

Designing Optimum Carry-Skip Adders

Vitit Kantabutra

Department of Mathematics and Computer Science

Drexel University

Philadelphia, Pennsylvania 19104

1 Introduction

In designing binary adders, the computer architect has many criteria to consider. Circuit speed, power consumption, and cost are of prime importance. For VLSI implementation, chip area must be kept low. In recent years, *carry-skip adders* have often been the adders of choice, since the best carry-skip adders (even the simple *one- or two-level* ones) can be comparable in speed to the fastest adders such as carry-lookahead adders [4]; yet unlike the carry-lookahead adders, the carry-skip adders remain quite low in cost, chip area, and power consumption [10]¹. We must mention, though, that for extremely large operand sizes, Ofman's adders are faster than carry-skip adders².

The present paper describes a method for designing optimum-speed one-level carry-skip adders. This method always yields the fastest adders if the assumptions of Guyot, et al. [4] hold – that is, if the *ripple time* (a circuit parameter) of a carry signal is a linear function of the number of bit positions that the carry signal propagates through, and if the *skip time* (another circuit parameter) of a carry signal is a linear function of the number of blocks of bit positions skipped by the signal, or if these two parameter are such mildly nonlinear functions that they can be modeled by a linear function without any effect on any of the results obtained. The circuit design method to be described in this paper is useful because Guyot, et al. have shown that in device technologies such as 2-Alu CMOS the nonlinearities are often insignificant.

The skip time / ripple time ratio is called ρ . The method presented in this paper gives the fastest circuit, no matter what ρ is. The method given by

Guyot, et al. [4] is only guaranteed to yield circuits whose speeds are within twice the skip time of the optimum. The author has programmed his own algorithm, as well as the algorithm of Guyot, et al., and has performed a comparison study of the resulting circuits by these two algorithms, which is presented in Section 4 of this paper. There it is shown that Guyot, et al.'s circuits can have total carry propagation time that is more than 11% slower than those produced by the algorithm of the present paper. Our algorithm runs in polynomial time and takes about 8 seconds to run on a MacIntosh II for a sample input with an operand size of 128 bits and a ρ value of 2.0001. The circuits obtained are always optimum-speed. Additionally, our algorithm yields optimum-speed circuits for both integer and noninteger values of ρ , whereas Oklobdzija and Barnes' algorithm [9] only yields optimum circuits for integer values of ρ . If ρ is, in fact, not an integer, then their algorithm could still be run with the nearest integer input as an approximate value for ρ , but doing this would typically yield a circuit that is far from optimum. In Section 4 we give an example in which Oklobdzija and Barnes' algorithm yields a circuit that has a total carry propagation time over 16% slower than the carry propagation time of our optimum-speed circuit.

Two relevant papers appeared in the previous IEEE arithmetic circuit symposium. One of these papers is by Chan and Schlag. Chan and Schlag's work [3] allows the ripple time to be a wide range of functions, including any quadratic function of the number of bits rippled through by the carry signal. For certain technologies, this assumption would be an improved model for the ripple time, and could yield faster circuits. However, their algorithm appears to be slower than the one in this paper for practical operand sizes, although the exact analysis of the speeds of both algorithms seems difficult. Algorithm speed is important in the cases where the circuit designer wants to try different values of ρ to see which value gives the best circuit for the technology and the application at hand. It would be interesting to test the speed of all the carry-skip circuit design algorithms for the operand sizes for which carry-skip adders are fast. Asymptotic algorithm analysis doesn't seem very useful, since the $O(\log n)$ -delay adders, such as Ofman's adder [8], would beat the speed of carry-skip adders for extremely large operand sizes n . Guyot, et al.'s algo-

¹One-level carry-skip adders may be as fast as carry-lookahead adders, but are more likely to be 20%-30% slower for operand sizes of 64 bits or more. However, this is offset by the fact that these carry-skip adders are *many times* more efficient in terms of power consumption and design and implementation costs.

²If n is the operand size, then Ofman's adder exhibit a total delay of $O(\log n)$, whereas carry-skip adders exhibit a total delay of $O(\sqrt{n})$. Carry-lookahead adders are often also faster than carry-skip adders, though that isn't clear, since technological limitations currently dictate that carry-lookahead adders for operands of sizes larger than a few bits be in a modified (multi-level) form. And in this form, those adders are slowed down somewhat.

rithm seems to be the fastest algorithm, even though it doesn't always yield the fastest circuits. The other paper is by Silvio Turrini. Turrini's paper [10] suggests, without any convincing analysis, an algorithm that supposedly computes an optimum circuit for any combination of skip time and ripple time, with no restrictions on the number of skip levels. However, Turrini's work only covers a limited case of Guyot, et al.'s Model 1 (a description of Model 1 exists in the next subsection). The present paper, however, covers the full range of Model 1.

Carry-skip adders were invented for decimal arithmetic operations by Charles Babbage in the 1800's, and became quite popular in mechanical adding machines later that century. Modern interest in carry-skip adders only began in the early 1960's with the publication of Lehman and Burla's paper [6]. In the next section, we will present a technical introduction to carry-skip adders and then we will go on to survey the modern literature pertaining to carry-skip adders.

The author hopes that this paper will help carry-skip adders to gain the recognition they deserve for their high speed, energy efficiency, cost efficiency, and ease of implementation.

2 Technical Details

2.1 A Detailed Description of Carry-Skip Adders

Let us now develop a detailed description of carry-skip adders. Let the two binary operands be called $X = x_{n-1}x_{n-2}\dots x_2x_1x_0$ and $Y = y_{n-1}y_{n-2}\dots y_2y_1y_0$. Call the carry into and the carry out of bit position i respectively c_i and c_{i+1} . The result bit at position i is to be called z_i . The result can have at most $n + 1$ bits; z_0 up to z_n . The following relations for z_i and c_{i+1} are well-known:

$$z_i = x_i \oplus y_i \oplus c_i \quad (1)$$

$$c_{i+1} = x_i y_i + y_i c_i + c_i x_i \quad (2)$$

where \oplus and $+$ respectively denote the EXCLUSIVE OR and the (INCLUSIVE) OR operations, and when two variables are written next to each other they are to be ANDed together. The carry input c_0 to the circuit is 0 when the adder is used for adding two numbers in the usual way, but this input can be connected to the c_n output of another adder to create a multiple-precision adder.

Carry-skip adders are best understood by first looking at the carry-ripple adder. The carry-ripple adder is obtained by a straightforward implementation of an adder circuit according to the logical relations (1) and (2) for z_i and c_{i+1} given above. In this adder a carry signal may have to propagate all the way from the least significant position to the most significant position. An example would be the case where $\forall i$ except 0, $x_i = 0, x_0 = 1$, and $\forall i, y_i = 1$. Note that in every bit position i through which the carry signal travels, x_i and y_i are different. This is a significant fact, because in general, a carry signal will propagate through a bit

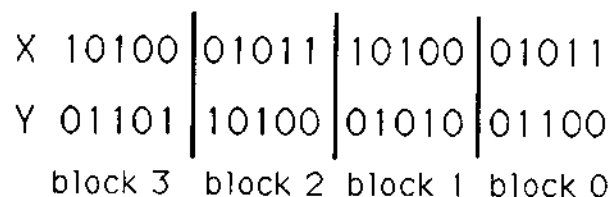


Figure 1: Dividing the bit positions into blocks

position i if and only if $x_i \neq y_i$. (To make this clearer, note that if $x_i = y_i$, then all carry signals to the left of position i are independent of all inputs in positions 0 through $i - 1$.) Thus, the chances that a carry signal travels a long way is small, because the probability of having a long string of bit positions, all with $x_i \neq y_i$, is small. In fact, early electronic computer researchers [1] have shown that the average time delay of a carry-ripple adder is only $O(\log n)$. (See [1].)

In a carry-skip adder, we take advantage of the observations just discussed. The bit positions are to be divided into *blocks* of contiguous positions³, as shown in Figure 1. Let us number these blocks 0, 1, 2, ... from right to left. When a block has only positions i with $x_i \neq y_i$, like block 2 in Figure 1, then we know that the carry out of that block is the same as the carry into the block⁴. We also know that the carry input to the bit positions in the block are either all 0's or all 1's, depending on whether the carry input to the least significant bit of the block is a 0 or a 1, respectively. Thus, no carry signal has to propagate through this block in a position-by-position fashion. Finding out if all positions i in the block have $x_i \neq y_i$ is an easy $O(\log n)$ process, and will even be regarded instead as a constant-time process for practical block sizes. (For example, even when a block is as large as 16 bits, which is unusual, we can use a layer of EXCLUSIVE OR gates and 2 layers of 4-input AND gates.) Similarly, it is a quick process to force the carry input bits in the block to a uniform value. Aside from these two types of time delays, there is also the delay caused by the final addition step, which is constant, and the delay due to carry propagation. Thus, in practical circuits this very last type of delay is the only type that causes nonconstant total circuit delay, and is the quantity to be minimized in order to obtain an optimum circuit.

There are two kinds of carry propagation delays. First there is the *ripple time*, r , which is the time it takes for a carry signal to pass through a single adder cell

³We consider $n + 1$, not n , bit positions because the result of the addition operation can have up to $n + 1$ bits.

⁴On the contrary, if for some position i in the block, $x_i = y_i$, then we may correctly think of the carry signal into the block (no matter it is 0 or 1) as stopping at the *rightmost* such bit position inside the block. This is because the carry signals to the left of that point is independent of the carry into the block. The carry out of the block is completely determined by the value of x_i at the *leftmost* bit position at which $x_i = y_i$. In fact, the carry out of the block is equal to x_i .

(a single bit position). Then there is the *skip time*, s , which is the time a carry signal needs to skip a block, i.e.; the time interval between the moment it is known to the circuit that the carry into a particular block is of a certain value and the moment when the carry out of that block gets set to the same value. The ripple time per bit is certainly constant, and usually the skip time for each block doesn't depend on the block size enough to be regarded as anything other than a constant⁵.

Thus, minimizing the total circuit delay involves partitioning the bit positions into blocks in the best possible way. Lehman and Burla [6] found the best partitioning scheme, if one is to try only blocks of equal size. The authors suggest, though, that unequal block sizes may yield a faster circuit. This suggestion is, indeed, correct. Majerski [7] studied how to minimize carry propagation time and the number of skips when unequal blocks are allowed. In both these papers, the ratio s/r (henceforth called ρ) was assumed to be 1, which is often unrealistic. (This ratio ρ depends on the particular circuit technology and the particular implementation involved.) Oklobdzija and Barnes [9] gave a method for computing optimum block sizes (block sizes that yield the fastest circuit), but only for $\rho = 2, 3, 4, 5, 6$, or 7. Yet in practice ρ is usually a non-integer, and sometimes less than 2. Recently Guyot, Hochet, and Muller [4] showed how to compute near-optimum block sizes for all rational values of ρ . The block sizes computed by their method are guaranteed to be such that the total circuit delay falls within $2s$ of the best possible. In the present paper we compute an *optimum* (guaranteed fastest possible, not just nearly so) circuit for any given ρ . Also, ρ can be any real number.

2.2 Deriving the Optimum-Circuit Algorithm

We will study a mathematical model for carry-skip adders and compute the delay based on that model. Two models for carry-skip adders are considered by Guyot, et al. [4] - Model 1 was developed based on the assumption that the skip time doesn't depend on the length of the block being skipped, whereas Model 2 was based on the opposite assumption. However, the dependence of the skip time on the block size was often found to be so small that Model 1 yields results that are no different from those obtained by using the more complicated Model 2. So in the present work we will use only Model 1, which can be described as follows:

A Description of Model 1:

We have mentioned before that the only nonconstant delay is that which is caused by carry propagation. Suppose that in position i , $x_i = y_i$. Then there is a carry out of either 0 or 1 from position i . The value of this carry depends solely on the values of x_i and y_i , and not on the input bits at any other positions. Because of this independence on other positions, the carry is said to be *generated* at position i . Now let

j be the first position to the left (left = more significant) of i such that $x_j \neq y_j$. Then the carry is said to *propagate* through positions $i + 1, i + 2, \dots, j - 1$, and finally to be *terminated* at position j . We must model and minimize the total carry propagation delay, which is equal to the longest delay (taken over all carry signals) caused by the propagation of a carry signal from the moment after it is generated up until when it is terminated. The delay caused by a carry signal from generation to termination is called the *life* of the signal. The life of a carry signal is divided into 3 parts:

1. Right after generation, the carry has to travel left to the nearest boundary of a block, unless the termination position j is reached first, and,
2. then the carry has to skip over as many blocks as it takes to reach the closest block boundary lying to the right of position j , and finally,
3. the carry travels to its termination at position j .

The delay involved in parts 1. and 3. combined is equal to the number of positions traveled multiplied by the ripple delay, r , whereas the delay caused by part 2. is given by the number of blocks skipped multiplied by the skip time, s .

To achieve the objective of minimizing the longest possible carry signal life, we have to partition the $n + 1$ bit positions into blocks in the best way. (Recall that we consider the total number of bit positions to be $n + 1$.) To see what "best" means, let us introduce a method of representation of how the bit positions are partitioned. This representation is equivalent to the one invented by Guyot, et al. [4].

Imagine an X-Y plane such that the Y axis points upwards as usual but the X axis points to the left instead of to the right. Again, let the blocks be numbered $0, 1, 2, \dots$ from right to left as in Figure 1. Now, if block b contains $m(b)$ bit positions, then put a special marker at the point $(b, m(b))$ on the X-Y plane. See Figure 2. Let M be the set of all such markers, and let μ be the set of all points b for which there is a marker with X coordinate b . Note that $\sum_{\forall b \in \mu} m(b) = n + 1$. Note also that the time needed for a carry signal to ripple from the first to the last bit in a block of size $m(b)$ is $r(m(b) - 1)$. Guyot, et al., however, used $rm(b)$. The difference doesn't affect the results in any way.

Theorem 1 *The longest possible carry life in a circuit is given by the expression*

$$\max_{\substack{\forall b_2, b_1 \in \mu \\ b_2 > b_1}} \{r(m(b_2) + m(b_1) - 2) + s(b_2 - b_1 - 1)\}.$$

Proof. This theorem follows immediately from the previous discussion on the life of a carry. ■

Some important isosceles triangles will now be introduced, because we'll find the best partitioning of the

⁵See [4] for a study of the effect of block size on skip time.

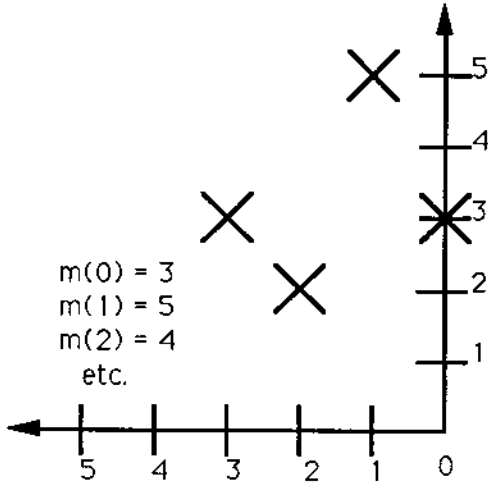


Figure 2: A representation of blocks

bit positions into blocks by finding the lowest height isosceles triangle that contains all the markers in the set M .⁶ Let $\rho = s/r$, and let Δ_ρ be the class of all isosceles triangles such that the two sloped sides have slopes ρ and $-\rho$, and such that the horizontal side (the base) lies on the X axis. Let $\delta_{\rho_{\min}}$ be the minimum-height member of Δ_ρ that contains all the markers of M . It is clear that $\delta_{\rho_{\min}}$ exists. That $\delta_{\rho_{\min}}$ is unique will be shown in Lemma 1, below. Let $H(\delta)$ denote the height of any isosceles triangle δ .

Lemma 1 *The triangle $\delta_{\rho_{\min}}$ is unique.*

Proof. Assume the contrary - that there are at least two distinct triangles of minimum height that contain all the markers of M . Let two of them be called δ_1 and δ_2 . Let $\delta_3 = \delta_1 \cap \delta_2$. Note that δ_3 is a triangle of smaller height than δ_1 and δ_2 , has sides of slope ρ and $-\rho$, and contains all the markers of M . This observation contradicts the assumption made at the beginning of the proof. ■

The following theorem is important because, as we will later see, it implies that in most cases, $\delta_{\rho_{\min}}$ completely determines an optimum-speed circuit.

Theorem 2 *If the sloped sides of $\delta_{\rho_{\min}}$ each touch at least one marker, and if at least one of these sides touches a marker somewhere other than at the apex of the triangle, then the worst carry signal life is given by $2rH(\delta_{\rho_{\min}}) - 2r - s$.*

Proof. Assume that the hypothesis of the theorem is true. Then there must be natural numbers i and

⁶This statement isn't quite accurate, but serves as a good intuitive explanation of the reason of existence of these isosceles triangles. As the paper progresses, the reader will see more accurately how these triangles are used for obtaining the best partitioning of the bit positions.

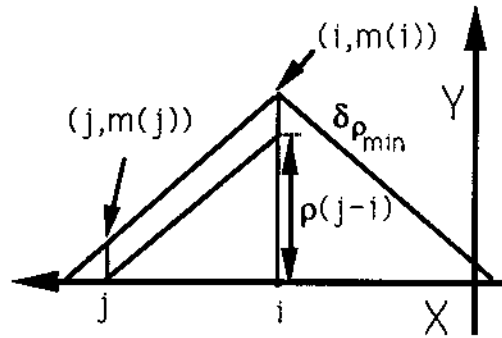


Figure 3: Two markers, one on each side of $\delta_{\rho_{\min}}$

j such that the point $(i, m(i))$ lies on the right side (right sloped boundary) of $\delta_{\rho_{\min}}$ and such that the point $(j, m(j))$ lies on the left side of the same triangle. In addition, at least one of those two points lies off the apex. Without loss of generality assume that $(j, m(j))$ does, as shown in Figure 3. Let us look at the case in which $(i, m(i))$ is at the apex. Consider the situation where a carry signal has to travel from the rightmost bit of block i to the leftmost bit of block j . This situation obviously can occur. By Theorem 1, the life of this carry signal is equal to $r(m(j) + m(i) - 2) + s(j - i - 1)$, which is the same as $r(m(j) + m(i) + \rho(j - i)) - 2r - s$. From Figure 5 we see that $m(j) + \rho(j - i) = H(\delta_{\rho_{\min}})$. It follows immediately that the carry signal life is $2rH(\delta_{\rho_{\min}}) - 2r - s$. The case in which both $(i, m(i))$ and $(j, m(j))$ are off $\delta_{\rho_{\min}}$'s apex can be dealt with similarly. Note finally that no carry signal can have a life longer than does a carry signal discussed above. ■

If $\delta_{\rho_{\min}}$ touches no markers other than one (which would be at the apex), then some definitions are needed, as follows: (Refer to Figure 4.) Let l_1 and l_2 , respectively, be lines parallel to the left and the right sides of $\delta_{\rho_{\min}}$ so that l_1 and l_2 both touch some markers, and the shaded triangle sided by l_1 , l_2 , and part of $\delta_{\rho_{\min}}$'s base contains all markers excepts the one at the apex of $\delta_{\rho_{\min}}$. Assume that l_1 is farther from the left side of $\delta_{\rho_{\min}}$ than l_2 is from the right side of the same triangle. (The other cases can be treated similarly.) Draw a vertical line V through the apex of $\delta_{\rho_{\min}}$, and let A, B, C, D , and G be points of intersections among previously-defined lines, as shown in Figure 4. E and F will be explained in the proof of Theorem 3, coming up next. Let $x(P)$ and $y(P)$, respectively, denote the X and the Y coordinates of a point P .

Theorem 3 *If $\delta_{\rho_{\min}}$ touches a marker at the apex (D) but touches no marker at any other place, then the worst possible total carry propagation delay of the circuit is given by the expression $r(y(C) + y(D)) - 2r - s$.*

Proof. By the definition of l_1 and l_2 , there must be some marker on l_2 . Let E be the location of this

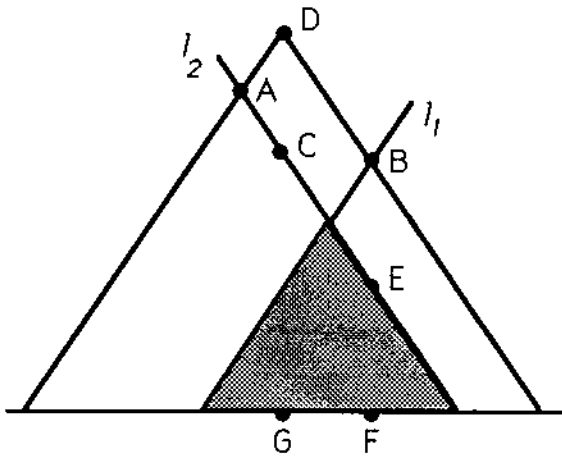


Figure 4: The case where $\delta_{\rho_{\min}}$ touches the markers only at the apex

marker, and let F be the point on the X axis vertically aligned with E . It is easy to see that the longest carry life in the entire circuit is the life of a carry signal that is generated at the rightmost bit position in the block numbered $x(E)$ and is terminated at the leftmost bit position in the block numbered $x(D)$. Thus, the longest carry signal life is equal to $r(y(C) + y(E) - 2) + s(\|FG\| - 1)$. Using similar geometric reasoning as in the proof of Theorem 2, we can see that this expression is equal to the expression $r(y(C) + y(D)) - 2r - s$. ■

Now define Δ_ρ^h to be the set of all members of height h of Δ_ρ .

Lemma 2 For every circuit with delay no more than $2rh - 2r - s$, there exists a member δ of Δ_ρ^h such that at most one marker of the circuit lies strictly outside δ . In case that for every member of Δ_ρ^h , there is at least one marker outside the member, there exists a $\delta \in \Delta_\rho^h$ such that there is exactly one marker outside δ , and such that the marker outside δ is vertically aligned with the apex.

Proof. Pick any circuit with no more than $2rh - 2r - s$ units of delay. Draw an arbitrary member of Δ_ρ^h - call it δ . If the circuit doesn't have more than one marker outside δ , and the marker outside δ is vertically aligned with the apex, then the lemma is already proven. Otherwise there are two cases to consider:

(a) There are two markers at positions i and j such that these markers are vertically above the right and the left sides of δ , respectively. (There might be more than 2 markers outside δ .) One of these markers may be vertically aligned with δ 's apex.

(b) All markers are above the same side of δ and no marker is vertically aligned with δ 's apex.

In case (a) we can see, using a similar geometric argument as in the proofs of Theorems 2 and 3, that the longest possible carry life is longer than $2rh - 2r - s$. So case (a) can't occur.

In case (b), assume without loss of generality that all markers are above the left side. Slide δ to the left. Eventually one of the following events must occur:

1. All markers are in δ .
2. Some marker, say $(i, m(i))$, which has always remained outside δ , is now aligned with the apex of δ .

In case event (1) occurs, the lemma is clearly proven. In case event (2) occurs, it is easy to see that at the time of such occurrence, no other marker than $(i, m(i))$ can be outside of δ , or else the circuit's maximum delay would be greater than $2rh - 2r - s$. Hence the lemma is also proven in this case. ■

Theorem 4 Let h be a positive real number such that the members of Δ_ρ^h has base length at least 2. Then, among all circuits with largest n (with $s/r = \rho$) of delay up to $2rh - 2r - s$, there is at least one with one of the following properties:

1. All the markers are contained in⁷ some member δ of Δ_ρ^h , or,
2. For some member δ of Δ_ρ^h , only 1 marker of the circuit is strictly outside δ . This marker is vertically aligned with the apex of δ , and is less than 1 unit above it.

Proof. Pick any largest circuit c (largest = largest number n of operand bits) with delay up to $2rh - 2r - s$, and suppose that there is no member of Δ_ρ^h that will contain all of c 's markers. Then by Lemma 2, there must be a $\delta \in \Delta_\rho^h$ that contains all but one marker, and this marker is vertically aligned with the apex of δ . We will modify c to obtain a new circuit, c' , such that c' has the same size and delay upper bound as mentioned above for c , but either has all the markers in δ or has only one marker outside δ , such that this marker is vertically aligned with the apex and is less than 1 unit above it. If we can do this then we will have proven the theorem. Understanding how to modify c involves looking at Figure 5. Figure 5 is just Figure 4 with δ and its apex H added in, and with unneeded labels G, F , and E removed. By Theorem 3, the given circuit has delay $r(y(C) + y(D)) - 2r - s$. Since the delay is at most $2rh - 2r - s$, we have $h \geq \frac{y(C) + y(D)}{2}$. The strategy now is to shorten the block numbered by $x(H)$ (the X coordinate of the point H)

⁷Not strictly.

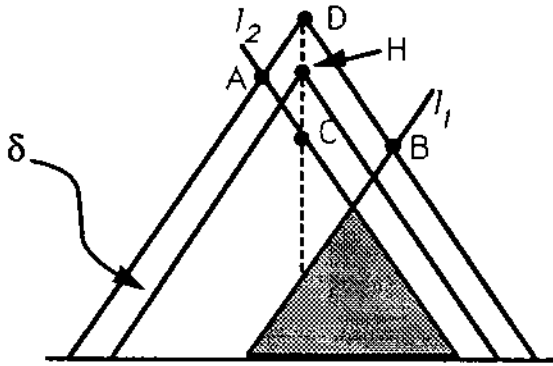


Figure 5: Related to obtaining a new circuit with all the markers or all but one in δ .

just enough so that $\lfloor m(x(H)) \rfloor \leq h$, and then to put the bit positions removed from that block elsewhere. Since $h \geq \frac{y(C)+y(D)}{2}$, it follows that $y(H) - y(C) \geq y(D) - y(H)$. This last relation is very important, because it means that roughly, we can remove $y(D) - y(H)$ bit positions from block $x(H)$ and insert them in any other block. Exactly speaking, though, this is not always the case, because $y(D) - y(H)$ and $y(H) - y(C)$ are not necessarily integers.

There are 2 cases to consider. The first case is when $y(D) - y(H) < 1$. Then there is nothing to do, since the conclusion of the theorem is already true. So let us consider the second case, in which $y(D) - y(H) \geq 1$. In this case we can move $\lfloor y(D) - y(H) \rfloor$ bit positions from block $x(H)$ to some other block. From the geometry of Figure 5, it is clear that if this other block exists, then this move can be done. We have thus assumed the existence of 2 blocks in all. This is fine, since we have assumed that the base of δ is of length at least 2, and so the base covers at least 2 integer points on the X axis. ■

Our main goal is that when n is given, we would like to design a minimum-delay n -bit circuit. But it follows from Theorem 4 that for any delay d , some maximum- n circuits of delay up to d have all or all but one of their markers in some $\delta \in \Delta_\rho^h$ (provided that the base of δ is larger than 2 units). This will now be stated formally as follows:

Theorem 5 *Let a real number d be given. If $d \geq s - 2r$, then among all the maximum- n circuits with the skip time/ripple time ratio equal to the given ρ , there is one such that either*

1. *all the circuit's markers fit into some member of Δ_ρ^h , where $h = (d + 2r + s)/2r$, or,*
2. *there exists a member δ of Δ_ρ^h (again, where $h = (d + 2r + s)/2r$) such that only 1 marker is outside δ . Moreover, this marker is vertically aligned with the apex of δ , and is less than 1 unit above it.*

Such a circuit will be of delay at most d .

Proof. If $d \geq s - 2r \geq 2\rho r - 2r - s$ then $h = (d + 2r + s)/2r \geq 2\rho r/2r = \rho$. The base length b of each member of Δ_ρ^h is given by the expression $\frac{h}{\rho} = \rho$, and so $b = 2h/\rho \geq 2$. Thus the corollary follows from Theorem 4. ■

In the next section, we will use Theorem 5 and the other theorems to design an algorithm that outputs optimum-speed carry-skip adders.

3 The Procedure

We will now describe the algorithm such that, when given the number n of operand bit positions, will compute a partitioning of the positions into blocks⁸ so as to yield the minimum worst-case circuit delay. If the number of blocks in all optimum circuits is less than 2 (the trivial and unrealistic case), then the optimum circuit is simply the only existing circuit. Otherwise, an optimum circuit can be found by using the results of the previous section, which culminated in Theorem 5. The trick is simply to find the minimum delay d_{\min} such that there exists a circuit of at least n bit positions with delay d_{\min} . The technique used will be a "binary search" on the amount of delay. That is, we will first start with a range of delays (real numbers) $[d_{\text{low}}, d_{\text{high}}]$ such that we know that d_{\min} is within this range. Then compute what the size (size = maximum number of result bits) of the largest circuit is that will possess a worst-case delay of up to d_{mid} , where $d_{\text{mid}} = \frac{d_{\text{low}} + d_{\text{high}}}{2}$. Then halve the range of delays at d_{mid} in such a way that the new range will still contain d_{\min} . It remains only to show how to determine the size of the largest circuit when a delay figure is given. This will now be explained.

Let us suppose that we have drawn the picture of some member δ of Δ_ρ^h for some arbitrarily chosen h . Then the size of the largest circuit whose markers are all contained in δ is equal to the number of points in the set $\delta \cap S$ where $S = \{(x, y) | x \in \mathbf{Z} \text{ and } y \in \mathbf{Z}_+\}$. Moreover, we can read off the design of one such circuit from the picture by inspection! The circuit we mean is the one whose markers correspond to the maximal points⁹ of S in δ .

So if there is a largest circuit for the given delay figure such that this circuit's markers are all contained in some triangle in Δ_ρ^h for the appropriate h , then to compute this circuit we just have to find which triangle contains the most points of S . This can be done by starting with some arbitrary triangle $\delta \in \Delta_\rho^h$, and then sliding the base of δ along the X axis. During the slide, points in S enter and leave δ . Is this slide a finite process? To answer this question, notice two things:

⁸As noted earlier, we actually partition $n + 1$, not n , bit positions into blocks, because we must accommodate the most significant bit of the result.

⁹Points with maximum Y coordinate.

First, $|\delta \cap S|$ is a periodic function of the position¹⁰ of δ , with period 1; and second, the events that points of S enter and leave δ are discrete events. The first observation means that we only need to slide δ by 1 unit, in either direction we choose. Because of this and the second observation, the slide is indeed a finite process.

Now there is a possibility that the largest circuits are such that the markers are not all contained in the relevant member of Δ_ρ^h . Then it is not hard to see that in this case, there is only one largest circuit. This circuit can be computed by drawing a member δ of Δ_ρ whose apex is vertically aligned with an integer point on the X axis. Suppose the apex lies at the point which we will call (i, y) . By Theorem 4, the apex is less than 1 unit below the marker of block i . Since the marker must be at an integer point, y can't be an integer. Let $j = \lceil y \rceil$. Then the circuit is simply the one whose markers are the maximal points of S in δ , except that the marker for block i is taken to be at (i, j) . Compute a circuit by this latter procedure anyway whether its delay is within the amount specified or not, and whether it is a largest circuit or not. If the delay is too great, or if the circuit is not a largest one, then just discard it.

The sliding part of the process is detailed in Table 1.

4 Test Results

4.1 Comparison of This Paper's Circuits with Guyot, et al.'s Circuits

As mentioned before, the author has programmed both his algorithm and Guyot, et al.'s, and has run them on a variety of inputs. Table 2 compares the circuits resulting from some test runs. It is not really understood when Guyot, et al.'s algorithm performs well, and when it does relatively poorly, except that it seems to perform poorly when ρ is only a small amount away from a round number, like 1.01 or 1.0001. There are times when the circuits produced by their algorithm are as good as our results, and there are times when their circuits possess carry propagation delays that are more than 11% worse than ours, which are guaranteed to be optimum-speed circuits. All circuit delays are all calculated according to the formulas presented in this paper.

4.2 Comparison with Oklobdzija and Barnes' Result

As mentioned in the introduction, Oklobdzija and Barnes' algorithm [9] can be used for finding optimum-speed carry-skip adders, but only for integer values of ρ . If ρ isn't an integer, then their algorithm can still find a circuit, but such a circuit can be far from optimum. In the following example, their algorithm produces a circuit which has a carry propagation delay that is 16.08% slower than the carry propagation delay of the optimum circuit, produced by the algorithm in our paper:

```

begin
place  $\delta$  so that its left vertex is at 0;
pool := number of markers in  $\delta$ ;
left1 := number of markers on left side of  $\delta$ ;
left2 := 0; right := 0; oldmax := 0;
deltaposition := 0; {Initial position of  $\delta$ }
flag := true;
while flag do begin
{evaluate number of markers in triangle}
pool := pool - left2;
pool := pool + right;
if pool > oldmax then
begin
oldmax := pool;
deltaposition := X coordinate of
left vertex of  $\delta$ ;
end;
{slide triangle}
slide  $\delta$  to the right until a marker is hit;
if the left corner of  $\delta$  is at or
beyond  $x = 1$  then flag := false;
left2 := left1;
left1 := no. of markers on left side of  $\delta$ ;
right := no. of markers on right side of  $\delta$ ;
end; {while}
output the circuit determined by deltaposition;
end;

```

Table 1: Triangle sliding program

Suppose $\rho = 1.334$ and the number of bits is 32. Then our algorithm produces the circuit 1 2 3 5 6 6 4 3 2, with a carry propagation delay of 10.3380r units. They, on the other hand, must approximate ρ by either 1 or 2. If they were to use $\rho = 1$, then the circuit they would obtain is 1 2 3 4 5 6 5 4 3 2 1, with a carry delay of 12.006r units. On the other hand if they were to use $\rho = 2$, then the circuit they would obtain is 2 4 6 8 6 4 2, with a carry delay of 12.000r units. So they should use $\rho = 2$, which gives a deterioration of 16.08% over our optimum circuit's speed. (We calculate all carry delays according to our theorems. When calculating the delays of Oklobdzija and Barnes' circuits, we must remember that ρ is 1.334, not 1 or 2.)

5 Conclusion

In this paper, we have presented a method of designing the fastest one-level carry-skip adders. These adders are almost the fastest known adders, and are of clean design. It is possible, however, to apply the carry-skip principle once or several more times to obtain even faster, but somewhat more complicated, circuits. In [4] a procedure for finding near-optimum two-level carry-skip circuits was presented. An important next step in this research area would be to find a way to compute optimum two-level carry-skip adders. It shouldn't be too difficult to see that the method presented in this paper can be extended for that purpose.

¹⁰The X coordinate of the left end of the base.

bits	ρ	This paper's circuit's carry delay	Guyot, et al.'s circuit's carry delay	Guyot, et al.'s % deviation from optimum	This paper's circuit	Guyot, et al.'s circuit
32	1.0001	9.0005r	10.0000r	11.105%	1 2 3 4 5 6 5 3 2 1	1 2 3 4 6 6 4 3 2 1
32	1.025	9.1250r	10.0000r	9.589%	1 2 3 4 5 6 5 3 2 1	1 2 3 4 6 6 4 3 2 1
32	1.05	9.2500r	10.0000r	8.108%	1 2 3 4 5 6 5 3 2 1	1 2 3 4 6 6 4 3 2 1
32	5.5	19.0000r	20.0000r	5.2632%	2 8 13 7 2	5 10 11 6
64	2.0001	19.0007r	20.0001r	5.2598%	2 4 6 8 10 11 9 7 5 2	1 3 5 7 9 12 10 8 5 3 1
128	2.0001	28.0013r	29.0006r	3.5688%	2 4 6 8 10 12 14 16 14 12 10 8 6 4 2	2 4 6 9 11 13 15 16 14 12 10 7 5 3 1
128	.85	18.8500r	19.0000r	0.7958%	1 2 3 4 4 5 6 7 8 9 10 10 10 9 8 7 6 5 4 4 3 2 1	1 2 2 3 4 5 6 7 7 8 9 10 10 9 8 7 7 6 5 4 3 2 2 1
64	.85	12.8r	12.8r	0.0000%	1 2 3 4 5 5 6 7 7 6 5 4 3 3 2 1	1 2 3 4 5 5 6 7 7 6 5 4 3 3 2 1

Table 2: Comparing the circuits produced by the algorithms of Kantabutra and of Guyot, et al.

However, it seems that such an extension would be very complicated.

6 Acknowledgments

The author would like to thank Deborah L. Arnold for technical assistance in this project.

References

- [1] A. Burks, H. H. Goldstine, and J. von Neumann, "Preliminary discussion of the logic design of an electronic computing instrument," Institut. Adv. Study, Princeton, NJ, 1946 (reprinted in C. G. Bell, *Computer Structures: Readings and Examples*, New York: McGraw-Hill, 1971).
- [2] J. J. F. Cavanagh, *Digital Computer Arithmetic, Design and Implementation*. New York: McGraw-Hill, 1984.
- [3] Pak K. Chan and Martine D.F. Schlag, "Analysis and design of CMOS manchester adders with variable carry-skip," in *Proc. 9th IEEE Symp. on Comput. Arithmetic*, 1989.
- [4] A. Guyot, B. Hochet, and J.-M. Muller, "A way to build efficient carry-skip adders," *IEEE Transactions on Computers*, October 1987.
- [5] J. P. Hayes, *Computer Architecture and Organization, 2nd ed.* New York: McGraw-Hill, 1988.
- [6] M. Lehman and N. Burla, "Skip techniques for high-speed carry propagation in binary arithmetic units," *IRE Trans. Electron. Comput.*, p. 691, Dec 1961.
- [7] S. Majerski, "On determination of optimal distributions of carry skips in adders," *IEEE Trans. Electron. Comput.*, vol. EC-16, Feb. 1967.
- [8] Yu. Ofman, "On the algorithmic complexity of discrete functions," *Soviet Physics - Doklady*, vol. 7, no. 7, January 1963.
- [9] V. G. Oklobdzija and E. R. Barnes, "Some optimal schemes for ALU implementation in VLSI technology," in *Proc. 7th Symp. Comput. Arithmetic*, 1985.
- [10] Silvio Turrini, "Optimal group distribution in carry-skip adders," in *Proc. 9th IEEE Symp. on Comput. Arithmetic*, 1989.