oplus b_i \quad (7)$$

Now, instead of utilizing traditional carries, a new type of carry, known as Ling carries are produced where the i[th] Ling carry in [10] is defined to be

$$c_i = H_i \cdot p_i \quad (8)$$

Where,

$$H_i = c_i + c_{i-1} \quad (9)$$

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

$$H_i = g_i + g_{i-1} + p_{i-1} \cdot g_{i-2} + \ldots\ldots + p_{i-1} \cdot p_{i-2} \cdot p_{i-3} \cdot \ldots\ldots p_1 g_0 \quad (10)$$

We can see from equation (8) that Ling carries can be calculated much faster than Boolean carry. Consider the case of c_4 and H_4

$$c_4 = g_4 + p_4 \cdot g_3 + p_4 \cdot p_3 \cdot g_2 + p_4 \cdot p_3 \cdot p_2 \cdot g_1 + p_4 \cdot p_3 \cdot p_2 \cdot p_1 \cdot g_0$$

$$H_4 = g_4 + g_3 + p_3 \cdot g_2 + p_3 \cdot p_2 \cdot g_1 + p_3 \cdot p_2 \cdot p_1 \cdot g_0$$

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

$$s_i = d_i \oplus c_{i-1} \quad (11)$$

Substituting equation (8) in (11), we get

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

However, according to [15] the computation of the bits si can be transformed as follows:

$$s_i = \overline{H}_{i-1} \cdot d_i + H_{i-1}(d_i \oplus p_{i-1}) \quad (12)$$

Equation (12) can be implemented using a multiplexer with H_{i-1} as the select line, which selects either di 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 [17] a methodology to develop parallel-prefix Ling adders using Kogge-Stone [4] and Knowles [8] algorithm was developed. Here, Ling carry Hi and Hi+1 for consecutive even and odd positions i and i+1 is

$$H_i = (G_i^*, P_{i-1}^*) \circ (G_{i-2}^*, P_{i-3}^*) \circ \cdots \circ (G_0^*, P_{-1}^*) \quad (13)$$

$$H_{i+1} = (G_{i+1}^*, P_i^*) \circ (G_{i-1}^*, P_{i-2}^*) \circ \cdots \circ (G_1^*, P_0^*) \quad (14)$$

where,

$$G_i^* = g_i + g_{i-1} \quad (15)$$

$$P_i^* = p_i \cdot p_{i-1} \quad (16)$$

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

$$H_3 = g_3 + g_2 + p_2 \cdot g_1 + p_2 \cdot p_1 \cdot g_0$$

$$H_4 = g_4 + g_3 + p_3 \cdot g_2 + p_3 \cdot p_2 \cdot g_1 + p_3 \cdot p_2 \cdot p_1 \cdot g_0$$

This can be further reduced by (15) and (16) to

$$H_3 = G_3^* + P_2^* \cdot G_1^*$$

$$H_3 = (G_3^*, P_2^*) \circ (G_1^*, P_0^*)$$

$$H_4 = G_4^* + P_3^* \cdot G_2^* + P_3^* \cdot P_1^* \cdot G_0^*$$

$$H_4 = (G_4^*, P_3^*) \circ (G_2^*, P_1^*) \circ (G_0^*, P_{-1}^*)$$

This, according to [17], allows the parallel prefix computation of Ling adders using a separate tree for even and odd indexed positions. Using this methodology a 16 bit adder is implemented using the Kogge-Stone tree and then utilize that block to develop 32 and 64 bit adders. The gates and blocks used for this implementation are then modified using transmission gates. Cells other than Gray and Black cells that are used in Ling Adders are as explained in Fig.5 to Fig. 15.

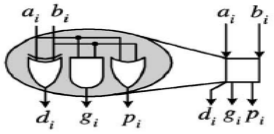

Figure 5: Bit generate and propagate in Ling CLA

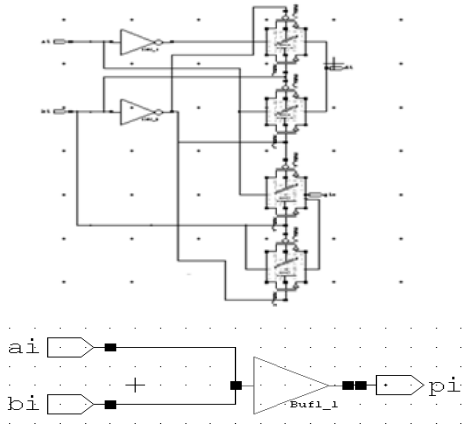

Figure 6: Bit generate, propagate and half-sum bits using transmission Gates

Figure 5 and 6 forms the first stage in the adder. It generates the bit generate, bit propagate and half sum bits (for Ling adders) i.e g_i, p_i and d_i respectively which are used extensively in the next stages to generate block generate and propagate. Figure 7 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 following block

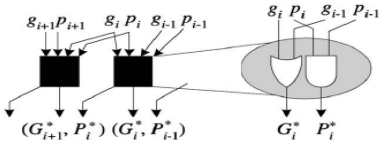

Figure 7: Ling Generate and propagate (Ling carry) in Ling CLA

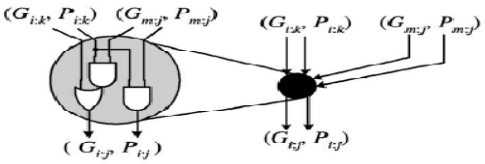

Figure 8: Block generate and propagate in Ling CLA

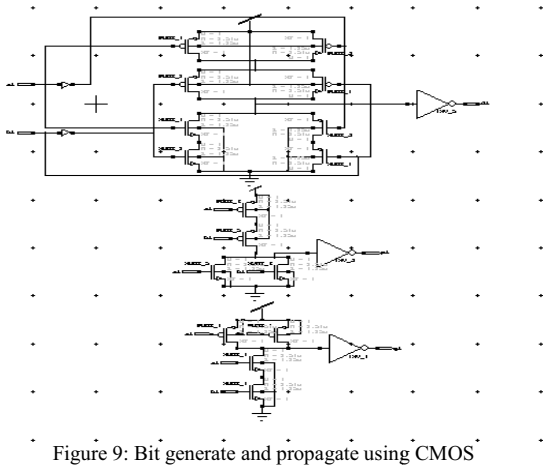

Figure 9: Bit generate and propagate using CMOS

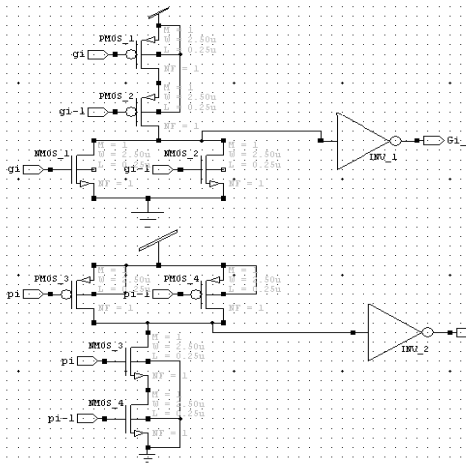

Figure 10: Block generate and propagate (Ling carry) using CMOS

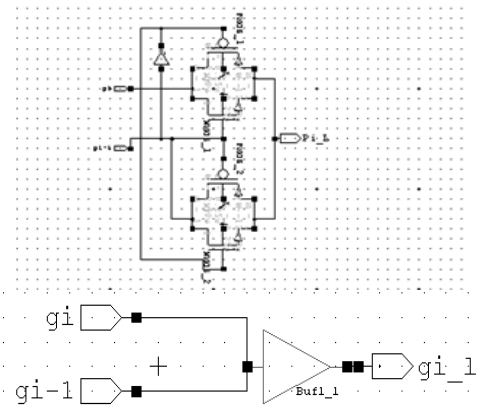

Figure 11: Block generate and propagate (Ling carry) using transmission gates

Finally the block generates are used to calculate the final sum along with the bit propagate half sum bits to calculate the sum as follows:

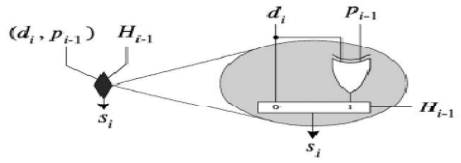

Figure 21: Sum in Ling CLA

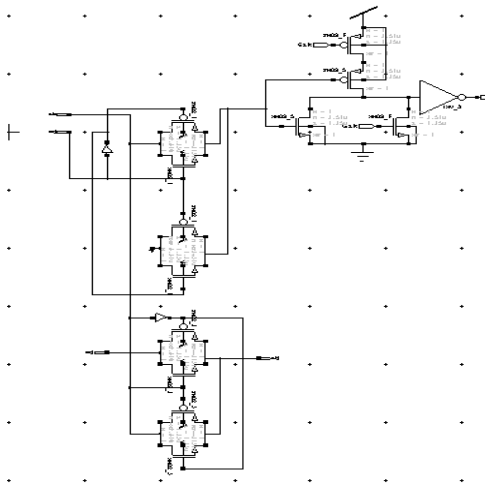

Figure 12: Block generate and propagate using transmission gates

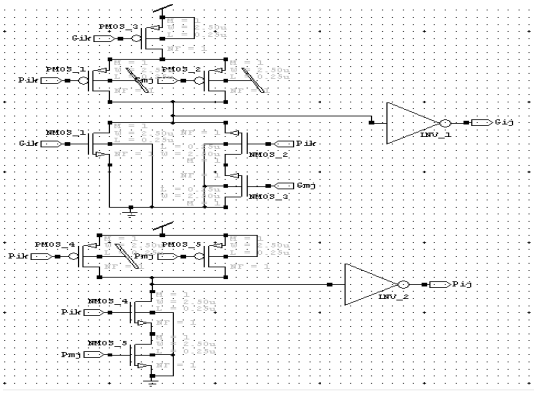

Figure 13: Block generate and propagate using CMOS

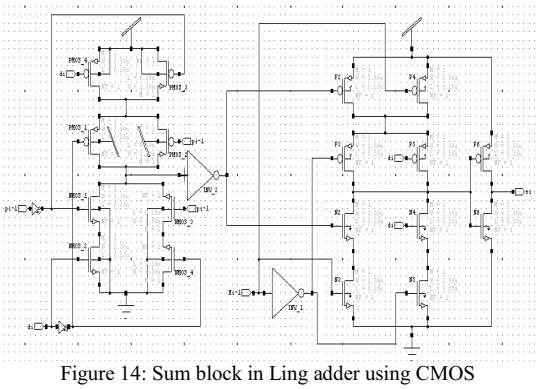

Figure 14: Sum block in Ling adder using CMOS

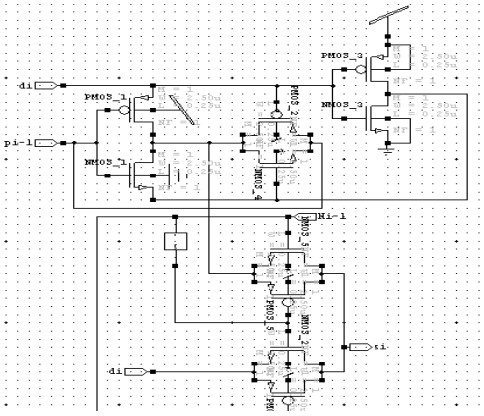

Figure 15: Sum block in Ling adder using transmission gates

IV. Simulations And Results

A. Kogge-Stone implementation

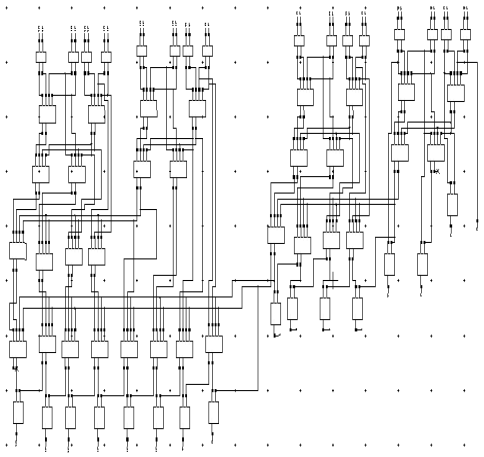

Figure 16: 16 bit Kogge-Stone adder using transmission gates

B. Brent-Kung implementation

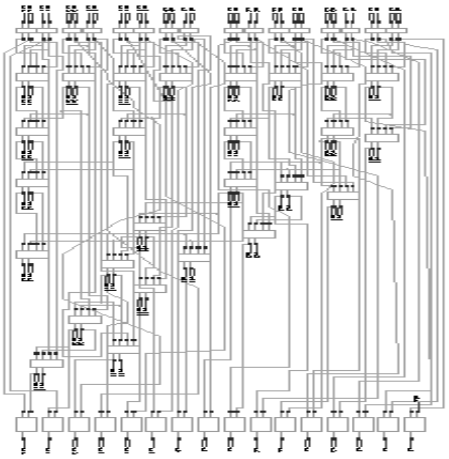

Figure 17: 16 bit Brent-Kung adder using transmission gates

C. Sklansky implementation

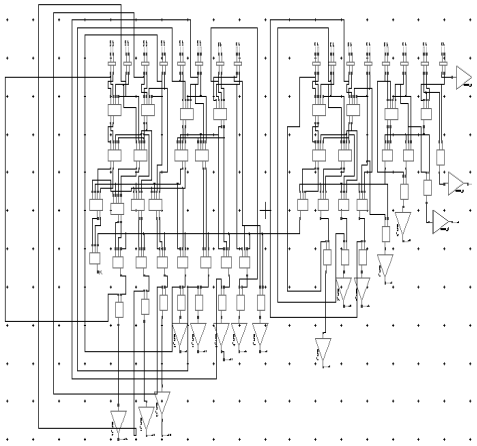

Figure 18: Schematic of 16 bit Sklansky adder using transmission gates

D. Han- Carlson implementation

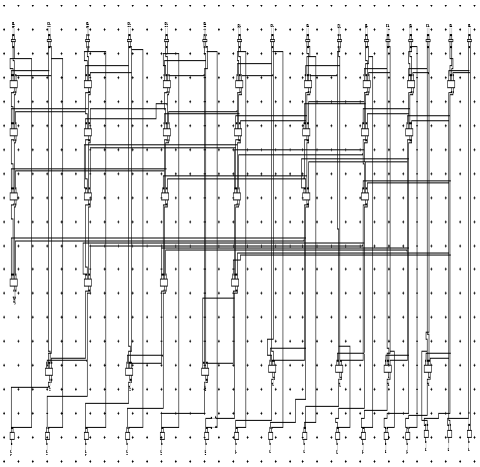

Figure 19: Schematic of 16 bit Han Carlson adder using transmission gates

E. Kogge-Stone Ling implementation

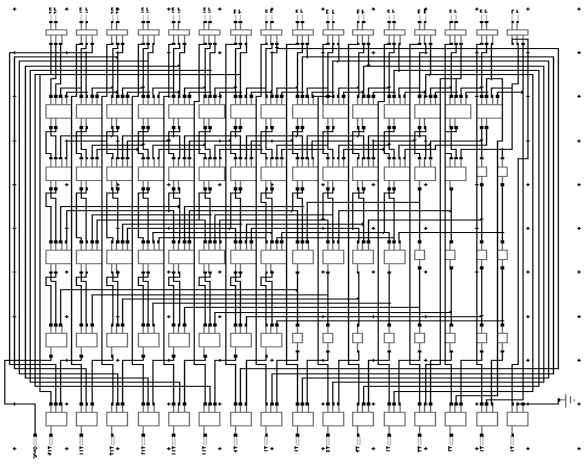

Figure 20: Schematic of 16 bit Kogge-Stone Ling adder using transmission gates

Here 8,16 and 32 adders using transmission gates and CMOS gates are constructed. In each circuit, measurement of power, area and delay is done. The performance parameters are obtained for this as well using 130nm technology file and the different performance parameters are compared for both the cases. This is repeated for all the tree adders under consideration.

		CMOS Logic			Transmission Logic		
	Bits	Power	Delay	Area	Power	Delay	Area
Kogge-Stone Adder	8	4.137mW	2.18e-11	486	1.8799mW	2.33e-10	432
	16	7.694mW	2.87e-11	1140	5.272mW	2.38e-10	1056
	32	13.648mW	3.01e-11	2658	10.314mW	2.70e-10	2345
Brent-Kung	8	0.18uW	7.08e-10	598	0.13uW	7.03e-10	470
	16	0.4uW	9.24e-10	1268	0.3uW	9.03e-10	1012
	32	12.5uW	11.25e-10	2494	0.614uW	10.4e-10	1982
Sklansky	8	17.88mW	8.08e-10	415	8.92mW	7.75e-10	323
	16	36.34mW	11.15e-10	1047	18.73mW	10.95e-10	763
	32	65.13mW	22.03e-10	2199	40.2mW	21.2e-10	1659
Han-Carlson	8	10.81mW	61.66e-09	440	1.9178mW	60.18e-09	312
	16	13.54mW	82.21e-09	992	6.411mW	81.33e-09	736
	32	13.99mW	104.8e-09	2208	9.825mW	100.3e-09	1696
Kogge-Stone Ling	8	0.313uW	23.1e-10	742	0.139uW	19.3e-10	530
	16	0.6uW	35.2e-10	1655	0.3104uW	28.5e-10	1250
	32	13.3mW	42.8e-10	3382	0.4105uW	37.4e-10	2690

TABLE 1: COMPARSION OF 8,16 AND 32 BIT ADDERS IN CMOS AND TRANSMISSION GATE LOGIC

V. CONCLUSIONS

Several high-speed adders are implemented using CMOS logic and Transmission gate logic using Kogge-Stone, Brent-Kung, Han-Carlson and Sklansky implementation along with Ling adders using Kogge-Stone implementation. With this comparative study, choice of adders can be easily made depending on the requirements. It is observed that the power and delay in the adders using transmission gates is much lesser than those adders which used CMOS gates. Also, the number of gates required in the adders using transmission gates is lesser than the number of gates required in the adders built using CMOS logic. All the designs and simulations were performed using 130nm technology file which is a submicron technology file. Thus, we can safely conclude that, the usage of transmission gates to implement basic adder blocks which are used in the implementation of adders, improve the performance of adders in submicron designs.

REFERENCES

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

[2] B. Parhami, Computer Arithmetic—Algorithms and Hardware Designs. Oxford Univ. 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 Trans. Computers, vol. 22, no. 8, pp. 786-792, Aug. 1973.

[5] J. Sklansky; 'Conditional-Sum Addition Logic' IRE Trans., EC-9:226-231, June 1960.

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

[7] T. Han and D. Carlson, "Fast Area-Efficient VLSI Adders," Proc. Symp. Computer Arithmetic, pp. 49-56, May 1987.

[8] H. Ling, "High-Speed Binary Adder," IBM J. R&D, vol. 25, pp. 156-166, May 1981.

[9] R.E. Ladner and M.J. Fisher, "Parallel Prefix Computation," J. ACM, vol. 27, no. 4, pp. 831-838, Oct. 1980.

[10] S. Knowles, "A Family of Adders," Proc. 14th Symp. Computer Arithmetic, pp. 30-34, Apr. 1999. Reprinted in ARITH-15, pp. 277-281.

[11] A. Beaumont-Smith and C.C. Lim, "Parallel-Prefix Adder Design," Proc. 15th Symp. Computer Arithmetic, pp. 218-225, June 2001.

[12] S. Vassiliadis, "Recursive Equations for Hardwired Binary Adders," J. Electronics, vol. 67, no. 2, pp. 201-213, Aug. 1989.

[13] S. Knowles, "A family of adders," Proc. I f h IEEE Symp. Comp. Arith., pp. 277-281, June 2001.

[14] R.P. Brent, H.T. Kung; 'A Regular Layout for Parallel Adders' IEEE Trans., C-31(3):260-264, March 1982.

[15] R.E. Ladner, M.J. Fischer; 'Parallel Prefix Computation' JACM, 27(4):831-838, Oct.1980.

[16] High-Speed Parallel-Prefix VLSI Ling Adders Giorgos Dimitrakopoulos and Dimitris Nikolos, IEEE transactions on computers, Vol. 54, No. 2, February 2005