A New Design for a Lookahead Carry Generator

Hercule Kwan
Trimble Navigation
2105 Donley Drive
Austin, Texas 78758

Robert Leonard Nelson, Jr.
Trimble Navigation
2105 Donley Drive
Austin, Texas 78758

Earl E. Swartzlander, Jr.
Department of Electrical and Computer Engineering
University of Texas at Austin
Austin, Texas 78712

Abstract

We propose a new scheme for the Texas Instruments SN54182/SN74182 lookahead-carry generator. The SN54182/SN74182 provides a redundant carry output, C_{n+x}. Our re-design provides an alternative to the original design that improves the performance of adder implementations. We analyze gate delays of lookahead adders with sizes of 16, 32 and 64 bits. We examine both CMOS-ASIC vendor technology and standard TTL implementations. In all cases, gate delays of sum and carry bits are improved.

1 PROBLEM

The Texas Instruments SN54182/SN74182 lookahead-carry generator is commonly used with the SN54181/SN74181 adder to implement carry-lookahead adders. The SN54182/SN74182 generates three intermediate carries, C_{n+x}, C_{n+y}, and C_{n+z}, but no overall carry, forcing the user to use the slower carry from the appropriate SN54181/SN74181 adder. We propose a new design of the lookahead-carry generator. The pins of the SN54182/SN74182 are already fully used. The first carry, C_{n+x}, is redundant and its pin could be used for a final carry from the lookahead-carry generator that can be connected to the next cascade of the carry-lookahead adder. Using the former C_{n+x} pin for the new design reduces gate delays in adder implementations and thus improves performance.

2 INTRODUCTION

Among different implementations of high-speed adders, the carry-lookahead adder is one of the fastest. A carry-lookahead generator generates carry bits ahead of time without the need of rippling through several full or half adders, as is done in ripple-carry adders. The possibility of calculating these carry bits beforehand makes them available to next cascade of adders faster; and therefore computations of sum bits of the next cascade can take place sooner.

Assume there are two n-bit numbers A and B. They can be expressed as $A_{n-1}...A_2A_1A_0$ and $B_{n-1}...B_2B_1B_0$ respectively. Consider a four-bit ALU module. The inputs are subscripted n, n+1, n+2 and n+3, where index n is a multiple of four. We define $P_n = A_n + B_n$ and $G_n = A_nB_n$. The output carry C_{n+1} generated by a full adder with A_n, B_n and C_n as inputs is

$$C_{n+1} = G_n + P_nC_n \quad (1)$$

The equations for the next three output carries are

$$C_{n+2} = G_{n+1} + P_{n+1}G_n + P_{n+1}P_nC_n \quad (2)$$

$$C_{n+3} = G_{n+2} + P_{n+2}G_{n+1} + P_{n+2}P_{n+1}G_n + P_{n+2}P_{n+1}P_nC_n \quad (3)$$

$$C_{n+4} = G_{n+3} + P_{n+3}P_{n+2}G_{n+1} + P_{n+3}P_{n+2} \cdot P_{n+1}G_n + P_{n+3}P_{n+2}P_{n+1}P_nC_n \quad (4)$$

These equations can be found in many books on computer arithmetic [1].

There are several papers that dealt with delay optimizations of carry-lookahead adders. Chan, Schlag, Thomborson and Oklobdzija [2] examined block carry-lookahead adders (BCLA) using multidimensional dynamic programming techniques. Their goal was to minimize the worst-case delay of carries C_1 to C_{n-1} of an n-bit adder. Variable-block-size BCLAs were constructed by considering the size of the gates (fan-in) and fan-out of the signals feeding these gates. A gate delay model was then set up to be optimized by a SPARCstation.

The M-valued carry-lookahead adder and its variants were proposed by Ling, Manzoul and Hu [3]. This concept introduces a binary term Ti which is defined as

1058-6393/95 $4.00 © 1995 IEEE

$$T_i = P_i \oplus G_i \quad (5)$$
$$= a_i \oplus b_i \quad (6)$$

Then equation (1) becomes

$$C_{i+1} = G_i + T_i C_i \quad (7)$$

Hu claimed that P_i is usually more expensive to evaluate than T_i in binary. Hu proposed a few improved m-valued carry-lookahead adders based on Ling's variation of the conventional carry-lookahead adder.

In another paper, Lee and Oklobdzija studied optimization of carry-lookahead structure by varying group sizes and lookahead levels [4]. They proposed a delay model by taking into account the fan-in and fan-out of the circuits. They also considered BCLAs.

All three aforementioned papers deal with the problem of minimizing gate delays in carry-lookahead structures. The discussions in [2] and [4] are similar. But none of them considers the possibility of modifying current hardware. The conventional carry-lookahead generator seems to have an oversight in its original design and should be adjusted. In this paper we intend to correct the existing logic for the SN54182/SN74182 and reduce gate delays in carry-lookahead adder implementations.

3 APPROACH

The SN54182/SN74182 provides logic for calculating the output carries defined in equation (1), (2) and (3). Their names are C_{n+x}, C_{n+y}, and C_{n+z} respectively. To implement a 64-bit carry-lookahead adder, three levels of carry-lookahead logic are needed. It is desired to find a new method to reduce gate delays in carry-lookahead adders.

First we observed that the carry bit C_{n+x} is the same as the ripple carry produced by the first four-bit module in a 16-bit carry-lookahead adder as shown in Figure 1. This means that this carry bit C_{n+x} is redundant and that its pin could be used for other purposes.

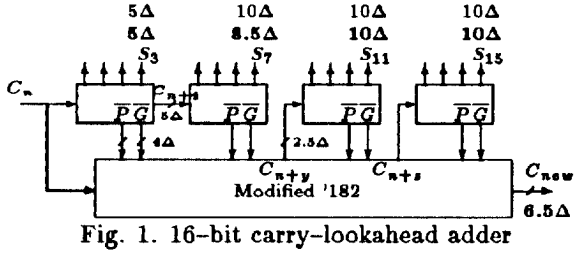

Fig. 1. 16-bit carry-lookahead adder

Since the second four-bit module gets the carry input from the previous module directly, the C_{n+x} output can be removed from the SN54182/SN74182.

Figure 2 shows a 32-bit carry-lookahead adder. The first 16-bit carry-lookahead module provides a carry bit into the second 16-bit module. In the conventional design, this carry is calculated by the last 4-bit adder module which is produced by adding the two input bits and C_{n+z}. The summation results in a sum bit and a carry bit. This carry bit is then fed into the next 16-bit carry-lookahead module and its first 4-bit adder module. The equation for this carry bit is the same as the one for C_{n+4}.

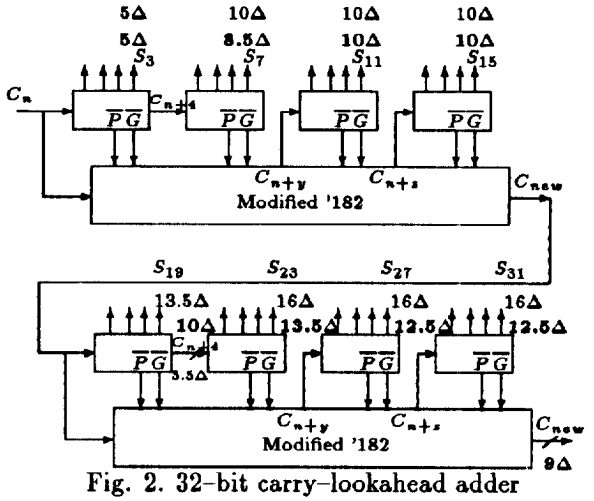

Fig. 2. 32-bit carry-lookahead adder

It is possible to construct logic to calculate C_{n+4} within the same carry-lookahead generator. The pin originally designated as C_{n+x} can be used to put out C_{n+4}. In this paper, this new output is termed C_{new}.

We obtained the logic equation for C_{new} by first considering that C_{new} is equal to the C_{n+4} that is defined in equation (4). After that, the equation has to be transformed into a form that can be fitted into the existing logic of the SN54182/SN74182. The following is part of the logic equation for C_{new}.

$$\begin{aligned} C_{new} &= G_b + P_b C_n \\ &= G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 \\ &\quad + P_3 P_2 P_1 P_0 C_n \\ &= G_3 + P_3(G_2 + (P_2(G_1 + P_1(G_0 + P_0 C_n))) \\ &\neq \overline{G_3}(\overline{P_3} + (\overline{G_2}(\overline{P_2} + \overline{G_1}(\overline{P_1} + \overline{G_0}(\overline{P_0} + \overline{C_n}))))) \\ &= \overline{G_3}\,\overline{P_3} + \overline{G_3}\,\overline{G_2}\,\overline{P_2} + \overline{G_3}\,\overline{G_2}\,\overline{G_1}\,\overline{P_1} \\ &\quad + \overline{G_3}\,\overline{G_2}\,\overline{G_1}\,\overline{G_0}(\overline{P_0} + \overline{C_n}) \\ &= \overline{G_3}\,\overline{P_3} + \overline{G_3}\,\overline{G_2}\,\overline{P_2} + \overline{G_3}\,\overline{G_2}\,\overline{G_1}\,\overline{P_1} \\ &\quad + \overline{G_3}\,\overline{G_2}\,\overline{G_1}\,\overline{G_0}\,\overline{P_0} + \overline{G_3}\,\overline{G_2}\,\overline{G_1}\,\overline{G_0}\,\overline{C_n} \end{aligned} \quad (8)$$

A detailed derivation of the above equation is given in the Appendix. Transformed expressions of the other original carries, C_{n+y} and C_{n+z}, as well as P and G, are also explicitly provided.

Figure 5 is a diagram of a 64-bit carry lookahead adder using five modified SN54182/ SN74182's. Equation (8) shows that two five-input AND gates and

a five–input NOR gate are needed to implement the logic for C_{new}. According to several high–speed–logic data books, the propagation delay times of 8–input (AND, OR, or NOR) gates do not differ much from the ones of 4–input (AND, OR, or NOR) gates [6, 7, 8]. When the number of inputs increases to 13, propagation delay times begin to rise significantly due to switching logic. Therefore there is a trade off between speed and the number of inputs of a logic gate. For this above reason, we conclude that a 5–input gate would have about the same propagation delay time as 4–input gate has. This fact holds true in the case of TTL logic gates. In actual implementations of CMOS logic gates, however, we have to consider the effects of fan–in and fan–out. Let us look at CMOS logic first. We used a delay model which is derived in a similar fashion as the one in Lee and Oklobdzija [4]. The same CMOS–ASIC vendor is chosen but the technology we used is newer than the one described in their paper [5]. In their derivations, the propagation delay times t_{PLH} and t_{PHL} were averaged to provide the constants for the equations. For CMOS gates, the low–to–high delay is longer than the high–to–low delay due to asymmetry in the conductivities of the N–channel and P–channel transistors, which are typically in the ratio 3:1. We felt that it is not correct to average these two times because we always want to assume the worst case scenario. Therefore we used the t_{PLH} propagation delay time to derive our delay model. The equations for the delay model are as follows:

$$t_{NOR} = 0.07558 + 0.1173fi + (0.00152 + 0.06159fi)fo \quad (9)$$

$$t_{INV} = 0.2319 + 0.0652fo \quad (10)$$

$$t_{XOR} = 0.6313 + 0.0643fo \quad (11)$$

Equation (9) is valid for fan–in of 2, 3, and 4. For fan–in of 5, a different equation is used:

$$t_{NOR} = 0.8815 + 0.0646fo \quad (12)$$

Notice that in our new design, the new logic for C_{new} is obtained by five AND–gates into a five–input NOR gate. Two of the five AND–gates are five–input gates. We claim that the time delay of this logic is the same as a five–input NOR. This is due to the fact that the worst case occurs when all five p–channel transistors that are in series pull up simultaneously, which is what happens in a five–input NOR gate also. On the contrary, for five n–channel transistors pulling down in series, the time delay is much shorter. Therefore in our calculations we assume an AND–OR–INVERT logic structure that has AND gates with maximum input size of i to have the same low–to–high delay time as an i–input NOR gate.

Tables 1, 2 and 3 show the comparisons of gate delays of the original implementation and our new design using CMOS logic. The numbers are normalized to unit delay of an inverter.

	S_3	S_7	S_{11}	S_{15}	C_{16}
Original delay	7.8	12.7	12.9	13.5	12.9
New delay	7.8	11.6	12.9	13.5	8.9

Table 1. 16–bit carry–lookahead adder delay time comparisons

	S_3	S_7	S_{11}	S_{15}	S_{19}
Original delay	7.8	12.7	12.9	13.5	17.4
New delay	7.8	11.6	12.9	13.5	13.5

	S_{23}	S_{27}	S_{31}	C_{32}
Original delay	20.6	20.7	21.4	20.8
New delay	17.3	16.9	17.5	12.9

Table 2. 32–bit carry–lookahead adder delay time comparisons

	S_3	S_7	S_{11}	S_{15}	S_{19}
Original delay	7.8	12.7	12.9	13.5	15.7
New delay	7.8	11.6	12.9	13.5	13.5

	S_{23}	S_{27}	S_{31}	S_{35}	S_{39}
Original delay	18.9	19.1	19.8	15.9	19.1
New delay	17.3	16.8	17.5	15.9	19.1

	S_{43}	S_{47}	S_{51}	S_{55}	S_{59}
Original delay	19.3	19.9	16.5	19.8	19.9
New delay	19.3	19.9	16.5	19.8	19.9

	S_{63}	C_{64}
Original delay	20.6	19.9
New delay	20.6	11.9

Table 3. 64–bit carry–lookahead adder delay time comparisons

Secondly we look at TTL logic. Figure 3 shows the delay time versus the number of inputs of the Texas Instruments ALS74 series of NAND gates [6]. The gates shown in this figure are the '00, '10, '20, '30 and '133 (2, 3, 4, 8 and 13 inputs respectively). By observing the curve for t_{PHL}, we see that the delay time of the 13–input gate is twice the value of the 8–input gate. Since the propagation delay times of an 8–input gate and a 4–input gate are about the same, we know that the propagation delay time of a 5–input gate must almost be the same also.

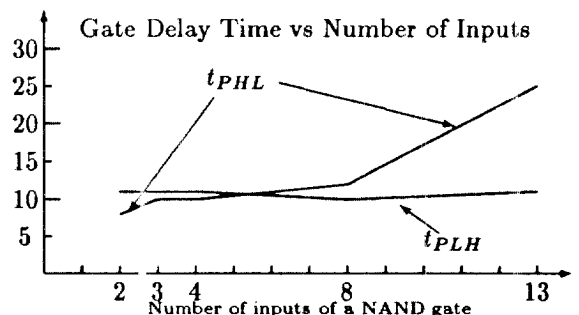

Fig. 3. Delay time vs. number of inputs of 74ALS series NAND gates

The logic diagram of the modified SN54182/ SN74182 is shown in Figure 4. The upper dashed box highlights the newly added logic for C_{new} and the lower dashed box indicates the deleted logic of C_{n+x}. Observe that in the original design, logic gates of input size four are used and gate delays of each of the original outputs are 2.5Δ, where Δ is the delay time of one gate. The gate delays of the newly added logic for C_{new} are also 2.5Δ.

There is one thing we have to mention here. We assume that the gate delays of G and P are almost the same and have 1Δ. According to the data sheets for the SN54181/SN74181 arithmetic logic units (ALU), from any A or B (inputs) to G or P, the propagation delay times are almost the same. Since it takes only 1Δ to form P, we may assume that the gate delay to form G is also 1Δ. Looking further at the propagation delay times of a NOR gate and of a NAND gate from the LS family, we see that the propagation time of an AND–OR–NOT gate is about 1.5 times bigger than that of either a NOR gate or a NAND gate. Since P is formed by a NAND gate, it is reasonable for us to assume that G has a gate delay of about 1.5Δ. This is the reason why we marked 4Δ[1] for G and P (from the '181 at the first level) in Figure 1.

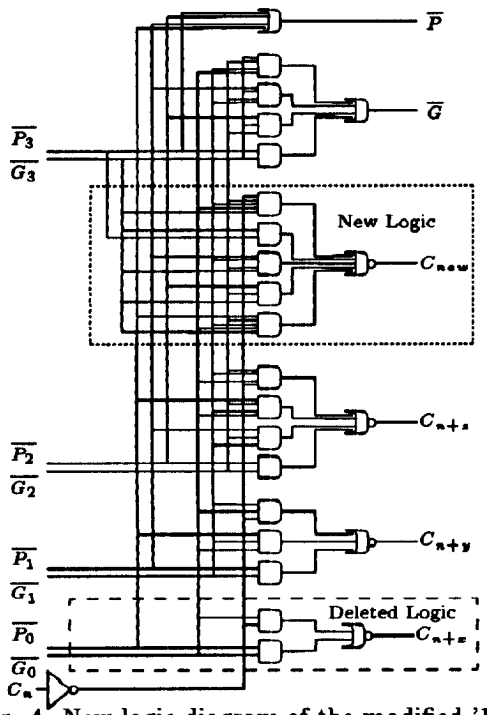

Fig. 4. New logic diagram of the modified '182

This is a very significant improvement. C_{new}, which is equivalent to C_{n+4}, is generally obtained by passing $\overline{P}$ and $\overline{G}$ into the next higher level of carry–lookahead generator and combining with C_n carry input. It takes 4Δ to generate $\overline{P}$ and $\overline{G}$ and it then takes 2.5 more gate delays to generate C_{n+4} by combining $\overline{P}$ and $\overline{G}$ with C_n. In the new design, C_{new} is available at the same time as $\overline{P}$ and $\overline{G}$ are. This means it is not required for $\overline{P}$ and $\overline{G}$ to go the next higher level of lookahead logic to form the carry that is later to be fed into the next cascade of adders. The availabilty of C_{new} at this level means a total of 2.5Δ is saved to produce the same carry.

[1]AND-OR-NOT + AND-OR-NOT + NOT = 1.5Δ + 1.5Δ + 1Δ = 4Δ.

4 RESULTS

CMOS–ASIC Implementations: In the 16–bit lookahead adder, the sum bits of the second 4–bit ALU module save about 1.1Δ while C_{16} saves about 4Δ. In the 32–bit lookahead adder, at most 3.9Δ are saved in some of the sum bits and C_{32} saves 7.9Δ. In the 64–bit lookahead adder, at most 2.3Δ are saved in some of the sum bits. Some of the sum bits have a slight increase in delay but still do not exceed the maximum delay of the sum bits. C_{64} saves 8Δ.

TTL Implementations: Figure 1, 2 and 5 show the block diagrams of 16–bit, 32–bit and 64–bit carry–lookahead adders. Gate delays are marked on all important bit outputs. The upper row of each group of numbers represents their original gate delays and the bottom row indicates their new values.

In the 16–bit lookahead adder, the gate delays of the second 4–bit ALU module range from 8.5Δ to 10Δ, which saves 1.5Δ. In the 32–bit lookahead adder, at most 3.5Δ are saved in some of the bits and in most cases 1.5Δ. In the 64–bit lookahead adder, some of the sum bits drop from 15Δ to 12.5Δ.

In none of the cases above do the gate delays exceed the maximum attained by the original implementations. In all three implementations, gate delays have dropped. The new design of the SN54182/SN74182 appears to have improved on the conventional design.

5 CONCLUSIONS

A new design of the SN54182/SN74182 lookahead–carry generator is proposed. It overcomes a flaw of the conventional design. The C_{n+x} output is removed from this pin and replaced by a new output carry defined in equation (4). Gate delays of lookahead adders with sizes of 16, 32 and 64 bits are analyzed. In all cases, gate delays of sum bits are improved. This design successfully provides an alternative to the original

design and improves the performance of adder implementations.

6 APPENDIX

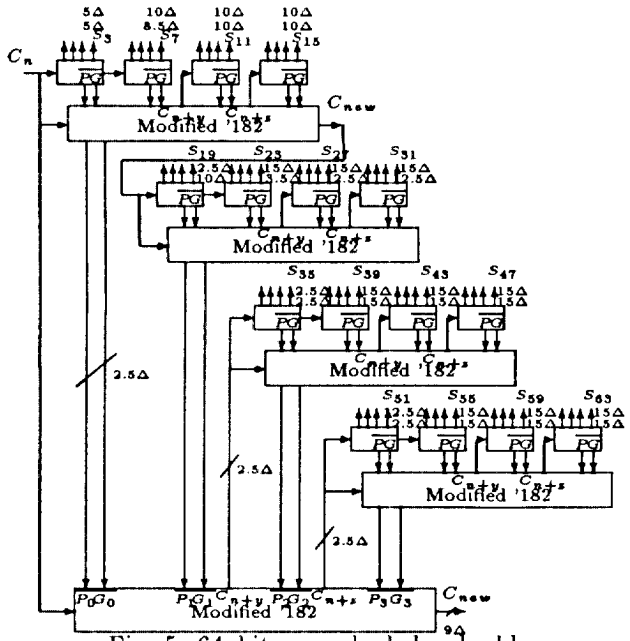

Fig. 5. 64-bit carry-lookahead adder

$$
\begin{aligned}
C_{new} &= G_b + P_b C_n \\
&= G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_n \\
&= G_3 + P_3 G_2 + P_3 (P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_n) \\
&= G_3 + P_3 G_2 + P_3 (P_2 (G_1 + P_1 G_0 + P_1 P_0 C_n)) \\
&= G_3 + P_3 G_2 + P_3 (P_2 (G_1 + P_1 (G_0 + P_0 C_n))) \\
&= G_3 + P_3 G_2 + P_3 (P_2 (G_1 + (\overline{P_1} + \overline{G_0}(\overline{P_0} + \overline{C_n}))')) \\
&= G_3 + P_3 G_2 + P_3 (P_2 (\overline{G_1}(\overline{P_1} + \overline{G_0}(\overline{P_0} + \overline{C_n})))') \\
&= G_3 + P_3 G_2 + P_3 (P_2 (\overline{G_1}\,\overline{P_1} + \overline{G_1}\,\overline{G_0}\,\overline{P_0} + \overline{G_1}\,\overline{G_0}\,\overline{C_n})') \\
&= G_3 + P_3 (G_2 + (\overline{P_2} + \overline{G_1}\,\overline{P_1} + \overline{G_1}\,\overline{G_0}\,\overline{P_0} + \overline{G_1}\,\overline{G_0}\,\overline{C_n})') \\
&= G_3 + P_3 (\overline{G_2}(\overline{P_2} + \overline{G_1}\,\overline{P_1} + \overline{G_1}\,\overline{G_0}\,\overline{P_0} + \overline{G_1}\,\overline{G_0}\,\overline{C_n}))' \\
&= (\overline{G_3}\,(\overline{P_3} + \overline{G_2}\,\overline{P_2} + \overline{G_2}\,\overline{G_1}\,\overline{P_1} + \overline{G_2}\,\overline{G_1}\,\overline{G_0}\,\overline{P_0} \\
&\quad + \overline{G_2}\,\overline{G_1}\,\overline{G_0}\,\overline{C_n}))' \\
&= (\overline{G_3}\,\overline{P_3} + \overline{G_3}\,\overline{G_2}\,\overline{P_2} + \overline{G_3}\,\overline{G_2}\,\overline{G_1}\,\overline{P_1} \\
&\quad + \overline{G_3}\,\overline{G_2}\,\overline{G_1}\,\overline{G_0}\,\overline{P_0} + \overline{G_3}\,\overline{G_2}\,\overline{G_1}\,\overline{G_0}\,\overline{C_n})' \qquad (13)
\end{aligned}
$$

The equation for C_{n+y} is as follows:

$$
\begin{aligned}
C_{n+y} &= G_1 + P_1 G_0 + P_1 P_0 C_n \\
&= G_1 + P_1 (G_0 + P_0 C_n) \\
&= (\overline{G_1}(\overline{P_1} + \overline{G_0}(\overline{P_0} + \overline{C_n})))' \\
&= (\overline{G_1}(\overline{P_1} + \overline{G_0}\,\overline{P_0} + \overline{G_0}\,\overline{C_n}))' \\
&= (\overline{G_1}\,\overline{P_1} + \overline{G_1}\,\overline{G_0}\,\overline{P_0} + \overline{G_1}\,\overline{G_0}\,\overline{C_n})' \qquad (14)
\end{aligned}
$$

The equation for C_{n+z} is as follows:

$$
\begin{aligned}
C_{n+z} &= G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_n \\
&= G_2 + P_2 (G_1 + P_1 (G_0 + P_0 C_n)) \\
&= (\overline{G_2}(\overline{P_2} + \overline{G_1}\,\overline{P_1} + \overline{G_1}\,\overline{G_0}\,\overline{P_0} + \overline{G_1}\,\overline{G_0}\,\overline{C_n}))' \\
&= (\overline{G_2}\,\overline{P_2} + \overline{G_2}\,\overline{G_1}\,\overline{P_1} + \overline{G_2}\,\overline{G_1}\,\overline{G_0}\,\overline{P_0} \\
&\quad + \overline{G_2}\,\overline{G_1}\,\overline{G_0}\,\overline{C_n})' \qquad (15)
\end{aligned}
$$

The equation for G is

$$G = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 \qquad (16)$$

and the equation for P is

$$P = P_3 P_2 P_1 P_0 \qquad (17)$$

7 REFERENCES

[1] J. Y. Cheung and J. G. Bredeson, *Modern Digital Systems Design*, West Publishing Company, pp. 160–164, 1990

[2] P. K. Chan, M. D. F. Schlag, C. D. Thomborson, and V. G. Oklobdzija, "Delay Optimization of Carry-Skip Adders and Block Carry-Lookahead Adders," *Proceedings of the 10th IEEE Symposium on Computer Arithmetic*, Piscataway, NJ, USA., pp. 154–164.

[3] Z. Hu, "Variants of an improved m–valued carry look–ahead adder," *International Journal of Electronics*, Vol. 71, 1991, pp. 799–803.

[4] B. D. Lee and V. G. Oklobdzija, "Optimization and Speed Improvement Analysis of Carry–Lookahead Adder Structure," *Twenty–fourth Asilomar Conference on Signals, Systems and Computers*, v. 2, pp. 918–922.

[5] *0.7–Micron Array–Based Products Databook*, LSI Logic Corporation, January 1993.

[6] *Texas Instruments ALS/AS Logic Data Book*, Texas Instruments Inc., 1986.

[7] *National Semiconductor FAST Advanced Schottky TTL Logic Databook*, National Semiconductor Corporation, 1990.

[8] *Motorola CMOS Logic Data book*, Motorola Inc., 1991.