High Speed Binary Addition

Robert Jackson and Sunil Talwar

***Abstract*— Addition of two binary numbers is a fundamental operation in electronic circuits. Applications include arithmetic logic unit, floating-point operations and address generation. It is widely accepted that there is no single best adder implementation. Modern adder architectures utilize a hybrid scheme based on, among others, various parallel prefix, carry select and Ling architectures.**

The parallel prefix method implements logic functions which determine whether groups of bits will generate or propagate a carry. These functions are hierarchically combined to calculate the carry into any bit. Ling adders reduce delay by using a simplified version of the group generate. However, the method only reduces complexity at the first level; all subsequent combinations in the hierarchy have the same complexity as the parallel prefix method. In this article we present novel architectures which have reduced complexity at all levels.

I. Introduction

Addition of two binary numbers is the simplest and arguably the most important of arithmetic operations. Consequently much attention has been devoted to optimizing adder delay. Several architectures have been proposed, examples include carry-lookahead [9], conditional sum [8], carry-select [1], parallel prefix [5], [6], [4] and Ling [7]. Modern adder architectures utilize a hybrid scheme. The parallel prefix and Ling methods share the property that the generate term is much more complex than the propagate term, this provides some hope of providing speedup. We are naturally led to the question of whether there exist functions of more balanced complexity each of which is simpler than the generate function. In this article we shall show that this is indeed the case. We will derive reduced generate functions which are a generalization of the Ling pseudo carry.We will also find that these functions have natural propagate counter parts, which we shall call hyper propagate. The reduced generate and hyper propagate functions are of similar complexity with each function being of a lower complexity than the generate or pseudo carry functions.

The article is focused on constructing the carry. We are currently investigating the families of adders that can be derived from the reduced generate and hyper propagate functions.

The article is organized as follows. The next section outlines and compares the parallel prefix and Ling approach to computing the carry. We also isolate the two main ingredients in Ling architectures, factorization to define the simpler generate, pseudo-carry, and recursion to compute any carry in a tree architecture. The third section shows that Ling's factorization is but one among many possible factorizations. Using these new factorizations we show that there is a non-intuitive logic equation for computing any carry. We also introduce the reduced generate functions. In the fourth section we combine the parallel prefix equation with the newly derived equation to derive recursive formulation. This derivation naturally leads us to hyper propagate functions. We also show that there are recursion formulas for these. We then compare the complexity of our functions with that of parallel prefix at various radices. In the last section we give some examples.

II. Parallel Prefix and Ling adders

We will consider the addition of two binary numbers $x_n x_{n-1} \dots x_0$ and $y_n y_{n-1} \dots y_0$. Also $F_{j:k}$ signifies a function of the the bits $x_j x_{j-1} \dots x_k$ and $y_j y_{j-1} \dots y_k$.

Parallel prefix architectures are based on two logic functions, called generate and propagate. The first stage of the adder forms bit-generate (g) and bit-propagate (p) functions for each bit:

$$g_i = x_i y_i, \quad p_i = x_i + y_i \tag{1}$$

These functions are combined to form group generate and group propagate functions according to the well known equations:

$$G_{n:0} = G_{n:k} + P_{n:k} G_{k-1:0}, \quad P_{n:0} = P_{n:k} P_{k-1:0} \tag{2}$$

This is the binary tree method. Higher radix formulations are possible, for ternary trees the equations are:

$$\begin{aligned} G_{n:0} &= G_{n:k} + P_{n:k} G_{k-1:k'} + P_{n:k} P_{k-1:k'} G_{k'-1:0}, \\ P_{n:0} &= P_{n:k} P_{k-1:k'} P_{k'-1:0} \end{aligned} \tag{3}$$

The drawback of higher radix methods is that although the number of combining levels is fewer, the complexity of the logic functions at each level is greater.

Ling observed a variation of the above, which allows for a speed up on the parallel prefix method. We will see that our method is a generalization of Ling's approach and so we present it in some detail. Ling observed that if the delay of the carry term could be reduced by increasing the delay of some other term, the overall delay will be reduced as long as the carry term is still on the critical path. There are two components to Ling's method. First, he defines a function of lower complexity than the generate function. Ling noted that every term in

$$G_{n:0} = g_n + p_n g_{n-1} + \dots + p_n p_{n-1} \dots p_1 g_0 \tag{4}$$

contains p_n except for the very first term, which is simply g_n. However, $g_n = p_n g_n$ and so p_n can be factored out of $G_{n:0}$ to create a pseudo-carry $H_{n:0}$, where

$$G_{n:0} = p_n H_{n:0}, \quad H_{n:0} = g_n + G_{n-1:0} \tag{5}$$

The function $H_{n:0}$ is simpler than the function $G_{n:0}$, the fan-in of each AND-gate is reduced by one. The second ingredient of Ling's method is recursion. Ling showed that the pseudo carry $H_{j:i}$ of a group could be constructed from the pseudo carries $H_{j:k}$ and $H_{k-1:i}$ of sub groups:

$$\begin{aligned} H_{j:i} &= g_j + G_{j-1:i} \\ &= g_j + G_{j-1:k} + P_{j-1:k}G_{k-1:i} \\ &= [g_j + G_{j-1:k}] + P_{j-1:k}p_{k-1}[g_{k-1} + G_{k-2:i}] \\ &= H_{j:k} + P_{j-1:k-1}H_{k-1:i} \end{aligned} \tag{6}$$

The Ling adder does have the problem that to produce the actual carry out, the p_j and $H_{j:i}$ need to be combined. This extra delay can however be eliminated by noting that the critical path for a n-bit adder is in producing the $n-1^{th}$ bit which can be expressed as:

$$\begin{aligned} s_{n-1} &= x_{n-1} \oplus y_{n-1} \oplus G_{n-2:0} \\ &= x_{n-1} \oplus y_{n-1} \oplus p_{n-2}H_{n-2:0} \end{aligned} \tag{7}$$

But p_{n-2} can be computed faster than $H_{n-2:0}$ and so a multiplexer can be used:

$$\begin{aligned} s_{n-1} = &\overline{H_{n-2:0}}(x_{n-1} \oplus y_{n-1}) \\ &+ H_{n-2:0}(x_{n-1} \oplus y_{n-1} \oplus p_{n-2}) \end{aligned} \tag{8}$$

Just as for the parallel prefix method, more than two groups can be combined, for a ternary tree the pseudo carry equation is:

$$\begin{aligned} H_{j:i} = &H_{j:k} + P_{j-1:k-1}H_{k-1:k'} \\ &+ P_{j-1:k-1}P_{k-2:k'-1}H_{k'-1:i} \end{aligned} \tag{9}$$

We note that Ling's method only reduces complexity at the first level, we see that equation (6) is of the same complexity as the generate function in equation (2) and the same holds for higher radix architectures.

III. Factorizations and Reduced generate Functions

In this section we show that the Ling factorization is but one of many possible. This will allow us to define reduced generate functions, which are our counter parts to the Ling pseudo carry. As an example we consider the function:

$$\begin{aligned} G_{4:0} &= g_4 + p_4g_3 + p_4p_3g_2 + p_4p_3p_2g_1 + p_4p_3p_2p_1g_0 \\ &= p_4[g_4 + g_3 + p_3g_2 + p_3p_2g_1 + p_3p_2p_1g_0] \\ &= [g_4 + p_4p_3][g_4 + g_3 + g_2 + p_2g_1 + p_2p_1g_0] \\ &= [g_4 + p_4g_3 + p_4p_3p_2][g_4 + g_3 + g_2 + g_1 + p_1g_0] \end{aligned} \tag{10}$$

The first equality is the Ling factorization, the bracketed expression being the Ling pseudo carry. We will now see that there is a general factorization, to describe this we need some logic functions:

$$\begin{aligned} D_{j:k} &= G_{j:k} + P_{j:k} \\ &= G_{j:k+1} + P_{j:k} \\ B_{j:k} &= g_j + g_{j-1} + ... + g_k \end{aligned} \tag{11}$$

In passing we note that the function $D_{j:k}$ is high if the addition $x_jx_{j-1}\ldots x_k + y_jy_{j-1}\ldots y_k + 1$ produces a carry out and the function $B_{j:k}$ is high if a carry is generated in any bit position in the range $j : k$. We leave it to the readers to convince themselves that in general:

$$G_{j:i} = D_{j:k}[B_{j:k} + G_{k-1:i}] \tag{12}$$

Note that if $j-k=1$ then this equation reduces to the usual Ling factorization. We thus have a whole host of reduced complexity generate functions $B_{j:k} + G_{k-1:i}$ which we shall denote by $R^{(j-k+1)}_{j:i}$, the subscript indicates the input bits and the superscript represents the range of B. Thus a Ling pseudo carry would be represented as $R^{(1)}_{j:i}$, in our notation.

IV. Recursion

The second ingredient of Ling's formulation is recursion, that is, a pseudo carry over a group can be constructed in terms of pseudo carries over sub groups. We shall now see that this is also possible for our reduced generate function. As an example we will construct a radix three implementation, higher radix implementations are also possible and we leave their derivation to the reader. The derivation relies on both the logic equations:

$$G_{j:i} = D_{j:k}[B_{j:k} + G_{k-1:i}] \tag{13}$$

$$G_{j:i} = G_{j:k} + P_{j:k}G_{k-1:i} \tag{14}$$

Recall that we wish to construct a reduced generate function over a group from reduced generate functions over subgroups. We first divide the group of bits $n-1:0$ into three "equal sized" subgroups $n-1:k$, $k-1:k'$ and $k'-1:0$. We further choose m and m' to be "midpoints" of the second and third groups respectively. Now consider the reduced generate function:

$$R^{(n-m)}_{n-1:0} = B_{n-1:m} + G_{m-1:0} \tag{15}$$

We first use equation (14) to decompose $G_{m-1:0}$. We thus have:

$$R^{(n-m)}_{n-1:0} = B_{n-1:m} + G_{m-1:k'} + P_{m-1:k'}G_{k'-1:0} \tag{16}$$

We now note the $B_{n-1:m}$ is simply an OR of bit-level generates and so we can rewrite:

$$\begin{aligned} R^{(n-m)}_{n-1:0} = &[B_{n-1:k}] + [B_{k-1:m} + G_{m-1:k'}] \\ &+ P_{m-1:k'}G_{k'-1:0} \end{aligned} \tag{17}$$

The two bracketed terms are reduced generate functions, we still need to eliminate the last generate function. We now make use of equation (13):

$$G_{k'-1:0} = D_{k'-1:m'}[B_{k'-1:m'} + G_{m'-1:0}] \tag{18}$$

Substituting this into (17) we have:

$$\begin{aligned} R^{(n-m)}_{n-1:0} = &B_{n-1:k} + [B_{k-1:m} + G_{m-1:k'}] \\ &+ [P_{m-1:k'}D_{k'-1:m'}][B_{k'-1:m'} + G_{m'-1:0}] \end{aligned} \tag{19}$$

which has the desired form:

$$R^{(n-m)}_{n-1:0} = R^{(n-k)}_{n-1:k} + R^{(k-m)}_{k-1:k'} + [P_{m-1:k'}D_{k'-1:m'}]R^{(k'-m')}_{k'-1:0} \tag{20}$$

We have shown that a reduced generate function over a group can be recursively constructed from reduced generate functions over subgroups. We are however faced with the problem that the function $P_{m-1:k'}D_{k'-1:m'}$, which we shall refer to as hyper propagate, also needs to be constructed recursively. The construction is along similar lines to the above, we will show that we can construct a hyper propagate function over a group from hyper propagate and reduced generate functions over sub groups. We shall need the following self evident equations:

$$D_{j:i} = D_{j:k}[B_{j:k} + D_{k-1:i}] \tag{21}$$

$$D_{j:i} = G_{j:k} + P_{j:k}D_{k-1:i} \tag{22}$$

For generality consider the function:

$$Q^{(n-m)}_{n-1:0} = P_{n-1:m}D_{m-1:0} \tag{23}$$

Again we divide the group of bits $n-1:0$ into three subgroups $n-1:k$, $k-1:k'$ and $k'-1:0$. We further choose m and m' to be "midpoints" of the second and third groups respectively. We first decompose the D term using equation (21):

$$Q^{(n-m)}_{n-1:0} = P_{n-1:m}D_{m-1:k'}[B_{m-1:k'} + D_{k'-1:0}] \tag{24}$$

Since $P_{n-1:m}$ is simply an AND of bit-level propagates, we can rewrite:

$$Q^{(n-m)}_{n-1:0} = [P_{n-1:k}][P_{k-1:m}D_{m-1:k'}][B_{m-1:k'} + D_{k'-1:0}] \tag{25}$$

The first two bracketed terms are hyper propagate functions, the last D function can be further simplified. We now make use of equation (22):

$$\begin{aligned} Q^{(n-m)}_{n-1:0} = {} & [P_{n-1:k}][P_{k-1:m}D_{m-1:k'}] \\ & [B_{m-1:k'} + G_{k'-1:m'} + P_{k'-1:m'}D_{m'-1:0}] \end{aligned} \tag{26}$$

Which has the required form:

$$Q^{(n-m)}_{n-1:0} = Q^{(n-k)}_{n-1:k}Q^{(k-m)}_{k-1:k'}[R^{(m-k')}_{m-1:m'} + Q^{(k'-m')}_{k'-1:0}] \tag{27}$$

For the sake of completeness we rewrite equation (20), the reduced generate function, in terms of reduced generate and hyper propagate functions:

$$R^{(n-m)}_{n-1:0} = R^{(n-k)}_{n-1:k} + R^{(k-m)}_{k-1:k'} + Q^{(m-k')}_{m-1:m'}R^{(k'-m')}_{k'-1:0} \tag{28}$$

The above is an example derivation for radix three. There are in fact a myriad of possible architectures. Table I illustrates some higher radix examples and compares them to their prefix counterparts. We note that in each case the reduced generate function is simpler than the equivalent generate function, the hyper propagate function is more complex than the equivalent propagate function but no more complex than the reduced generate. We further note that in general our radix $n+1$ functions are comparable in complexity to radix n generate function.

V. Examples

We now illustrate our method with two examples. The first is a radix three 27-bit carry.
We need to construct the function $G_{26:0}$. We decompose this as

$$G_{26:0} = D_{26:14}R^{(13)}_{26:0} \tag{29}$$

Using the radix three decomposition we have:

$$\begin{aligned} R^{(13)}_{26:0} &= R^{(4)}_{26:18} + R^{(4)}_{17:9} + Q^{(5)}_{13:5}R^{(4)}_{8:0} \\ D_{26:14} &= D_{26:23}[R^{(4)}_{26:18} + Q^{(5)}_{22:14}] \end{aligned} \tag{30}$$

where

$$\begin{aligned} R^{(4)}_{8:0} &= R^{(1)}_{8:6} + R^{(1)}_{5:3} + P_{4:2}R^{(1)}_{2:0} \\ R^{(4)}_{17:9} &= R^{(1)}_{17:15} + R^{(1)}_{14:12} + P_{13:11}R^{(1)}_{11:9} \\ R^{(4)}_{26:18} &= R^{(1)}_{26:24} + R^{(1)}_{23:21} + P_{22:20}R^{(1)}_{20:18} \\ Q^{(5)}_{13:5} &= P_{13:11}P_{10:8}[P_{7:5} + R^{(1)}_{8:6}] \\ Q^{(5)}_{22:14} &= P_{22:20}P_{19:17}[P_{16:14} + R^{(1)}_{17:15}] \\ D_{26:23} &= p_{26}[P_{25:23} + R^{(1)}_{26:24}] \end{aligned} \tag{31}$$

Each

$$\begin{aligned} R^{(1)}_{k+2:k} &= g_{k+2} + g_{k+1} + p_{k+1}g_k \\ P_{k+2:k} &= p_{k+2}p_{k+1}p_k \end{aligned} \tag{32}$$

If a sum needs to be computed then we proceed as Ling. We have

$$\begin{aligned} sum_{27} &= (x_{27} \oplus y_{27}) \oplus G_{26:0} \\ &= (x_{27} \oplus y_{27}) \oplus D_{26:14}R^{(13)}_{26:0} \\ &= \overline{R^{(13)}_{26:0}}(x_{27} \oplus y_{27}) + R^{(13)}_{26:0}(x_{27} \oplus y_{27} \oplus D_{26:14}) \end{aligned} \tag{33}$$

The next example is for a 32-bit carry. We use a combination of radix 2 and radix 4 equations. We need to construct the function $G_{31:0}$. We decompose this as

$$G_{31:0} = D_{31:21}R^{(11)}_{31:0} \tag{34}$$

Using the radix four decomposition we have:

$$\begin{aligned} R^{(11)}_{31:0} &= R^{(3)}_{31:24} + R^{(3)}_{23:16} + Q^{(5)}_{20:13}R^{(3)}_{15:8} + Q^{(5)}_{20:13}Q^{(5)}_{12:5}R^{(3)}_{7:0} \\ D_{31:21} &= D_{31:29}[R^{(3)}_{31:24} + Q^{(5)}_{28:21}] \end{aligned} \tag{35}$$

where

$$\begin{aligned} R^{(3)}_{7:0} &= R^{(1)}_{7:6} + R^{(1)}_{5:4} + P_{4:3}R^{(1)}_{3:2} + P_{4:3}P_{2:1}R^{(1)}_{1:0} \\ R^{(3)}_{15:8} &= R^{(1)}_{15:14} + R^{(1)}_{13:12} + P_{12:11}R^{(1)}_{11:10} + P_{12:11}P_{10:9}R^{(1)}_{9:8} \\ R^{(3)}_{23:16} &= R^{(1)}_{23:22} + R^{(1)}_{21:20} + P_{20:19}R^{(1)}_{19:18} + P_{20:19}P_{18:17}R^{(1)}_{17:16} \\ R^{(3)}_{31:24} &= R^{(1)}_{31:30} + R^{(1)}_{29:28} + P_{28:27}R^{(1)}_{27:26} + P_{28:27}P_{26:25}R^{(1)}_{25:24} \\ Q^{(5)}_{12:5} &= P_{12:11}P_{10:9}P_{8:7}[R^{(1)}_{7:6} + P_{6:5}] \\ Q^{(5)}_{20:13} &= P_{20:19}P_{18:17}P_{16:15}[R^{(1)}_{15:14} + P_{14:13}] \\ Q^{(5)}_{28:21} &= P_{28:27}P_{26:25}P_{24:23}[R^{(1)}_{23:22} + P_{22:21}] \\ D_{31:29} &= p_{31}[R^{(1)}_{31:30} + P_{30:29}] \end{aligned} \tag{36}$$

Each

$$\begin{aligned} R^{(1)}_{k+1:k} &= x_{k+1}y_{k+1} + x_ky_k \\ P_{k+1:k} &= (x_{k+1} + y_{k+1})(x_k + y_k) \end{aligned} \tag{37}$$

TABLE I

COMPARISON OF COMPLEXITY AT VARIOUS RADICES

Radix	Parallel Prefix	New
2	$G_1 + P_1G_0$ P_1P_0	$-$
3	$G_2 + P_2G_1 + P_2P_1G_0$ $P_2P_1P_0$	$R_2 + R_1 + Q_1R_0$ $Q_2Q_1[R_0 + Q_0]$
4	$G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0$ $P_3P_2P_1P_0$	$R_3 + R_2 + Q_2R_1 + Q_2Q_1R_0$ $Q_3Q_2Q_1[R_0 + Q_0]$
5	$G_4 + P_4G_3 + P_4P_3G_2 + P_4P_3P_2G_1 + P_4P_3P_2P_1G_0$ $P_4P_3P_2P_1P_0$	$R_4 + R_3 + R_2 + Q_2R_1 + Q_2Q_1R_0$ $Q_4Q_3Q_2Q_1[R_0 + Q_0]$

If a sum needs to be computed then we proceed as Ling. We have

$$\begin{aligned} sum_{32} &= (x_{32} \oplus y_{32}) \oplus G_{31:0} \\ &= (x_{32} \oplus y_{32}) \oplus D_{31:21} R^{(11)}_{31:0} \\ &= \overline{R^{(11)}_{31:0}}(x_{32} \oplus y_{32}) + R^{(11)}_{31:0}(x_{32} \oplus y_{32} \oplus D_{31:21}) \end{aligned} \tag{38}$$

VI. CONCLUSION

We have exposited a new theory of high speed binary addition. In particular we have shown that the Ling method can be generalized to all levels. The theory allows for a myriad of architectures suitable for different design styles. As an example we have shown that the parallel prefix method requires n-input gates for radix n addition, whereas our method allows for radix n+1 addition with n-input gates. We are currently investigating families of adders [4] due to our equations.

REFERENCES

[1] O.J. Bedrij, “Carry Select Adder”, IRE Trans., EC-11, pp.340-346, June 1962.

[2] R.C. Jackson and S. Talwar, “High Speed Adder”, US Patent Application, No. 60/436,179, Dec. 2002.

[3] R.C. Jackson and S. Talwar, “A Logic Circuit and Method for Carry and Sum Generation and Method of Designing Such a Logic Circuit”, US Patent Application, Nov. 2003.

[4] S. Knowles, “A Family of Adders”, Proc. 14th IEEE Symp. on Computer Arithmetic, pp.30-34, 1999.

[5] P.M. Kogge and H.S. Stone,“A Parallel Algorithm for Efficient Solution of a General Class of Recurrence Equations”, IEEE Trans. Computers, Vol. C-22, No. 8, pp.786-793, Aug. 1973.

[6] R.E. Ladner and M.J. Fischer, “Parallel Prefix Computatiom”, Journal of ACM, Vol. 27, No.4, pp.831-838, Oct. 1980.

[7] H. Ling, “High Speed Binary Adder”, IBM Journal of Research and Developement, Vol. 25, No. 3, pp.156-166, 1981.

[8] J. Sklansky, “Conditional-Sum Addition Logic”, ire TRANS., EC-9, pp. 226-231, June 1960.

[9] A. Weinberger and J.L Smith, “A Logic for High-Speed Addition”, Nat. Bur. Stand. Circ., 591, pp.3-12, 1958.