Design Strategies for Optimal Multiplier Circuits

Charles Martel*
Dept. of Computer Science
University of California at Davis
Davis, CA 95616
Email: martel@cs.ucdavis.edu

Vojin Oklobdzija
Dept. of Electrical and Computer Engineering
University of California at Davis
Davis, CA 95616
Email: vojin@ece.ucdavis.edu

R. Ravi†
Dept. of Computer Science
Princeton University
Princeton, NJ 08544
Email: ravi@cs.princeton.edu

Paul F. Stelling‡
Dept. of Computer Science
University of California at Davis
Davis, CA 95616
Email: stelling@cs.ucdavis.edu

Abstract

We present new design and analysis techniques for the synthesis of fast parallel multiplier circuits. In [4], Oklobdzija, Villeger, and Lui suggested a new approach, the Three Dimensional Method (TDM), for Partial Product Reduction Tree (PPRT) design that produces multipliers which outperform the current best designs. The goal of TDM is to produce a minimum delay PPRT using full adders. This is done by carefully modelling the relationship of the output delays to the input delays in an adder, and then interconnecting the adders in a globally optimal way. Oklobdzija, et. al. suggested a good heuristic for finding the optimal PPRT, but no proofs about the performance of this heuristic were given.

We provide a formal characterization of optimal PPRT circuits and prove a number of properties about them. For the problem of summing a set of input bits within the minimum delay, we present an algorithm that produces a minimum delay circuit in time linear in the size of the inputs. Our techniques allow us to prove tight lower bounds on multiplier circuit delays. These results are combined to create a program which finds optimal TDM multiplier designs.

Keywords: *Multiplier design, Partial product reduction, Algorithms, Circuit design.*

1 Introduction

The design of efficient logic circuits is a fundamental problem in the design of high performance processors. The design of fast parallel multipliers is important since multiplication is a commonly used and expensive operation. Designing fast parallel multipliers is particularly critical for specialized chips which support multiplication intensive operations such as digital signal processing and graphics. Thus there have been many research projects and papers on the design of fast parallel multipliers; these results are surveyed in [1, 3, 13]. Continuing research in the area has led to a steady improvement in the designs for Partial Product Reduction Trees (PPRTs) for parallel multiplier designs as evidenced in the progression of work in [14, 2, 12, 10, 11, 6]. However, almost all of this prior work focused on finding good basic building blocks (compressors) using adders which could then be connected in a regular pattern to build a multiplier. In contrast, our approach is to design a faster multiplier by finding a globally optimal way of interconnecting low-level components.

We now discuss the design problem in more detail (a complete description is given in Section 2). After computing the partial products, the multiplication problem for two n-bit numbers can be reduced to two problems. First use a PPRT to add $(2n-1)$ columns of bits, producing two bits for each column (carrys are incorporated from each column to the next). Then add the two $(2n-1)$-bit numbers produced using a final adder, which we will call a carry-propagate adder (See Figure 1). The basic problems we address here relate to designing fast PPRTs.

In [4] the Three Dimensional Method (TDM) for globally designing the PPRT of a parallel multiplier circuit is described. The goal of the TDM is to produce a minimum delay PPRT using full adders ((3,2) adders) and a small number of half adders ((2,2) adders). In [4] it was shown that the TDM has the advantages of minimizing the number of devices required for the PPRT and allowing standard ASIC formats and utilities for generating layouts for tree-based circuits to be used. The speed improvements

*Research supported by NSF Grant CCR-94-03651.

†Most of this work was done when this author was at the Dept. of Computer Science, University of California at Davis and supported by NSF Grant CCR-91-03937. The author also acknowledges support from a DIMACS postdoctoral fellowship.

‡Research supported by NSF grants CCR-94-03651 and CCR-91-03937.

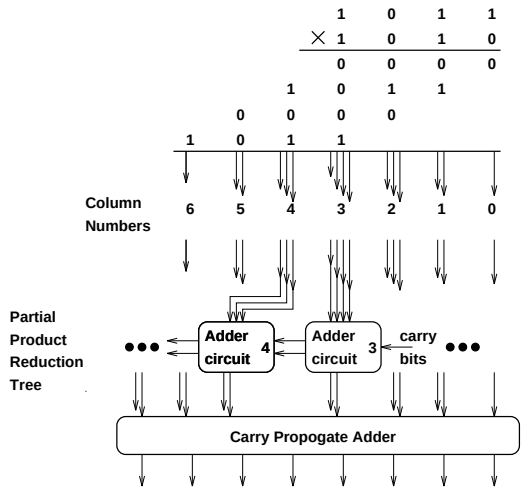

Figure 1: **The basic design of the multiplier, modelled after the long multiplication method for multiplying two binary numbers. Column i has weight 2^i. Each column of bits is added in a circuit that generates two sum bits and a number of carry bits that feed into the adder circuit for the next column. The output bits from the columns are summed in a fast adder to produce the final result of the multiplication.**

are achieved by carefully modelling the relationships of the output delays to the input delays in an adder, and then interconnecting the adders in a globally optimal way. They describe a linear-time heuristic for constructing PPRTs which outperform the best current designs. However, the question of whether the algorithm always derives the optimal TDM-based PPRT was left open. In this paper we describe a number of new techniques for designing and analyzing PPRTs. Specifically, we describe:

(i) An optimal class of circuits which can be used in constructing PPRTs.

(ii) A linear time algorithm for producing a delay-optimal circuit which sums a set of input bits.

(iii) A program which finds optimal PPRT circuits.

Using the program we have found optimal PPRT circuits for multiplying two numbers of size up to 40 bits. With improvements we will be able to solve larger problems. Our results show that the heuristic in [4] gives PPRT circuits with optimal or near-optimal overall delay. Our program finds PPRT circuits with better delay profiles. The techniques we developed allow the program to sharply prune the search space of PPRT circuits, and thus allow optimal circuits of larger sizes to be designed.

The basic component we use in our PPRT design is a *Full Adder* which takes three input bits available at times $a \leq b \leq d$ and produces a sum at time $s = \max(b + x_2, d + x_3)$ $(x_2 \geq x_3 \geq 0)$ and a carry at time $c = d + y_3$ $(y_3 \geq 0)$. We denote such a full adder by $g = (a, b, d)$, and use subscripting to refer to a specific adder (e.g., $g_i = (a_i, b_i, d_i)$ with sum output s_i and carry output c_i). This very general model applies to current technology and is likely to hold for any foreseeable technologies. Except where noted, our results hold for this general model, and in fact apply so long as the function for c is non-decreasing in d, and the function for s is non-decreasing in b and also non-decreasing in d for fixed b. Some of our results apply to more restricted cases that reflect current technology.

Based on current technology, the time used by an adder to generate its outputs from its inputs can be normalized to XOR delay units, corresponding to the number of XOR gates (or approximate equivalent) traversed by the input signals. Thus, the values "$x_2 = 1$" and "$x_3 = 2$" can be used as the delays for the sum, corresponding to the number of XOR gates traversed by the b and d inputs, respectively. Similarly, the value "$y_3 = 1$" can be used for the carry delay, corresponding to the time needed for the d input to traverse two NAND gates, each having delay roughly half that of an XOR gate [3]. Under these conditions the global optimization problem can be viewed as minimizing the number of equivalent XOR gates on the longest path in the PPRT circuit. Thus our results apply to any technology as long as the number of equivalent XOR gates on a path remains the critical delay of the circuit, and the relative delays of XOR and NAND gates remain unchanged. Both these assumptions have continued to hold during the rapid development of logic technologies in the past [3]. For conciseness, we will refer to the case with the normalized values of $x_2 = 2$, $x_3 = 1$, $y_3 = 1$ as the *standard problem*.

Optimizing for delay using the full adder model described above leads to improved performance in real circuit designs. In [4] designs in 1 micron CMOS-ASIC technology were simulated using a timing simulator from LSI Logic [15]. The simulated delays closely matched those predicted by using the delays of the standard full adder model described above, and the new design outperformed competing designs by 11-25%. Thus the optimization problem we address here seems to be a sound model of actual circuit delays, and can provide substantial improvements in performance.

The PPRT designs we develop use local connections between gates and are similar in overall structure to classical PPRT designs. Thus there should be no special problems in their layout or wiring.

Our work differs from that of Paterson, Pippenger, and Zwick [7, 8, 9], who also looked at adding multiple columns of bits down to two bits per column. However, they only considered designs where each column uses an identical circuit, rather than designs which optimize across all columns which is our focus. Using global optimization, we are able to design multiplication circuits whose delays beat the upper bounds provable for their more restricted circuits.

In the next section, we describe the multiplier design problem in more detail. In Section 3 we study properties of optimal carry vectors and present further properties of circuits used in optimal TDM designs. In

Section 4, we study the objective of minimizing only the delay of an output bit in designing adder circuits and present an optimal strategy for doing this. In Section 5, we present a lower bound for the global problem derived by analyzing a relaxed version of the problem. In Section 6 we describe our program which uses the prior results to search for optimal PPRT circuits. Finally, in Section 7 we present some open problems related to this work.

2 The multiplier setting

We now give a more detailed description of the PPRT design problem. In the long multiplication method, we compute the product of the first number with each bit of the second and arrive at $2n-1$ addition problems (one along each column of the long multiplication table), the result of which gives the final value of the multiplication. We number the columns generated in the long multiplication table right to left (least significant to most significant) starting with 0 and ending with $2n-2$ (See Figure 1).

In constructing a PPRT for use in a mutiplier for two n-bit numbers, we assume that all of the n^2 partial product bits are available at time 0. The examples given in this section will all assume the standard problem, although the principles apply generally.

All the bits in column i represent i^{th} significant bits in the product, and thus have weight 2^i. A full adder working on three inputs of weight 2^j for any $j \geq 0$ produces a sum bit of weight 2^j and a carry bit of weight 2^{j+1}. Thus the carry bits from the full adders summing the bits in column i represent bits of weight 2^{i+1} and are fed as inputs to the addition problem in column $i+1$. The sorted list of times at which the carry bits are produced in the circuit for a column is called the *carry vector* for that column. Thus the inputs for each column consist of the partial product bits for that column available at time zero and the carry bits from the previous coulumn available at the times designated in the carry vector. The TDM thus consists of constructing the circuits for the columns in order of increasing significance based on the derived input times. The problem of building a fast PPRT using the TDM therefore reduces to one of finding adder circuits for the columns that yield the two sum bits with little delay and generate "good" carry vectors.

2.1 Two addition problems

The PPRT of a parallel multiplier adds the bits in each column i until only two bits of weight 2^i remain (as well as all the carry bits of weight 2^{i+1}). These bits are used to form two $(2n-1)$-bit numbers X and Y, where X_i and Y_i, the bits of weight 2^i in X and Y respectively are the two output bits from column i. X and Y are then fed into a carry propogate adder that adds two $(2n-1)$-bit numbers to produce the final multiplication value (See Figure 1). In the TDM each column circuit takes as input the partial product bits corresponding to that column and any carry bits generated by the previous column and adds them to produce two sum bits and a number of carry bits for

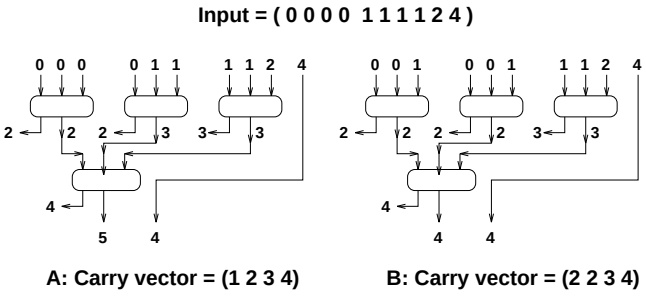

Figure 2: **Circuit A produces a better carry vector than circuit B while B achieves better final delay than A in summing all the input bits to two bits.**

input to the next column. Figure 2 (A) illustrates a circuit which takes 10 input bits of weight 2^i. The circuit produces two output bits of weight 2^i at times 4 and 5, and four carry bits of weight 2^{i+1} at times 1, 2, 3, and 4. The total time to carry out the multiplication is the time to generate partial product bits, complete the column additions in the PPRT, and add the final $(2n-1)$-bit numbers. Our goal in the multiplier design is to decrease this total time by minimizing the time when the last sum bit from any column is produced.

2.2 Adjusting the parity

It is not possible to produce two sum bits from a circuit that uses full adders alone when the number of bits to be summed is odd. Consider, e.g., the case when three bits are to be summed. Using a full adder on these bits produces a single sum bit. A full adder "consumes" three of the bits to be summed and produces one new bit to be summed, thus reducing the number of bits to be summed by two. If we start with an even number of bits to be summed, we can use full adders to get down to two sum bits as required for each column. Note that if we have $2(k+1)$ bits to be summed, we can do this using k full adders each of which produces a carry bit. However, starting with an odd number of bits results in a state where only one bit remains.

We fix this parity problem using a half adder, which takes two input bits and produces a sum and a carry bit. We denote by (a,b) a half adder that takes in two bits at times $a \leq b$, and produces a sum bit at time $s = b+v_2$ ($v_2 \geq 0$) and a carry bit at time $c = b+w_2$ ($w_2 \geq 0$) [4]. As in the case of full adders, for analysis relating to current technology we use values corresponding to normalized XOR delay units, i.e., $v_2 = 1$ and $w_2 = 0.5$ (for one AND gate). Since the half adder reduces the number of bits to be summed by one, a single half adder fixes the parity problem when the number of bits to be summed is odd. Following [4], we use the half adder on the earliest two inputs in the input vector, to change the parity of the number of bits to be even. To add $(2k+1)$ bits for a column in this way, we use a single half adder and $(k-1)$ full adders generating a total of k carry bits. The number of input bits and

gates for the columns are shown in Table 1. Note that the number of half adders and full adders used varies only with the size of the multiplication, and not with the manner in which they are inter-connected.

Table 1: **Number of Input Bits and Gates in a PPRT for n-bit Multiplication** ($2 \leq k \leq 2n-2$)

Column k:	$k \leq n-1$	$k = n$	$n+1 \leq k$
# 0 Bits	$k+1$	$k-1$	$2n-k-1$
# Carry-In Bits	$k-2$	$k-2$	$2n-k-1$
Total Input Bits	$2k-1$	$2k-3$	$4n-2k-2$
# Half Adders	1	1	0
# Full Adders	$k-2$	$k-3$	$2n-k-2$
Total Half Adders	$n-1$		
Total Full Adders	$(n-1)(n-3)$		

2.3 Circuit notation and definitions

We now present a formal definition of a the TDM.

Definition. A **TDM PPRT** uses column circuits to produce two sum bits for each column (except column 0). The circuit for each colum uses the minimum number of full adders (plus one half adder on the smallest inputs when the number of inputs is odd) needed to produce the output bits from the partial product bits for that column and the carry-out bits (if any) from the previous column.

The following definitions and notation will be used to simplify our discussion of column circuits. (For conciseness we will use *circuit* to mean a column circuit in a TDM PPRT.) We denote by I the vector representing the bits input to a circuit. We use $C(I)$ to denote the circuit C applied to input vector I. The values in I correspond to the *times* at which the signals are available in non-decreasing order. If C has m full adders and I consists of k input bits then $C(I)$ will produce a vector V_s of the times when the $k-2m$ sum bits become available in non-decreasing order and a similar vector V_c for the m carry bits.

We use the following definition when comparing input or output vectors:

Definition. For two vectors $V = (v_1, v_2, ..., v_k)$, $U = (u_1, u_2, ..., u_k)$, we say that V **dominates** U iff $v_i \leq u_i \ \forall i \in \{1, 2, ..., k\}$. We write $V \leq U$ to denote that V dominates U. If the two vectors are unequal then V **strictly dominates** U, and we can write $V < U$. Two vectors are **incomparable** if there exists i, j such that $v_i < u_i$ and $v_j > u_j$. A vector $V \in S$ is said to be **undominated** in S if no other vector in S strictly dominates it.

For example, if $U = (1,2,3)$, $V = (2,2,3)$, and $W = (1,2,4)$ then $U < V$, $U < W$, and V and W are incomparable.

We also extend the definition of dominance to circuits and gates: If m-adder circuits $C(I)$ and $C'(I)$ produce vectors V_c and V_s, and V_c' and V_s', respectively, then $C(I)$ *dominates* $C'(I)$ iff $V_c \leq V_c'$ and $V_s \leq V_s'$, and we use the notation $C(I) \leq C'(I)$. $C(I)$ *strictly dominates* $C'(I)$ if $C(I) \leq C'(I)$ and $V_c \neq V_c'$ or $V_s \neq V_s'$, and we can write $C(I) < C'(I)$. If no m-adder circuit for I strictly dominates $C(I)$, then we say that $C(I)$ is an *undominated circuit*. Note that an undominated column circuit for input I need not be part of an optimal PPRT since two circuits $C(I)$ and $C'(I)$ may be incomparable. However, if we construct a multiplier circuit by summing the bits in each column i (which are the original bits of weight 2^i and the carry bits from column $i-1$), it is clear that we can always construct an optimal circuit by using an undominated circuit for each column.

Similar to the case for circuits, we say that for two gates g_i and g_i', g_i *dominates* g_i' ($g_i \leq g_i'$) if $c_i \leq c_i'$ and $s_i \leq s_i'$. Note that in the case of gates we remove the restriction that the inputs be the same. This leads us immediately to the following observation:

Observation 2.1 The outputs s_i and c_i of gate g_i depend only on b_i and d_i. Therefore g_i dominates g_j if $b_i \leq b_j$ and $d_i \leq d_j$.

We now introduce notation that simplifies our analysis of circuits on the gate level. Given an input vector I we consider strategies for constructing circuits for I. In our constructions it is convenient to think of the circuits as being constructed in the following way: The first gate, g_1, has all three of its inputs from $I_0 = I$, we now update I_0 to I_1 which has the three inputs to g_1 removed, but s_1, the output sum bit of g_1, is added. We then construct g_2 using three inputs from I_1, and update I_1 to get I_2, and so on.

2.4 Trade-off between delay and good carry vectors

We now examine the problem of designing the column circuits for the TDM PPRT. Suppose we aim to produce the two bits from each of the columns within delay d. Then the circuit for each column must, for its given input vector I, produce the two sum bits in delay $d' \leq d$ along with a "good" carry vector. An optimal carry vector in this case must be undominated in the set of carry vectors for circuits $C(I)$ which produce their sum bits in time $\leq d'$. We now will show that such a carry vector may not be undominated in the set of all carry vectors generated by circuits for I.

There is often a trade-off between making the carry vector good in the sense that it is undominated and minimizing the delay of the output bits in a circuit. For example, in Figure 2 circuit A generates a carry vector that strictly dominates the carry vector of circuit B but does not achieve as good a delay for the two sum bits. The design of the PPRT thus involves a judicious choice of the delay of the sum bits produced in each of the columns coupled with a strategy for finding a circuit that achieves that delay and produces an undominated carry vector. I.e., a PPRT with optimal delay will have column circuits each of which produces an undominated output vector consisting of the carry vector concatenated with the larger of the two sum outputs.

2.5 3-greedy approach

In [4] a heuristic for TDM design is proposed that we will call the *3-greedy approach*. The 3-greedy approach is as follows: take for the inputs of each gate g_i the three smallest values in I_{i-1}. For example, the adder circuit shown in Figure 2 (A) is 3-greedy. Gate $g_1 = (0,0,0)$ takes as its inputs the three smallest values in $I_0 = I = (0,0,0,0,1,1,1,1,2,4)$, and generates a sum bit at time 2. Thus $I_1 = (0,1,1,1,1,2,2,4)$, and we build gate $g_2 = (0,1,1)$. Similarly, $I_2 = (1,1,2,2,3,4)$, $g_3 = (1,1,2)$, $I_3 = (2,3,3,4)$, $g_4 = (2,3,3)$, and $V_s = (4,5)$. Also, $V_c = (1,2,3,4)$.

Note that the 3-greedy strategy produces undominated circuits since the 3-greedy approach produces the lexicographically smallest carry vector, but it may be possible to produce circuits with better sum delays using other strategies.

3 Optimal column-addition circuits

In this section, we consider circuits that sum a vector of input bits and produce undominated output vectors. As mentioned earlier, the optimal PPRT uses a circuit of this form to sum the bits in each column. We show that we need only consider a restricted class of addition circuits that have certain nice properties.

We will now discuss the *2-greedy strategy* which is defined as follows: always construct gate g_i using the two smallest values in I_{i-1} plus a third value in I_{i-1}. The key fact which we will prove is that for any column circuit $C(I)$, there exists a 2-greedy column circuit $C'(I)$ which dominates $C(I)$. Thus in searching for optimal combinations of column circuits, we can restrict our attention to 2-greedy circuits.

3.1 The 2-greedy strategy

In this section we prove several properties of 2-greedy circuits. We begin with a few definitions.

Definition. A **lexicographic ordering** of the gates of a circuit $C(I)$ (or, loosely, a lexicographical ordering of $C(I)$) is one in which for each pair of gates $g_i = (a_i, b_i, d_i)$ and $g_j = (a_j, b_j, d_j)$, g_i precedes g_j in the ordering whenever:

- $a_i < a_j$; or
- $a_i = a_j$ and $b_i < b_j$; or
- $a_i = a_j$, $b_i = b_j$, and $d_i < d_j$.

If $a_i = a_j$, $b_i = b_j$, and $d_i = d_j$ then the gates can appear in either order. We assume that all the circuits $C(I)$ which we discuss have their gates numbered in lexicographic order. We shall show that for any 2-greedy circuit there always exists a construction order which is also a lexicographic order.

The following definitions give some of the ways that gates can be related to each other:

Definition. Gate g_j is an **immediate descendant** of gate g_i if s_i is an input to g_j. Similarly, g_j is a **descendant** of g_i if g_j is an immediate descendant of g_i or of a gate g_k that is a descendant of g_i.

Definition. Two gates are said to be **independent** if neither gate is a descendent of the other.

We begin by proving that when constructing a gate g_i we can always use three values in I_{i-1} rather than using an input which is the sum bit from a higher numbered gate. A circuit is *feedback-free* iff when the gates are numbered in lexicographic order, no gate g_i has an input which is the sum bit from g_j where $j > i$.

Lemma 3.1 For any circuit $C(I)$ which is not feedback-free, there is a feedback-free circuit $C'(I)$ such that $C'(I) \leq C(I)$.

Proof. If $C(I)$ is not feedback-free then there exists a gate g_i with input s_j, where $i < j$. We now show that we can restructure the circuit $C(I)$ to make it feedback-free without degrading the outputs of the circuit. By definition, $a_i \leq a_j < s_j$. Thus $s_j = b_i$ or $s_j = d_i$.

Case 1. $s_j = d_i$.

Case 1a. $b_i \leq b_j$.

$$\begin{aligned} g_j &= (a_j, b_j, d_j) \\ c_j &= d_j + y_3 \\ s_j &= \max(b_j + x_2, d_j + x_3) \\ g_i &= (a_i, b_i, s_j) \\ c_i &= s_j + y_3 \\ &= \max(b_j + x_2 + y_3, d_j + x_3 + y_3) \\ s_i &= \max(b_i + x_2, s_j + x_3) \\ &= \max(b_i + x_2, b_j + x_2 + x_3, d_j + 2x_3) \\ &= \max(b_j + x_2 + x_3, d_j + 2x_3) \end{aligned}$$

By rearranging inputs we can get:

$$\begin{aligned} g'_i &= (a_i, b_i, d_j) \\ c'_i &= d_j + y_3 \\ s'_i &= \max(b_i + x_2, d_j + x_3) \\ g'_j &= (a_j, b_j, s'_i) \\ c'_j &= s'_i + y_3 \\ &= \max(b_i + x_2 + y_3, d_j + x_3 + y_3) \\ s'_j &= \max(b_j + x_2, s'_i + x_3) \\ &= \max(b_j + x_2, b_i + x_2 + x_3, d_j + 2x_3) \end{aligned}$$

Thus by Observation 2.1 g'_i dominates g_j (since $b_i \leq b_j$). Further, $c'_j \leq c_i$ and $s'_j \leq s_i$ (because $b_i \leq b_j$), so g'_j dominates g_i. Therefore the sum and carry values from g'_i and g'_j are at least as good as those for g_j and g_i.

Case 1b. $b_i > b_j$. By rearranging inputs we can get $g'_i = (a_i, b_j, d_j)$, and $g'_j = (a_j, b_i, s'_i)$. Now g'_i has the same sum and carry as g_j, thus $s'_i = s_j (\geq b_i)$ and g'_j has the same sum and carry as g_i.

Case 2. $s_j = b_i$. In this case, we rearrange inputs to get $g'_i = (a_i, b_j, d_j)$, and $g'_j = (a_j, s'_i, d_i)$. As above, g'_i has the same sum and carry as g_j, thus $s'_i = s_j$ and g'_j has the same sum and carry as g_i.

Repeated applications of the above transformations to the smallest (i,j) pair violating the feedback-free property will convert $C(I)$ to the desired feedback-free circuit $C'(I) \leq C(I)$, proving the lemma. □

Lemma 3.2 For any feedback-free circuit $C(I)$ which is not 2-greedy there is a 2-greedy circuit $C'(I)$ such that $C'(I) \leq C(I)$.

Proof. Similar to the proof of Lemma 3.1, and omitted. □

The prior two lemmas show that given any circuit $C(I)$ we can convert it to a circuit $C'(I) \leq C(I)$ such that if the gates of $C'(I)$ are numbered lexicographically, each gate g_j in $C'(I)$ has as inputs the two smallest values in I_{j-1} plus a third input from I_{j-1}, i.e., $C'(I)$ can be constructed using the 2-greedy strategy.

Observation 3.3 Let C be a circuit constructible in lexicographic order using a 2-greedy approach. If g_i appears before g_j in the lexicographic ordering of C then g_i is created before g_j in a 2-greedy lexicographic construction of C and as a result, $a_i \leq b_i \leq a_j \leq b_j$.

3.2 Other constraints on undominated circuits

The following lemmas provide rules for minimizing the number of circuits that need to be constructed for an input vector I when attempting to find all undominated circuits $C(I)$. In particular, Corollary 3.8 ensures that we can restrict our search to certain "regular" circuits for I. The proofs for the following use similar techniques to the proofs for Lemmas 3.1 and 3.2 and are omitted.

Lemma 3.4 Let $C(I)$ be a 2-greedy circuit. Then there is a 2-greedy circuit $C'(I) \leq C(I)$ in which for each pair of gates g_i and g_j, if $i < j$ and $d_i \leq b_j$, then $d_i \leq a_j$.

Corollary 3.5 Let $C(I)$ be a 2-greedy circuit. Then there is a 2-greedy circuit $C'(I) \leq C(I)$ in which for each pair of gates g_i and g_j, if $i < j$ and $d_i = b_j$ then $a_j = b_j = d_i$.

Lemma 3.6 Let $C(I)$ be a 2-greedy circuit. Then there is a 2-greedy circuit $C'(I) \leq C(I)$ in which for each pair of gates g_i and g_j, if $i < j$ then $d_i \leq d_j$.

Lemma 3.7 Let $C(I)$ be a 2-greedy circuit, and let $x_2 \leq 2x_3$. Then there is a 2-greedy circuit $C'(I) \leq C(I)$ such that for each pair of gates g_i and g_j, if $i < j$ and $b_j < d_i \leq d_j$ then $d_j < d_i + x_2 - x_3$.

Note that Lemma 3.7 applies to the standard problem as described earlier, i.e., $x_2 = 2$, $x_3 = 1$, $y_3 = 1$. When $x_2 \leq x_3$ we define the following class of circuits:

Definition: A 2-greedy circuit C for a vector V that has no pairs of gates of the form excluded by Lemma 3.1, Lemma 3.2, Lemma 3.4, Lemma 3.6, and Lemma 3.7 is said to be in **regular form**, or, loosely, simply said to be **regular**.

Corollary 3.8 If $x_2 \leq 2x_3$, then for any circuit $C(I)$, there is a regular circuit $C'(I) \leq C(I)$.

The above lemmas can be used to severely limit an exhaustive search for an optimal solution to the standard problem, since they allow us to restrict the search to regular circuits for each input I.

4 Optimal delay circuits

In this section we consider the problem of finding for a given vector I the minimum delay circuit $C(I)$ that outputs a single sum bit (we will ignore the carry vector). We assume the standard problem previously described, where $x_2 = 2$ and $x_3 = 1$. We define a canonical circuit for each value of delay that can sum the maximum number of input bits for this delay, and show that a fitting strategy that attempts to greedily fit the input delays into the canonical circuit can be used to determine an optimal delay circuit for this set of inputs. Though we describe our strategy for the problem of summing a set of input bits down to a single bit within minimum delay, our method and analysis directly extend to the case when the input bits must be combined in full adders and reduced to some specified number of bits (possibly greater than one) within minimum delay.

4.1 Canonical circuits for a given delay

We define $S(t)$ as the maximum number of bits available at time zero, which can be added with full adders to produce a single sum by time t. We can write a simple recurrence for $S(t)$ as follows.

$$\begin{aligned} S(0) &= S(1) = 1 \\ S(t) &= S(t-1) + 2S(t-2) \quad t > 1 \end{aligned}$$

The recurrence follows from the observation that the best way to accommodate the most inputs completing in delay t is to have the last adder with output t from this circuit be $(t-2, t-2, t-1)$. Each of the three inputs to the last adder in this circuit is the output of a circuit of delay $t-2$ or $t-1$ which sums the maximum possible number of inputs [1]. This observation can also be used to easily infer that the circuit which sums $S(t)$ input bits available at time zero with delay t is unique. This unique circuit is termed the *Canonical circuit for delay t* and is denoted $C(t)$. Note that $C(t)$ can accommodate several inputs available at time 1, and that these 1 inputs feed adders of exactly two types: (0,0,1) and (1,1,2). Furthermore, all inputs with delay d for $d \geq 1$ feed into gates of the type $(d-1, d-1, d)$ or $(d, d, d+1)$. An illustration of $C(4)$ is in Figure 3.

Note that if our goal is to produce k sum bits within delay t, then at most $kS(t)$ bits can be summed. We define the *weight* of a vector V of input delays as $W(V) = S(v_1) + S(v_2) + ... + S(v_k)$.

Theorem 4.1 If $W(V) > k \cdot S(t)$, then inputs with delay V cannot be summed to k bits in delay $d \leq t$.

This theorem is useful for proving lower bounds on multiplier circuits since it bounds the delay for a column given a carry vector.

[1] The recurrence solves to $S(t) = \frac{2^{t+1} + (-1)^t}{3}$ for $t \geq 0$.

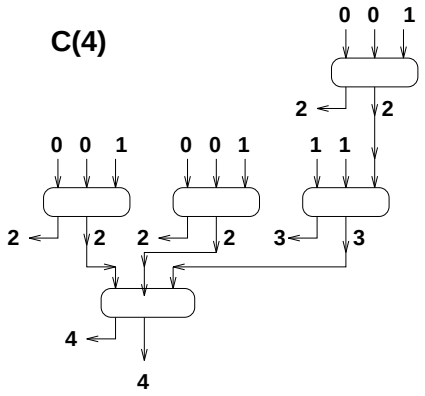

Figure 3: **The canonical circuit for delay four.**

4.2 The fitting strategy

We can now describe a fitting strategy to determine, given an input vector I, whether I can be summed to a single bit using a circuit of full adders in delay t. For technical reasons we assume that the input vector has an odd number of bits to be summed [2]. We use the canonical circuit $C(t)$, and try to fit the input signals in nonincreasing order of delays into appropriate positions in $C(t)$. Thus we work on the larger inputs earlier and fit an input with delay t' into a canonical subcircuit for delay t' in $C(t)$. After fitting a signal, we mark the whole fitted subcircuit and proceed to fit the remaining inputs in the unmarked portions of $C(t)$. (Input bits of time 0 can be fit into unmarked 1's in the canonical circuit). If we succeed in fitting all the input signals, it is easy to derive a circuit that sums all these inputs in delay t. In the case when all the input delays cannot be fitted in $C(t)$, we demonstrate that there is no circuit for these inputs completing in delay t.

Before we prove the correctness of the fitting strategy, we show that any delay optimal circuit can be put in a normal form that resembles the canonical circuits. I.e., that any circuit C of delay t can be modified into a circuit all of whose adders are drawn from the set $(k, k, k+1)$ where $0 \leq k \leq t-2$.

Lemma 4.2 Suppose C is a circuit that sums a vector V of inputs within delay t. We can convert C to a circuit which sums the vector V (augmented possibly with many zeros and ones) within delay t, such that every adder in the modified circuit is of the form $(k, k, k+1)$ for some $k \geq 0$.

Proof sketch. The idea is to show that whenever an adder is not of the required form, we can modify the inputs to that adder by expanding it with canonical subcircuits and possibly accommodating a few extra 0's to be summed, but not increasing the delay of the final sum bit. □

[2] When the number of bits is even, we assume that we sum the least two bits in a half adder as in the strategy in [4] and thus produce an odd number of bits to sum.

Next we prove the correctness of the fitting strategy.

Lemma 4.3 Consider the fitting strategy working on a canonical circuit $C(t)$ and a (sorted) vector V whose elements lie in the range $\{t-1, \ldots, t-i\}$ for some $i > 0$. Suppose the strategy successfully fits V into $C(t)$. After fitting V, let N be the maximum number of additional inputs of delay $t-i$ that can be fitted into the current circuit C. Then the vector V augmented with $N+1$ or more $(t-i)$'s cannot be summed in delay t.

We can use Lemma 4.3 to show the correctness of the fitting strategy.

Theorem 4.4 The fitting strategy finds a delay t circuit for the input vector V if one exists.

The following theorem shows that the fitting strategy can be implemented efficiently.

Theorem 4.5 Given a sorted input vector, the fitting strategy can be implemented to run in time linear in the size of the input vector.

5 A lower bound

Consider the design of the multiplier circuit with a full adder that, for inputs $a \leq b \leq d$, produces a sum at time $s = d + x_3$ and a carry at time $c = d + y_3$. Note that we do not claim that such a full adder can be built, but simply use this as a relaxed version of the original problem. Using techniques as in Lemma 3.2, we can show that for this relaxed problem, the 3-greedy approach described in Section 2.5 (i.e., repeatedly putting the three earliest bits into a full adder) is globally optimal for both sum and carry [3]. Note that the 3-greedy strategy can also be considered *carry-greedy*, since this choice of inputs reflects a local minimization of the carry vector generated.

The optimality of the 3-greedy strategy for the relaxed problem motivates the following lower bound for the original problem. Since the relaxed problem is a less constrained problem than the original, the optimal delay of a multiplier for the relaxed problem is a lower bound on the delay of the multiplier for the original problem. Furthermore, if we determine using other techniques a lower bound on the carry vector being input into a certain column in the multiplier, we can apply the 3-greedy strategy to the remaining columns under the relaxed sum output model to derive a lower bound for the original problem. In this way, a partial lower bound computation (using, e.g., arguments as in Lemmas 3.2 to 3.4) can be effectively completed using this strategy on a relaxed model of the adder gates.

6 Program and results

The results in the last three sections provide a strong set of tools for searching for a minimum delay multiplication circuit. We have developed a program

[3] If the number of inputs is even, then we assume as before that the two smallest are fed into a half adder.

which does a search over TDM PPRT circuits. The program restricts its search to regular circuits (as described in Section 3). For each column it considers all carry-vectors which could be input to the column, and for each such input generates all the regular circuits. This set of regular circuits is then pruned to contain only those circuits with undominated carry vectors and sum delays. The carry vectors from these circuits are used as inputs to the next column. We can also use the lower bounding techniques from Sections 4 and 5 to prune those carry vectors which must produce delays at least as large as our best solution to date. Using this program we showed that the 3-greedy method gives the optimal maximum delay for most problem sizes up to 44 bits. Further, it is within one of the optimal in those cases where it is not optimal. Our results are summarized in Table 2.

Table 2: **3-greedy vs. Optimal Delays for PPRT in Normalized XOR Delays (Standard Problem)**

Bit Size	Max Delay 3-G	Max Delay Opt	Program CPU Time
7-8	5	5	
9-10	6	6	
11-12	7	7	
13	7.5	7	
14-16	8	8	
17	9	8	
18-20	9	9	
21	9.5	9	
22	10	9	$<$ 3.3 sec
23-26	10	10	$<$ 39 sec
27-28	11	10	$<$ 1 min 6 sec
29-35	11	11	$<$ 2 min 37 sec
36	12	11	$<$ 3 hours 19 min
37-44	12	12	$<$ 6 hours
45-57	13	??	$>$ 2 months
58	13.5	??	
59-76	14	??	
77-98	15	??	
99-128	16	??	

The current version of the program solves a problem of size 16 in a few seconds, but requires hours to solve a 39 bit problem. Further refinements will need to be made before we can solve larger problems of size 64 bits or larger.

7 Conclusions

We have presented several techniques for analyzing the optimality of PPRTs designed using full adder cells. Many interesting issues remain open. We plan to do a simulation of a full design using the circuits produced by our program to validate that our abstract model of delay results in good performance in practice. We also plan to improve our search program to solve larger problems, and are investigating parallel solutions to help speed up the search. Another interesting issue is to optimize the final carry-propagate adder used by taking into account that the input bits arrive at different times. Preliminary work on this is in [5]. Note that our results on the optimality of regular circuits apply to both the *sum* and *carry* vectors produced. Thus a search over regular circuits is sufficient to find optimal PPRT circuits for feeding the final carry-propagate adder. A final topic is to consider optimizing over a broader class of design strategies which allow other uses of half adders and even new components.

References

[1] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, McGraw Hill, 1990.

[2] L. Dadda, "Some Schemes for Parallel Multipliers," *Alta Frequenza*, Vol. 34, pp. 349-356, March 1965.

[3] K. Hwang, *Computer Arithmetic: Principles, Architecture and Design*, John Wiley and Sons, 1979.

[4] V. G. Oklobdzija, D. Villeger, S. S. Liu, "A method for speed optimized partial product reduction and generation of fast parallel multipliers using an algorithmic approach," in press *IEEE Transaction on Computers*, 1995.

[5] V. G. Oklobdzija and D. Villeger, "Optimization and analysis of a carry-propagate adder under the non-uniform signal arrival profile," in preparation.

[6] V. G. Oklobdzija and D. Villeger, "Improving Multiplier Design by Using Improved Column Compression Tree and Optimized Final Adder in CMOS Technology", in press, IEEE Transactions on VLSI, 1995.

[7] M. S. Paterson, N. Pippenger and U. Zwick, "Faster circuits and shorter formulae for multiple addition, multiplication and symmetric Boolean functions," *Proceedings of the 31st Foundations of Computer Science* (1990), 642-650.

[8] M. S. Paterson, N. Pippenger and U. Zwick, "Optimal carry save networks," *Boolean functional complexity: Selected papers from the LMS symposium, Durham 1990*, Cambridge University Press (1992).

[9] M. S. Paterson and U. Zwick, "Shallow multiplication circuits and wise financial investments," *Proceedings of the 24th Symposium on the Theory of Computing* (1992), 429-437.

[10] M.R. Santoro, "Design and clocking of VLSI multipliers," Ph. D. dissertation, Technical report No. CSL-TR-89-397, October 1989.

[11] P. Song, G. De Michelli, "Circuit and architecture tradeoffs for high speed multiplication," *IEEE Journal of Solid State Circuits*, Vol,. 26, No. 9, September 1991.

[12] W. J. Stenzel, "A compact high speed parallel multiplication scheme," *IEEE Trans. on Computers*, Vol C-26, pp. 948-957, February 1977.

[13] E. Swartzlander, *Computer Arithmetic*, Vol. 1&2, IEEE Computer Society Press, 1990.

[14] C. S. Wallace, "A Suggestion for a Fast Multiplier," *IEEE Transaction on Computers*, Vol. EC 13, pp. 14-17, February 1964.

[15] 1.0-Micron Array-Based Products Databook, LSI Logic Corporation, September 1991.