A Static Low-Power, High-Performance 32-bit Carry Skip Adder

Kai Chirca[1,2], Michael Schulte[1,3], John Glossner[1,2], Haoran Wang[1],
Suman Mamidi[1,3], Pablo Balzola[1], and Stamatis Vassiliadis[2]

[1]Sandbridge Technologies, Inc.
White Plains, NY 10601 USA
jglossner@sandbridgetech.com

[2]Delft University of Technology
Electrical Engineering, Mathematics
and Computer Science Department
Delft, The Netherlands

[3]University of Wisconsin
Dept. of ECE
1415 Engineering Drive
Madison, WI, 53706, USA

Abstract

In this paper, we present a full-static carry-skip adder designed to achieve low power dissipation and high-performance operation. To reduce the adder's delay and power consumption, the adder is divided into variable-sized blocks that balance the inputs to the carry chain. The optimum block sizes for minimizing the critical path delay with complementary carry generation are achieved. Within blocks, highly optimized carry look-ahead logic, which computes block generate and block propagate signals, is used to further decrease delay. The adder architecture decreases power consumption by reducing the number of logic levels, glitches, and transistors. To achieve balanced delay, input bits are grouped unevenly in the carry chain. This grouping reduces active power by minimizing extraneous glitches and transitions. The adder has been implemented in 130nm CMOS technology. At 1.2V and 25C, typical performance is 1.086GHz and power dissipation normalized to 600MHz operation is 0.786mW.

1. Introduction

Adders are fundamental arithmetic components in many computer systems, since addition is the most common arithmetic operation in a wide variety of important applications [1,2]. Consequently, several adder implementations, including ripple carry, Manchester carry chain, carry skip, carry look-ahead, carry select, conditional sum, and various parallel prefix adders are available to satisfy different area, delay, and power requirements. Descriptions of each of these adder architectures are given in [2]. Comparisons between different types of adders in terms of area, delay, and power dissipation are provided in [3-5]. As noted in [2], Ripple carry and Manchester carry chain adders are the simplest, but slowest adders with $O(n)$ area and $O(n)$ delay, where n is the operand size in bits. Carry look-ahead, conditional sum, and parallel prefix adders have $O(n \cdot log(n))$ area and $O(log(n))$ delay, but typically suffer from irregular layout. Carry skip adders, which have $O(n)$ area and $O(n^{1/2})$ delay provide a good compromise in terms of area and delay, along with a simple and regular layout [6] . Carry skip adders also dissipate less power than other adders due to their low transistor counts and short wire lengths [3,4].

Several studies have been performed to reduce the delay of carry-skip adders [6-10]. Techniques presented in [6,7] select variable block sizes to minimize the delay of adders that use a single level of carry skip logic. Techniques presented in [8-10] allow multiple levels of skip logic, which further reduces delay at the cost of an increase in area and less regular layout. Although the techniques presented in [6-10] reduce the delay of carry skip adders, they did not take into account further delay optimizations that can achieved by complementary carry generation and the use of carry look-ahead techniques within the blocks. Furthermore, these techniques were not designed to reduce the power dissipation of carry skip adders.

This paper presents the design and implementation of a full-static carry skip adder with low power consumption and low critical path delay. The adder reduces delay and power consumption by using variable-sized blocks to balance the inputs to the carry chain. Further reductions in delay are achieved by using complementary carry logic in the carry chain and by using highly optimized carry look-ahead logic to compute block generate and block propagate signals. The adder has just seven logic levels on the critical delay path, where each logic level corresponds to one complex CMOS gate. Efficient AndOrInverters (AOIs) and OrAndInverters (OAIs) are used to reduce delay and power. In Section 2, we present our Carry Skip Adder implementation. In Section 3, we present our results and compare our design with other high-speed adders. In Section 4, we give some concluding remarks.

2. 32-bit adder implementation

Different adder implementations have been developed to optimize various design parameters. Some important design parameters include propagation delay, area utilization, and power dissipation. Most adder implementations tend to trade off performance and area. Occasionally dynamic switching power is considered. In our implementation, we are concerned with minimizing dynamic switching power and short circuit power, while achieving high performance.

With carry skip adders, the linear growth of carry chain delay with the size of the input operands is improved by allowing carries to skip across blocks of bits, rather than rippling through them [2]. The idea behind our adder design is to find the optimal bit partitioning to balance the propagation delay of the inputs to the carry chain. For the less significant portion of the adder, fewer bits are combined into blocks to speed up the carry generation. As the propagation delay increases along the carry chain, more bits are combined into blocks. Our design also uses carry look-ahead logic and complementary carry generation to reduce delay.

The carry out of the j^{th} bit C_{j+1}, can be expressed as:

$$C_{j+1} = G_j + P_j \cdot C_j \quad (1)$$

where

$$G_j = A_j \cdot B_j \quad \text{generate signal} \quad (2)$$

$$P_j = A_j + B_j \quad \text{propagate signal} \quad (3)$$

Assume a block has input bits from i to j, then C_i is the carry in from the previous block. Expanding the equations above, we get:

$$C_{j+1} = G_j + P_j \cdot G_{j-1} + P_j \cdot P_{j-1} \cdot G_{j-2} + \ldots + P_j \ldots P_i \cdot C_i$$
$$= G_{block} + P_{block} \cdot C_i = G_{j:i} + P_{j:i} \cdot C_i \quad (4)$$

where

$$G_{j:i} = G_j + P_j \cdot G_{j-1} + P_j \cdot P_{j-1} \cdot G_{j-2} + \ldots + P_j \cdot P_{j-1} \ldots P_{i+1} \cdot G_i$$
$$P_{j:i} = P_j \cdot P_{j-1} \cdot \ldots \cdot P_i \quad (5)$$

Therefore, while C_i is propagating along the carry chain, $G_{j:i}$ and $P_{j:i}$ can be calculated in parallel.

In our design, the number of bits, j-i+1, involved in one block strongly depends on the propagation delay of C_i in order to guarantee that $G_{j:i}$ and $P_{j:i}$ signals have the same propagation delay. Then, the complex logic gate used to generate the block carry out, C_{j+1}, has a glitch-free output, since all three input delays are matched. Applying this rule along the carry chain optimizes both the block partitions and the logic depth on the critical path.

2.1. Top-level architecture

To favor the carry generation at the lower bits and the sum generation at the higher bits, and to balance the inputs along the carry chain, a variable-sized block partition scheme is used, as shown in Figure 1. $A_{31:0}$ and $B_{31:0}$ are two 32-bit input operands and C_0 is the carry in. $S_{31:0}$ is the 32-bit sum output of the addition operation with a carry out signal of C_{32}. The smaller blocks at the two ends of the diagram are used to speed up the carry and sum generation. The block sizes in the middle part of the adder, which are multiples of three, balance the carry chain inputs very well. The reason for selecting these block sizes is explained below for each carry skip block. In the following discussion, we assume the inputs A, B, and C_0 are fed into the adders from input registers and have the same delay at the inputs of the adder.

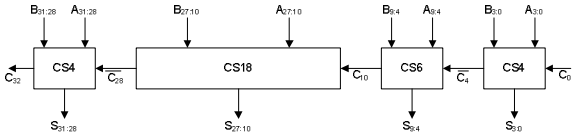

Figure 1. Block diagram of the 32-bit carry-skip adder with optimized block sizes

2.2. First 4-bit block

For the least significant bits, fast carry generation is more important than sum generation, since the sum bits are not on the critical delay path. To match the propagation delay of each carry in the carry chain, block P and G logic combines a small numbers of input bits, which means that block sizes are small. A structure similar to a ripple carry adder is used for the least significant bits because the ripple carry adder has good performance and power consumption when there are only a few adder input bits. The logic depth, however, accumulates to make larger block sizes advantageous at the more significant end.

The circuit architecture of the first 4-bit carry skip block (CS4) is shown in Figure 2. The LSB is on the right and the MSB is on the left. The delays for signals that affect the critical path delay are shown in parenthesis next to the signal's name. For example, $P_{3:2}$ (2) indicates that the block propagate signal, $P_{3:2}$, is available after two complex gate delays. The LSB addition is implemented using a standard full adder (FA). Since the sum generation is not on the critical path here, minimal size devices are used to reduce power consumption. This FA takes one complex gate delay to generate the inverted carry out, $\overline{C_1}$. Producing an inverted carry reduces the logic depth on the carry chain.

In the second FA, P and G logic operates in parallel with the carry generation in the first FA. The P and G logic computes inverted outputs

$$\overline{G_1} = \overline{A_1 \cdot B_1} \quad \text{and} \quad \overline{P_1} = \overline{A_1 + B_1} \quad (6)$$

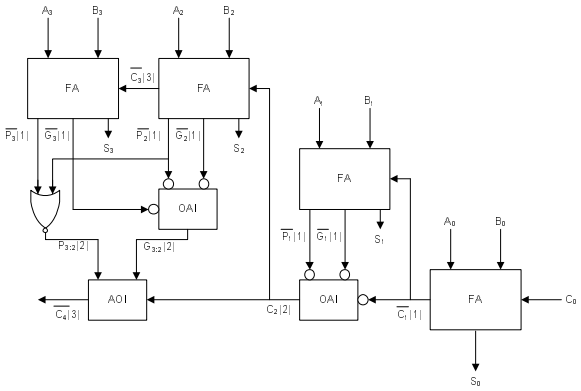

Figure 2. Block diagram of the first CS4 block

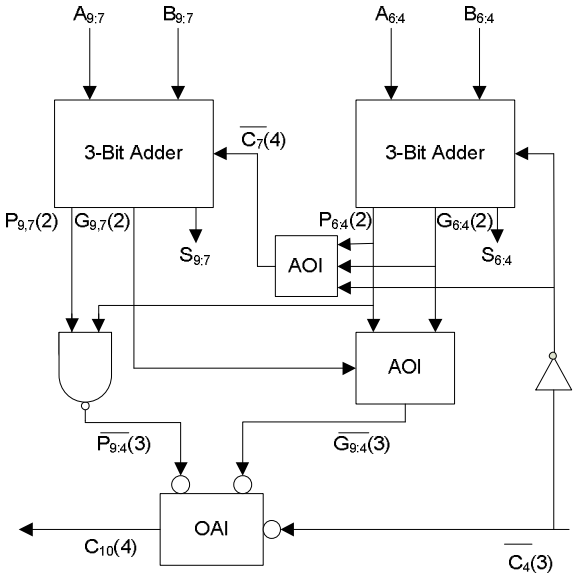

Figure 3. Block diagram of the CS6 block

As with the inverted carry, $\overline{C_1}$, we use one logic level to generate the inverted $\overline{G_1}$ and $\overline{P_1}$, instead of G_1 and P_1, to reduce the logic path length. The propagate signal is implemented using a NOR gate instead of an XNOR gate to simplify the logic without affecting the next carry generation. From Equation (1), the carry output of this bit, C_2, is expressed as:

$$C_2 = G_1 + P_1 \cdot C_0 = \overline{\overline{G_1} \cdot (\overline{P_1} + \overline{C_1})} \quad (7)$$

A simple OrAndInverter (OAI) gate takes the inverted inputs and produces the carry, C_2, after one more complex gate delay. By matching the path lengths on all the inputs of the OAI gate, glitches do not occur on its output. This assumes the NAND, NOR, and OAI gates have the same delays, which is done by properly sizing the transistors of the logic gates. As shown in Figure 2, C_2 is available after two complex gate delays.

The next block is a 2-bit adder with inputs $A_{3:2}$ and $B_{3:2}$, and carry in signal C_2. To match the two units of propagation delay on C_2, two generate and propagate signals are combined to form, $G_{3:2}$ and $P_{3:2}$. In Figure 2, the logic to produce $G_{3:2}$ and $P_{3:2}$ has two gate delays. The inverted carry out is computed as:

$$\overline{C_4} = \overline{G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot C_2} = \overline{G_{3:2} + P_{3:2} \cdot C_2} \quad (8)$$

$G_{3:2}$ and $P_{3:2}$ each are ready after two gate delays, which match the delay of the input C_2. They also have the same polarity as C_2. Since all the inputs are non-inverted, an AND-OR-INVERT (AOI) cell is used to generate the inverted carry, $\overline{C_4}$. As shown in Figure 2, $\overline{C_4}$ is available after three complex gate delays and the inputs used to compute $\overline{C_4}$ are balanced.

2.3. 6-bit block

The 6-bit carry skip (CS6) block has $\overline{C_4}$ from the previous block as the carry in signal. To match the $\overline{C_4}$ delay, G_{block} and P_{block} of this block have three levels of logic depth before entering the carry chain. This allows three pairs of bits to be used as inputs to the P and G logic. A block diagram for the 6-bit carry skip block (CS6) is shown in Figure 3.

At the top of Figure 3, the two 3-bit adders are identical. The block propagate and block generate signals, $P_{9:7}$, $G_{9:7}$, $P_{6:4}$, and $G_{6:4}$ are ready after two gate delays. Then these signals are fed into the third level P and G logic to produce $\overline{P_{9:4}}$ and $\overline{G_{9:4}}$. Notice that after 3 levels of logic, the signal outputs are all inverted, which means an OAI gate is used in the carry chain of this block and the carry out polarity is positive. The equation for C_{10} is:

$$C_{10} = \overline{\overline{G_{9:4}} \cdot (\overline{P_{9:4}} + \overline{C_4})} \quad (9)$$

Sum generations in this block require additional design consideration. If the second 3-bit adder takes the ripple carry out from the first 3-bit adder block, the subsequent sum generation will be on the critical path. To avoid ripple carries inside the block for sum generation, we use the carry skip logic to produce $\overline{C_7}$. Therefore, $P_{6:4}$ and $G_{6:4}$ from the first 3-bit block are re-used together with C_4 to produce $\overline{C_7}$ for the higher 3-bit block in order to reduce the delay path for sum generation. This way the higher 3-bit sum delay is only one gate delay greater than the lower 3-bit block delay.

By the end of the CS6 block, the 10 lower bits of the operands are added with a carry chain logic depth of four, and all the inputs along the carry chain are well balanced. Every bit input takes the same number of logic steps to the carry out C_{10}.

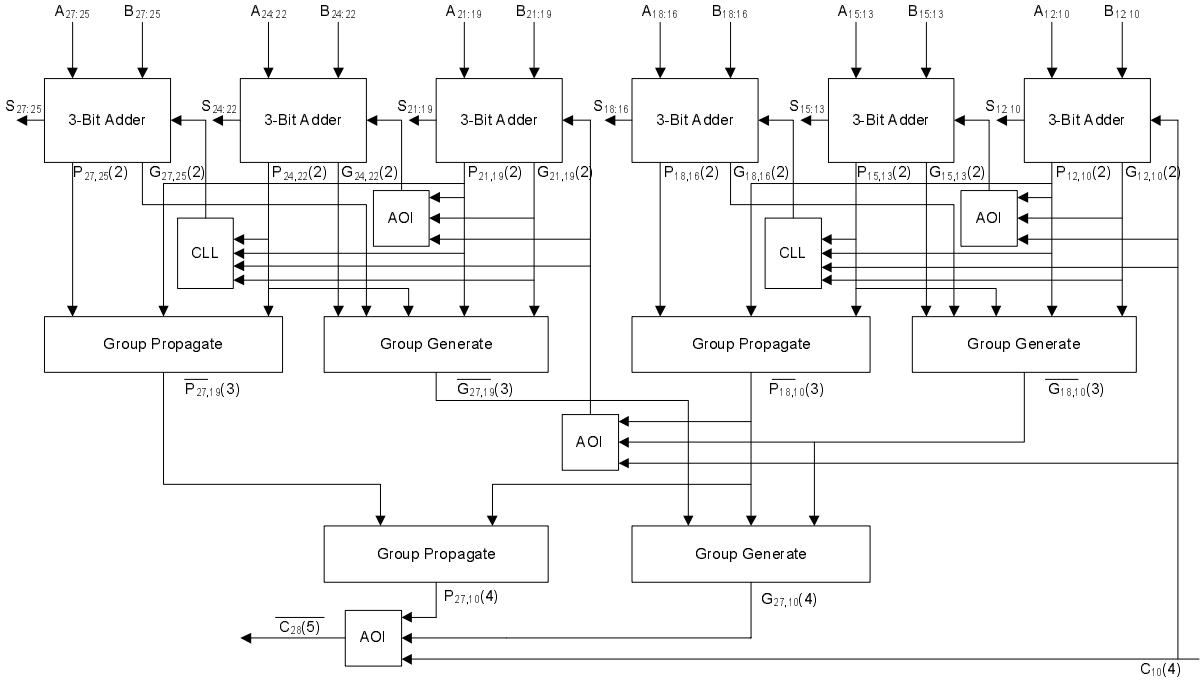

Figure 4. Block diagram of the CS18 block

Compared to the previous carry skip block, which has bits 2 and 3 joined in Figure 2, this block has 6 bits combined. Thus, a one logic level increase of P and G logic triples the number of input bits per block. The reason for this tripling comes from the usage of complex logic cells to generate $P_{6:4}$, $P_{9:7}$, $G_{6:4}$, and $G_{9:7}$. This ratio is also used in the next block.

2.4. 18-bit block

Since the carry generated by the previous block, $C_{10,}$ takes 4 levels of logic, the block P and G logic in this block has one more logic level to balance the carry chain inputs. Applying the factor of three discussed in the previous section, the total number of bits in this block is three times as large as the previous blocks, which gives a block size of 18 bits. The block diagram for the 18-bit carry skip block (CS18) is shown in Figure 4.

Notice that the sum calculations become more critical as the bit number increases. The 3-bits adders in Figure 4 use local P and G logic to balance the carry inputs for each 3-bit adder for sum generations. The capacitance on carry in C_{10} may also become high. This can be mitigated with local buffers, which are not shown in Figure 4. By the end of this block, 28 pairs of operand bits have been added, but only 5 levels of logic were used along the carry chain. Again, all critical paths are balanced to avoid glitches.

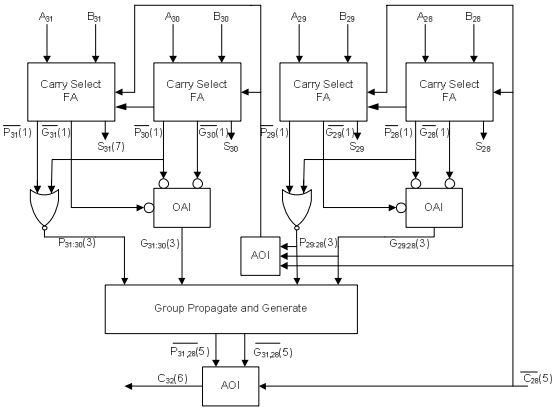

Figure 5. Block diagram of the final CS4 block

2.5. Final 4-bit block

Since there are only 4 bit inputs left in this block, and the sum generation is more critical than the carry generation, more effort is put into calculating the sum by using carry select logic to obtain the correct sum, and the same carry skip blocks to obtain the carries. The block diagram for the final 4-bit carry skip block (CS4) is shown in Figure 5. The logic depth of the final carry out, C_{32}, is equal to 6.

To balance the inputs along the carry chain with only a few operand bits, XNOR gates are employed, instead of NOR gates, to obtain the propagate signals. These signals can also be shared by the sum generators

to save hardware. Since the critical paths are well balanced on the carry chain, the critical paths of the 32-bit adder include the four sum outputs in this block. By using local carry skip adders and carry select style logic, the propagation delays of the sums are equal to 7 logic levels, which is one level more than the delay of the carry chain.

3. Results and Comparison

The adder was implemented for using 130nm CMOS technology. All input delay balancing along the carry chain was implemented in layout. To also balance wire lengths, the carry chain is put close to the block P and G logic. Full spice simulation with pessimistic wire load estimation was performed on the circuit. The adder achieves a typical frequency of 1.086 GHz at 1.2V and 25C in 130nm CMOS technology. The power dissipation for the adder when normalized to 600 MHz operation is 0.786 mW.

For comparison purposes, the delay for various adders in terms of minimum-sized fanout-of-four (FO4) inverter delays is given in Table 1. Since the 130nm process has an F04 delay of 71.2ps, the number of F04 delays for our adder is roughly 12.9. The F04 delays for the other 32-bit adders shown in Table 1 come from Table 5 of [11], assuming the other adders use inverting static CMOS. These results indicate that our adder has a critical delay path that is comparable to other high-speed adder designs. In practice, our adder is likely to be faster than reported in Table 1, since we used pessimistic wire load estimates.

Table 1. F04 delays for 32-bit adders

Adder Architecture	Number of F04 Delays
Ripple Carry	52.2
Carry Increment	27.5
Brent-Kung	13.7
Ladner-Fisher	12.9
Sklansky	21.6
Kogge-Stone	12.4
Han-Carlson	12.1
Knowles	12.7
Helper 1a from [11]	12.6
Helper 1b from [11]	11.6
Helper 1.5 from [11]	12.0
Helper 2 from [11]	11.7
Our adder	12.9

4. Conclusions

We have presented a carry skip adder implementation with balanced critical paths. To achieve balanced delay, input bits are grouped into variable-sized carry skip blocks. This grouping reduces active power by minimizing extraneous glitches. The adder has been implemented in 130nm CMOS technology and achieves a typical frequency of 1.086GHz at 1.2V and 25C. Average power dissipation normalized to 600MHz operation is 0.786mW.

References

[1] K. Uming , T. Balsara, and W. Lee, "Low-power design techniques for high-performance CMOS adders,'' *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 3, no. 2, pp. 327-333, June 1995.

[2] I Koren, *Computer Arithmetic Algorithms*, 2nd edition A. K. Peters, Ltd., Natick, MA, 2002.

[3] C. Nagendra, M. J. Irwin, and R. M. Owens, "Area-time-power tradeoffs in parallel adders,'' *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 53, no. 10, pp. 689-702, October 1996.

[4] T. K. Callaway, and E. E. Swartzlander, Jr., "Estimating the power consumption of CMOS adders,'' *Proceedings of the 11th IEEE Symposium on Computer Arithmetic*, pp. 210-216, July 1993.

[5] V. G. Oklobdzija, B. R. Zeydel, H. Dao, S. Mathew, and R. Krishnamurthy, "Energy-delay estimation technique for high-performance microprocessor VLSI adders,'' *Proceedings of the 16th IEEE Symposium on Computer Arithmetic*, pp. 272-279, June 2003.

[6] V. Kantabutra, "Designing optimum one-level carry-skip adders," *IEEE Transactions on Computers*, vol. 42, no. 6, pp. 759–764, June 1993.

[7] M. Alioto and G. Palumbo, "A simple strategy for optimized design of one-level carry-skip adders,'' *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 50, no. 1, pp. 141-148, January 2003.

[8] S. Turrini, "Optimum group distribution in carry-skip adders," in *Proceedings of the 9th IEEE Symposium on Computer Arithmetic*, pp. 96–103, September, 1989.

[9] P. Chan, M. Schlag, C. Thomborson, and V. Oklobdzija, "Delay optimization of carry-skip adders and block carry-lookahead adders using multidimensional dynamic programming," *IEEE Transactions on Computers*, vol. 41, no. 8, pp. 920-930, August 1992.

[10] V. Kantabutra, "Accelerated two-level carry-skip adders-a type of very fast adders," *IEEE Transactions on Computers*, vol. 42, no. 11, pp. 1389-1393, November 1993.

[11] D. Harris and I. Sutherland, "Logical effort of carry propagate adders," *Proceedings of the Thirty-Seventh Asilomar Conference on Signals, Systems and Computers*, vol. 1, pp. 873-878, November, 2001.