

Implementation of a Parallel Prefix Adder Based on Kogge-Stone Tree

Yancang Chen^{1,*}, Minlei Zhang¹, Pei Wei¹, Sai Sui¹, Yaxin Zhao¹ and Lunguo Xie²

¹Luoyang Electronic Equipment Test Center, Luoyang, China

²National University of Defense Technology, Changsha, China

*Corresponding author

Abstract—Binary adder is an important module for microprocessors. We present a novel structure of binary adder, named Hyper-Parallel Prefix Adder (HPPA) in this paper. The basic idea of HPPA is to divide the addition into two levels, both of which are parallel prefix adder architecture. It can perfectly reduce the delay of wire loads by reducing the wire length in the critical paths; hence the performance of the adder can be significantly improved. Moreover, we present an optimized carry tree structure which is based on Kogge-stone tree named Grouped-Kogge-Stone tree (GKS tree), which has small nodes especially in the top level. Combining HPPA with GKS tree, we designed and implemented a 64-bit full adder. At the top level, we divide the two 64-bit addition operands into 8 groups, each of which is 8-bit. At the bottom level, we divide each 8-bit group into 4 equal sub-groups. Both of the two levels are composed of parallel prefix adders based on Grouped-Kogge-Stone tree. We design the adder using Verilog HDL to verify the performance. We have implemented the adder under 0.13um CMOS process. Simulation results show that the maximal delay of the proposed adder layout is 578ps with the average power of 18.3mW.

Keywords—parallel prefix adder; CMOS process; maximal delay; average power; kogge-stone tree

I. INTRODUCTION

Binary adder is an important module of microprocessors. It is not only be used to complete addition and subtract operation, but also be used to achieve multiplication operation, division operation and so on. To the best of our knowledge, the main structures of adder include Carry-ripple, Carry-skip, Carry-select, Carry-look-ahead, Parallel- prefix, and so on.

Carry-Ripple is the basic structure of adders. Its carry bit must be spread from the least significant bit to the most significant bit which forms the critical path of the adder[4]. The carry-skip adder divides operands into several groups and can improve performance only in some cases. Because of the use of forward-looking, the performance of carry-select adders improved significantly. The critical path of carry-select adders is the cascade of the carry bit of the first group and several multiplexers. Each carry bit of carry-look-ahead adders has nothing to do with the front carry bit. The increasing complexity of each carry circuit makes the delay of adders increasing linearly. The parallel prefix adder is an improved architecture of carry-look-ahead adders.

Parallel prefix adder is particularly because it can be attractively fast and compact when implemented in VLSI [3].

The critical path of parallel prefix adder is the carry tree, so lots of people make their greatest efforts to improve the speed of carry tree and have presented lots of structures of carry tree, such as Kogge-Stone tree[1], Brent-Kung tree[5], Sklansky tree[6], Han-Carlson tree[7], Ladner-Fischer tree, Knowles tree and so on. These structures have the same goal which is to compute all the carry bits as soon as possible. The difference of these structures is the links between the nodes of trees. Among the architectures of carry tree, the Kogge-Stone tree is the fast architecture in principle, but its area and power costs are expensive.

In this paper, we present an optimized carry tree structure which is based on Kogge-stone tree named Grouped-Kogge-Stone tree (GKS tree). By using the GHC tree hierarchically, we can design optimized hyper-parallel prefix adders.

The remainder of this paper is organized as follows: Section 2 describes basic algorithmic of parallel prefix adder; Section 3 presents the Grouped-Kogge-Stone tree; Section 4 presents the hyper-parallel prefix adder (HPPA) based on GKS tree; Section 5 describes the design and implementation of a 64-bit HPPA based on 8-bit GKS tree, including the detailed structure of the adder, RTL model and verification, circuit of the logical, simulation results and comparison; Finally, we give the conclusions and the future works.

II. BASIC ALGORITHMIC OF PARALLEL PREFIX ADDER

There are many theories about parallel prefix adders. This section we will describe the basic algorithmic of parallel prefix adder. This is similar with [3][11][13][14][15], and we made some reasoning changes.

Parallel prefix adder is composed of bit-generate G_i and bit-propagate P_i functions, and can be considered as three stage circuits, preprocessing stage, parallel prefix calculation stage and computing sum stage, respectively. Figure 1 shows the details work flow and detail operations of each stage.

Let's consider the addition of two n-bit binary numbers which are denoted as $A=A_{n-1}A_{n-2}\cdots A_0$ and $B=B_{n-1}B_{n-2}\cdots B_0$. The carry bits and the sum are denoted as $C=C_{n-1}C_{n-2}\cdots C_0$ and $S=S_{n-1}S_{n-2}\cdots S_0$. The following functions are hold. Note, symbols $+$, \bullet and \oplus denote logical AND, OR and exclusive-OR operations, respectively.

$$\begin{aligned}
C_0 &= A_0 \bullet B_0 \\
S_0 &= A_0 \oplus B_0 \\
C_1 &= A_1 \bullet B_1 + (A_1 \oplus B_1) \bullet C_0 \\
S_1 &= A_1 \oplus B_1 \oplus C_0 \\
&\dots\dots \\
C_i &= A_i \bullet B_i + (A_i \oplus B_i) \bullet C_{i-1} \\
S_i &= A_i \oplus B_i \oplus C_{i-1} \quad (\forall i, 0 \leq i \leq n-1)
\end{aligned} \quad (1)$$

According to the well-known equations $G_i = A_i \bullet B_i$ and $P_i = A_i \oplus B_i$, the functions (1) can be simply written as:

$$\begin{aligned}
C_i &= G_i + (P_i) \bullet C_{i-1}, \\
S_i &= P_i \oplus C_{i-1} \quad (\forall i, 0 \leq i \leq n-1)
\end{aligned} \quad (2)$$

Unwinding the expression of C_i according to the equations (2) as follows

$$\begin{aligned}
C_i &= G_i + P_i \bullet C_{i-1} \\
C_i &= G_i + P_i \bullet G_{i-1} + P_i \bullet P_{i-1} \bullet C_{i-2} \\
&\dots\dots
\end{aligned} \quad (3)$$

$$C_i = G_i + P_i \bullet G_{i-1} + P_i \bullet P_{i-1} \bullet G_{i-2} + P_i \bullet P_{i-1} \bullet \dots \bullet P_{j+1} \bullet G_j + P_i \bullet P_{i-1} \bullet \dots \bullet P_j \bullet C_{j-1} \quad (4)$$

$$C_i = G_{i,j} + P_{i,j} \bullet C_{j-1} \quad (5)$$

where $G_{i,j} = G_i + P_i \bullet G_{i-1} + P_i \bullet P_{i-1} \bullet G_{i-2} + P_i \bullet P_{i-1} \bullet \dots \bullet P_{j+1} \bullet G_j$ and $P_{i,j} = P_i \bullet P_{i-1} \bullet \dots \bullet P_j$ represent group bit-generate and group bit-propagate, respectively[3][15]. [15] presented a prefix operator \circ to transform carry computation to prefix problem. The definition of prefix operator is shown as following.

$$(G, P) \circ (G', P') = (G + P \bullet G', P \bullet P')$$

Therefore, $(G_{i,j}, P_{i,j})$ can be represented as follow.

$$(G_{i,j}, P_{i,j}) = (G_i, P_i) \circ (G_{i-1}, P_{i-1}) \circ \dots \circ (G_{j+1}, P_{j+1}) \circ (G_j, P_j) \quad (6)$$

The equation (6) is the basis of prefix computation. The basic algorithmic of prefix operation is described in Figure I.

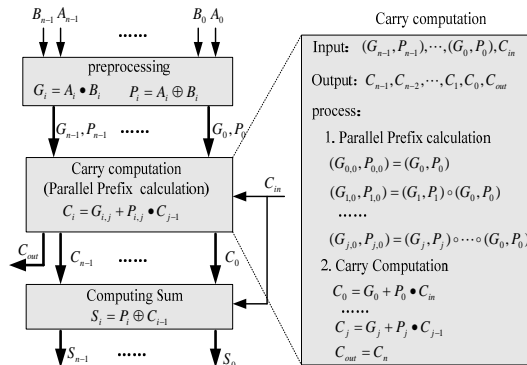


FIGURE I. THE WORK FLOW OF PARALLEL PREFIX ADDER

III. GROUPED-KOGGE-STONE TREE

In 1973, P. Kogge and H. Stone presented the Kogge-Stone tree [1] which is the fastest architecture in theory. Its fan-out is a constant in all the nodes of the carry tree, and its structure is regular and easy to be realized in VLSI. However, when N is 64, the number of prefix nodes is up to 321[2]. So, its area and power costs are expensive.

We presents Grouped-Kogge-Stone tree (GKS tree) based on Kogge-Stone tree in the following section. The principle of Parallel prefix adder based on GKS tree is similar with our previous adder presented in paper [2] which combines the advantages of parallel prefix adders and carry-select adders.

The tree presented by P. Kogge and H. Stone can be simply seen as 1-bit Grouped-Kogge-Stone tree (GKS tree). The tree needs to compute every carry bit, and the wires between correlative nodes are so long that they will become the critical path of the tree. In order to improve the speed of the tree, we can decrease nodes through dividing the tree's inputs into a few groups. Every group maybe includes N-bits. In principle, N can be any non-zero integers, but actually N has better to be 2's power, such as 2, 4, 8, 16 and so on, because with these values we can make every group has the same size and easy to be realized in VLSI. Figure II shows the architecture of a 64-bit adder's carry tree based on 4-bit GKS tree.

We can obtain the 8-bit GKS tree by combining some nodes of from 4-bit GKS tree. The reduce number of nodes is 32. Compared with the traditional Kogge-Stone tree, the 8-bit GKS tree has many advantages. It needs only 12.5% nodes of Kogge-Stone tree in the CM3, CM4 and CM5, so the wire is 12.5% as long as the traditional Kogge-Stone tree and the wire load delay is 1.56% (strictly is 1/64) of Kogge-Stone tree. The largest fan-out of GKS tree is the same as Kogge-Stone tree.

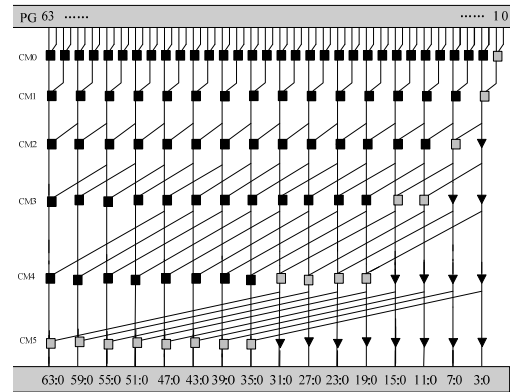


FIGURE II. 4-BIT GROUPED-KOGGE-STONE TREE OF 64-BIT ADDERS

IV. HYPER-PARALLEL PREFIX ADDER

We designed a Hyper-Parallel Prefix Adder (HPPA) by using Grouped-Kogge-Stone tree to improve the performance of the adder. In this adder, the two operands of addition are divided into several groups. As shown in Figure III, operand A is divided into j groups which are $A_0, A_1, \dots, A_i, \dots, A_{j-2}, A_{j-1}$, respectively. Accordingly, operand B should also be split into j groups which can be expressed as $B_0, B_1, \dots, B_i, \dots, B_{j-2}, B_{j-1}$.

Note that, A_i and B_i must have the same number of bits which assumed to be N_i . In the operation of A plus B , A_i and B_i must do addition operator with the carry bit which we do know yet. In order to improve the speed of the operation, we use two child parallel prefix adders to compute the sum, which is the same as carry-select adders. The select control signal comes from the GKS tree. All the groups shown in Figure III are divided again.

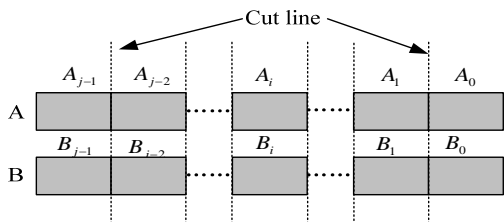


FIGURE III. DIVIDE OPERANDS A AND B

Actually, a hyper-parallel prefix adder has two levels, both of which are parallel prefix adders, which is the reason for its name. The top level combines all the lower level together.

The hyper-parallel prefix adder which has several child parallel prefix adders is the improvement of parallel prefix adder. It can fully develop parallelism inside adder, hence may significantly improve the performance of adder. In order to testify the performance of HPPA, we design a 64-bit HPPA based on 8-bit GKS tree.

V. IMPLEMENTATION OF 64-BIT HPPA

In order to verify the performance of the adder we presented, we design a 64-bit Hyper-Parallel Prefix Adder by simply dividing the two operands of adder equally. The two 64-bit operands of addition are divided into eight equal groups, so each group has 8-bit. Each of the 8-bit group is divided into 4 equal sub-groups.

Therefore, the 64-bit adder can be seen as two levels. At the bottom level of the adder, there are several 8-bit parallel prefix adders and at the top level, there is only one 64-bit adder which accepts the output of the bottom adders, other than the original operands. The remainder of this section, we will introduce the structure of each level of the adder in detail.

A. The Bottom Level of the Adder: 8-bit PPA

In our design, we use two different 8-bit parallel prefix adders, both of which are based on 2-bit GKS tree, one of them is used for non-carry bit addition and the other is used for carry addition, as shown in Figure IV. As we seen from the figure, there is only little difference between the two kinds of 8-bit PPA.

We reuse the common nodes of the two PPA to reduce the total area of the 64-bit adder. Except for the first 8-bit group of operands, each of the other groups uses both of the two kinds of 8-bit PPA once. The first group which is also called the lowest group just uses non-carry 8-bit PPA once, because our 64-bit adder doesn't has carry bit.

Both of the two 8-bit PPA have two common output signals G/P, which will be used in the top level of the 64-bit adder. The input signals of the multiplexers come from 2-bit Carry-Ripple adders. The output signals of the multiplexers are also used by the top level of the 64-bit adder.

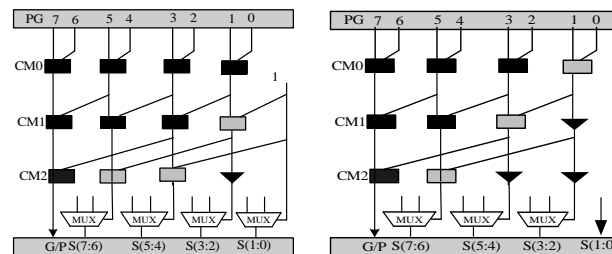


FIGURE IV. THE CARRY 8-BIT PPA AND NON-CARRY 8-BIT PPA BASED ON 2-BIT GKS TREE

B. The Top Level of the Adder

The architecture of the top level of the 64-bit adder is shown in Figure V. All of the input signals of the top level come from the bottom level.

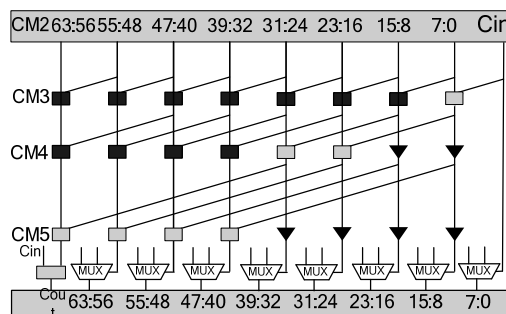


FIGURE V. TOP LEVEL OF THE ADDER

Comparing the Figure II with the Figure V, you can see that the input signals of the carry tree of the top level of the adder come from the output signals G/P, and the input of multiplexers come from the two kinds of 8-bit PPA. The first group does not need multiplexer because of non-carry bit. Of course, we can design adders support carry bit by change the carry tree without decreasing the performance of the adder. The critical path is lies in the carry tree, especially because of the wire loads delay.

C. RTL Model Design and Verification

We designed RTL model of the presented adder using Verilog HDL to verify its correctness. Then, we designed its test module to generate random stimulus, created the abstract model, compared outputs of the adder with the abstract model, and outputted the results of the comparison to a file. The RTL model of the adder has two modules, which are the bottom module and the top module respectively.

D. Implementation Circuits

In COMS process, AND gate is implemented by two level gate which are NAND and NOT, and OR gate is implemented by NOR gate cascades NOT gate. Therefore, we made

exiguous changes to the previous circuits to minimize the logical level of the adder. The changes are shown in Table I.

In our design, we use dynamic domino logical circuits to implement the P and the G, and use pass-transistor logical circuit to implement multiplexer. The other circuits are designed using static logic. The sizes of transistors in the circuits are elaborate set. In order to maximize the adder performance, we adopt skew CMOS circuits when we design the size of transistors. The skew CMOS logical circuit [12] is presented by Alexandre Solomatnikov to design noise-immune high-performance low-power static circuits.

We designed the 64-bit adder based on the structure we presented using 0.13um CMOS process. The layout of the adder we designed is shown in Figure VI. Its length is 635um, and height is 40um.

TABLE I. UNITS AND CORRESPONDING SYMBOLS

level	P	G
P&G	$\bar{P} = \bar{A}_i + \bar{B}_i$	$\bar{G}_i = \bar{A}_i \cdot \bar{B}_i$
CM0,CM2,CM4	$P = \overline{\bar{P}_i + \bar{P}_j}$	$G = \overline{\bar{G}_i \cdot (\bar{P}_i + \bar{G}_j)}$
CM1,CM3,CM5	$\bar{P} = \bar{P}_i \cdot \bar{P}_j$	$\bar{G} = \bar{G}_i + \bar{P}_i \cdot \bar{G}_j$

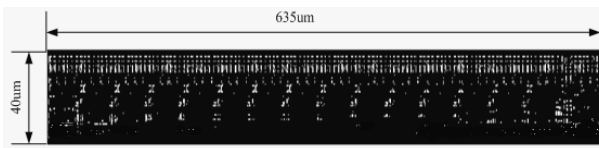


FIGURE VI. LAYOUT OF 64 BIT FULL ADDER.

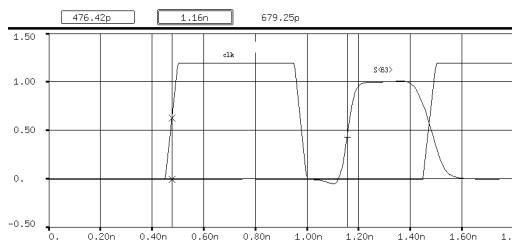


FIGURE VII. SIMULATION RESULT

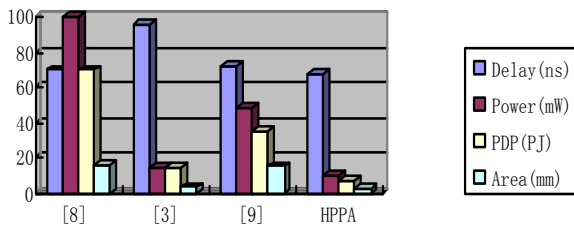


FIGURE VIII. PERFORMANCE COMPARISON

After the layout has been designed successfully, we simulate the layout under the worst case using 1.2V electrical

source. The simulation results show that the maximal delay of the proposed adder is 679ps, as shown in Figure VII. Figure VIII gives the vivid comparison of those adders. The figure also gives the detailed information about other adders designed in references [3], [8], [9], [10].

VI. CONCLUSIONS

In this paper, we present adder architecture: Hyper-Parallel Prefix Adder (HPPA) based on the Grouped-Kogge-Stone tree. It can fully develop parallelism inside adder and reduce the delay of wire loads by reducing the wire length in the critical path, hence can significantly improve the performance of adders. In the future, we will consider to improving the performance of the adder by reducing the carry tree level through other structures of nodes of the carry tree.

ACKNOWLEDGMENT

This work was supported by Natural Science Foundation of China (Grant No. 61303061) and State Key Laboratory of high performance computing (Grant No.201513-01).

REFERENCES

- [1] P. Kogge, H. Stone, IEEE Trans. Computers, 8, vol. C-22, no. 8, p. 786–793(1973).
- [2] Dong-Yu Zheng, Yan Sun, Shao-Qing Li and Liang Fang, J. Comput. Sci. & Technol., 1, p.25-27(2007).
- [3] Yan Sun, Thesis of master's degree of national university of defense technology, (2005).
- [4] Jan M.Rabaey, Anantha Chandrakasan, Borivoje Nikolic, Digaital integrated circuits, p412(2004).
- [5] R. Brent, H. Kung. IEEE Trans. Computers, vol. C-31, no. 3, pp. 260–264(1982).
- [6] J. Sklansky, Conditional-sum addition logic, IRE Trans, Electronic Computing, vol. EC-9, pp. 226-231(1960).
- [7] T. Han, D. Carlson. Proc. 8th Symp. Comp. Arith, pp. 49-56(1987).
- [8] Sun Xuguang, Mao Zhigang, Lai Fengchang. Design and Implementation of a 64bit CMOS Parallel Adder with Modified Architecture. chinese journal of semiconductors, vol 24,No.2, 2003.
- [9] A.Neve,H.Schettler,T.Ludwig,etal.Power-DelayProductMinimizationin High-Performance 64-bit Carry-Select Adders. IEEE Transactionin Very Large Scale Integration (VLSI)System,vol.12,no.3,March2004.
- [10] Xiujiang Ren, Optimized Design of 64bit GHz Integer Arithmetic and Logical Unit, Adder, Thesis of master's degree of national university of defense technology, (2007).
- [11] Xiaofei Fan, The Research and Design on 64-bit 1.47GHz High-Performance Integer Adder, Thesis of master's degree of national university of defense technology, (2008).
- [12] Alexandre Solomatnikov and et al,Skewed CMOS:Noise-Immune High-Performance Low-Power Static Circuit Family,in:Proceeding of the IEEE International Conference on Computer Design,241-246,2000
- [13] Lakshmanan, Ali Meaamar and Masuri Othman, High-Speed Hybrid Parallel-Prefix Carry-Select Adder Using Ling's Algorithm, ICSE2006 Proc, 2006.
- [14] Robert Jackson and Sunil Talwar, High Speed Binary Addition, 2004.
- [15] Giorgos Dimitrakopoulos, Dimitris Nikolos, High-Speed Parallel-Prefix VLSI Ling Adders, IEEE TRANSACTIONS ON COMPUTERS, VOL. 54, NO. 2, FEBRUARY 2005.