On The Complexity Of Booth Recoding

Wolfgang J. Paul *   Peter-Michael Seidel * †

Abstract

We formalize and prove the folklore theorems that Booth recoding improves the cost and cycle time of 'standard' multipliers by certain constant factors. We also analyze the number of full adders in certain 4/2-trees.

Keywords : multiplier design, booth recoding, VLSI model, layout analysis, delay and area complexity, wire effects.

1 Introduction

Addition trees are the central part in the design of fixed point multipliers. They produce from a sequence of partial products a carry save representation of the final product [29] [7] [12] [21] [11] [8] [30] [25]. Booth recoding [6] [24] [15] [5] [1] is a classical method which cuts down the number of partial products in an addition tree by a factor of 2 or 3 at the expense of a more complex generation of partial products.

In general, VLSI designers tend to implement Booth recoding, and a widely accepted rule of thumb says, that Booth recoding improves both the cost and the speed of multipliers by some constant factor around 1/4. In [1] [5] [3] [2] [4] however the usefulness of Booth recoding is challenged altogether. Finally, in an ambitious case study of around 1000 concrete designs [3] was performed, and for some technologies with small wire delays the fastest multipliers turned out *not* to use Booth recoding.

In spite of the obvious importance of the concept the (potential) benefits of Booth recoding have apparently received no theoretical treatment yet. This is probably due to the following facts:

- the standard models of circuit complexity [31] and VLSI complexity [26] [27] are only trusted to be exact up to constant factors.
- the standard theoretical VLSI model ignores wire delays.
- Conway/Mead style rules [16] [9] *do* consider wire delays, but do not invite paper and pencil analysis due to their level of detail.
- trivial addition trees with linear circuit depth have nice and regular layouts. But Wallace trees [29] [8] [25] and even 4/2-trees [11] [12] [14] [32] have more irregular layouts. In the published literature these layouts tend not to be specified at a level of detail which permits, say, to read off the length of wires readily.

In this paper we introduce a VLSI model which accounts – in a hopefully meaningful way – for constant factors as well as wire delays and which is at the same time simple enough to permit the analysis of large circuits. The model depends on a parameter ν which specifies the influence of the wire delays. In this model we study two standard designs of partial product generation and addition trees: the simple linear depth construction and a carefully chosen variant of 4/2-trees. We show, that Booth encoding improves for *all* reasonable values of ν both

*Department 14: Computer Science, University of Saarland, 66123 Saarbruecken, Germany. E-mail: {wjp,pmseidel}@cs.uni-sb.de.

†Supported by the PhD program "Efficiency and complexity of algorithms and computers" of the DFG.

the delay and the area of the resulting VLSI layouts by constant factors between X and Y minus low order terms which depend on ν and the length n of the operands. For $n = 24$ and $n = 53$ (the length of significands in the IEEE floating point format [10]) we specify the gain exactly.

The paper is organized in the following way. In section 2 we review Booth recoding as explained in [5] [13]. Section 3 provides a combinatorial lemma which will subsequently permit to count the number of full adders in certain 4/2-trees. In Section 4 we analyze the circuit complexity of Booth recoding. In section 5 we introduce the VLSI model. The detailed circuit model from [18] is combined with a linear delay model for nets of wires. Section 6 contains the specification and analysis of layouts. The layouts for 4/2-trees follow partly a suggestion from [28]. We conclude in section 7 and list some further work.

2 Preliminaries

For

$$a = a[n-1:0] = (a[n-1], \dots, a[0]) \in \{0,1\}^n$$

we denote by

$$< a >= \sum_{i=0}^{n-1} a[i] \cdot 2^i$$

the number represented by a, if we interpret it as a binary number. Conversely, for $p \in \{0, \dots, 2^n - 1\}$ we denote by $bin_n(p)$ the n-bit binary representation of p.
For $x \in \{0,1\}^n$ and $s \in \{0,1\}$ we define

$$x \oplus s = (x[n-1] \oplus s, \dots, x[0] \oplus s).$$

An (n,m)-*multiplier* is a circuit with n inputs $a = a[n-1:0]$, m inputs $b = b[m-1:0]$ and $n+m$ outputs $p = p[n+m-1:0]$ such that $< a > \cdot < b > = < p >$ holds.

Figure 1: $S_{j,k}$ in carry-save representation

2.1 Non-Booth

For $j \in \{0, \dots m-1\}$ and $k, h \in \{1, \dots m\}$ we define the partial sums

$$\begin{aligned} S_{j,k} &= \sum_{t=j}^{j+k-1} < a > \cdot b[t] \cdot 2^t \\ &= 2^j \cdot \sum_{t=0}^{k-1} < a > \cdot b[t+j] \cdot 2^{t+j} \\ &< 2^j \cdot \sum_{t=0}^{k-1} (2^n - 1) \cdot 2^{t+j} \\ &< 2^{j+n+k}. \end{aligned}$$

Then we have

$$\begin{aligned} S_{j,1} &= < a > \cdot b[j] \cdot 2^j \\ S_{j,k+h} &= S_{j,k} + S_{j+k,h} \end{aligned}$$

and

$$\begin{aligned} < a > \cdot < b > &= S_{0,m} \\ &= S_{0,m-1} + S_{m-1,1}. \end{aligned}$$

Because $S_{j,k}$ is a multiple of 2^j it has a binary representation with j trailing zeros. Because $S_{j,k}$ is smaller than 2^{j+n+k} it has a binary representation of length $n + j + k$ (see Fig. 1).

2.2 Booth Recoding

In the simplest form of Booth Recoding (called Booth-2) the multiplicator is recoded as suggested in Fig. 2.

With $b[m+1] = b[m] = b[-1] = 0$ and $m' = \lceil (m+1)/2 \rceil$ one writes

$$< b >= 2 < b > - < b >= \sum_{j=0}^{m'-1} B_{2j} \cdot 4^j$$

where

$$\begin{aligned} B_{2j} &= 2b[2j] + b[2j-1] - 2b[2j+1] - b[2j] \\ &= -2b[2j+1] + b[2j] + b[2j-1]. \end{aligned}$$

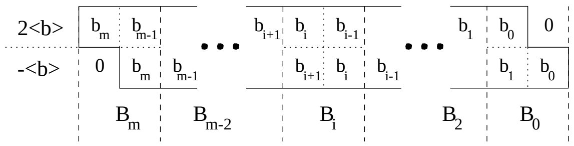

Figure 2: Booth digits B_{2j}

The numbers $B_{2j} \in \{-2, -1, 0, 1, 2\}$ are called *Booth digits* and we define their sign bits s_{2j} by

$$s_{2j} = \begin{cases} 0 & \text{if} \quad B_{2j} \geq 0 \\ 1 & \text{if} \quad B_{2j} < 0. \end{cases}$$

With

$$\begin{aligned} C_{2j} &= \ <a> \cdot B_{2j} \in \{-2^{n+1}+2, \ldots, 2^{n+1}-2\} \\ D_{2j} &= \ <a> \cdot |B_{2j}| \in \{0, \ldots, 2^{n+1}-2\} \\ d_{2j} &= bin_{n+1}(D_{2j}) \end{aligned}$$

the product can be computed from the sums

$$\begin{aligned} <a> \cdot <b> &= \sum_{j=0}^{m'-1} <a> B_{2j} \cdot 4^j \\ &= \sum_{j=0}^{m'-1} C_{2j} \cdot 4^j \\ &= \sum_{j=0}^{m'-1} s_{2j} \cdot D_{2j} \cdot 4^j. \end{aligned}$$

In order to avoid negative numbers C_{2j} one sums the positive E_{2j} instead.

$$\begin{aligned} E_{2j} &= C_{2j} + 3 \cdot 2^{n+1} \\ E_0 &= C_0 + 4 \cdot 2^{n+1} \\ e_{2j} &= bin_{n+3}(E_{2j}) \\ e_0 &= bin_{n+4}(E_0). \end{aligned}$$

This is illustrated in Fig. 3. The additional terms sum to

$$\begin{aligned} 2^{n+1}(1 + 3 \cdot \sum_{j=0}^{m'-1} 4^j) &= 2^{n+1}(1 + 3 \cdot \frac{4^{m'}-1}{3}) \\ &= 2^{n+1+2 \cdot m'}. \end{aligned}$$

Because $2 \cdot m' > m$ these terms are congruent to zero modulo 2^{n+m}. Thus

$$<a> \cdot <b> \equiv \sum_{j=0}^{m'-1} E_{2j} \cdot 4^j \bmod 2^{n+m}.$$

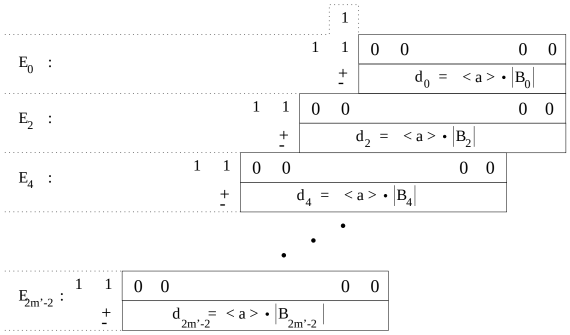

Figure 3: Summation of the E_{2j}

Lemma 1 *The e_{2j} can be computed by $< e_{2j} > \ = \ < 1\overline{s_{2j}}, d_{2j} \oplus s_{2j} > + \ s_{2j}$ and $< e_0 > = < \overline{s_0} s_0 s_0, d_0 \oplus s_0 > + s_0$.*

Proof: follows from the standard subtraction algorithm for binary numbers. □

By Lemma 1, computation of the numbers

$$\begin{aligned} F_{2j} &= E_{2j} - s_{2j} \\ f_{2j} &= bin_{n+3}(F_{2j}) \\ f_0 &= bin_{n+4}(F_0) \end{aligned}$$

is easy, namely

$$\begin{aligned} f_{2j} &= (1\overline{s_{2j}}, d_{2j} \oplus s_{2j}) \\ f_0 &= (\overline{s_0} s_0 s_0, d_0 \oplus s_0) \end{aligned}$$

Instead of adding the sign bits s_{2j} to the numbers F_{2j} one incorporates them at the proper position into the representation of F_{2j+2} as suggested in Fig. 4. The last sign bit does not create a problem, because $B_{2m'-2}$ is always non-negative. Formally, let

$$\begin{aligned} g_{2j} &= (f_{2j}, 0 s_{2j-2}) \in \{0,1\}^{n+5} \\ g_0 &= (f_{2j}, 00) \in \{0,1\}^{n+6}. \end{aligned}$$

Then

$$< g_{2j} > = 4 \cdot < f_{2j} > + s_{2j-2}$$

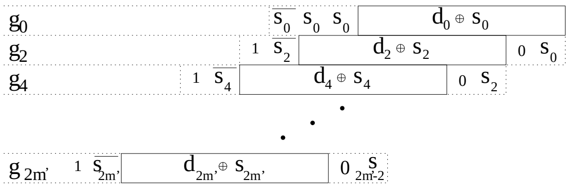

Figure 4: Partial product construction

and hence

$$\sum_{j=0}^{m'-1} E_{2j} \cdot 4^j = \sum_{j=0}^{m'-1} < g_{2j} > \cdot 4^{j-1}.$$

We define

$$S'_{2j,2k} \quad = \quad \sum_{t=j}^{j+k-1} < g_{2t} > \cdot 4^{t-1}.$$

Then we have

$$\begin{aligned} S'_{2j,2} &= < g_{2j} > \cdot 4^{j-1} \\ S'_{2j,2(k+h)} &= S'_{2j,2(k+h)} + S'_{2(j+k),2h} \end{aligned}$$

and

Lemma 2 *$S'_{2j,2k}$ is a multiple of 2^{2j-2} and $S'_{2j,2k} < 2^{n+2j+2k+2}$. Therefore, at most $n + 2k + 4$ non-zero positions are necesary to represent $S'_{2j,2k}$ in both carry-save or binary form.*

Proof: easy induction on k. $\square$

3 A combinatorial lemma

Let T be a complete binary tree with depth μ. We number the levels ℓ from the leaves to the root from 0 to μ. Each leaf u has a weight $W(u)$. For some natural number k we have $W(v) \in \{k, k+1\}$ for all leaves and the weights are nondecreasing from left to right. Let m be the sum of the weights of the leaves. For $\mu = 4, m = 53$ and $k = 3$ the leaves could for example have the weights 3333333333344444. For each subtree t of T we define $W(t) = \sum(W(u))$ where u ranges over all leaves of t. For each interior node v of T we define $L(v)$ resp. $R(v)$ as the weight of the subtree rooted in the left resp. right son of v. We are interested in the sums

$$H_\ell = \sum_{Level\ \ell} L(v)$$

where v ranges over all nodes of level ℓ and in

$$H = \sum_{\ell=0}^{\mu-1} H_\ell.$$

We show

Lemma 3 *$H \leq (\mu \cdot m)/2$.*

Proof: By induction on the levels of T one shows that in each level weights are nondecreasing from left to right and their sum is m. Hence

$$\begin{aligned} 2H_\ell &\leq \sum_{Level\ \ell} L(v) + \sum_{Level\ \ell} R(v) \\ &= \sum_{Level\ \ell} W(v) \\ &= m \end{aligned}$$

and the lemma follows. $\square$

In the above estimate we have replaced each weight $L(v)$ by the arithmetic mean of $L(v)$ and $R(v)$ hereby overestimating $L(v)$ by

$$\begin{aligned} h(v) &= (L(v) + R(v))/2 - L(v) \\ &= (R(v) - L(v))/2). \end{aligned}$$

For each node v in level ℓ the 2^ℓ leaves of the subtree rooted in v form a continuous subsequence of the leaves of T. Hence all nodes in level ℓ except at most one have weights in $\{k \cdot 2^\ell, (k+1) \cdot 2^\ell\}$. Therefore, in each level ℓ there is at most one node v_ℓ such that $h(v_\ell) \neq 0$. We set $h(\ell) = h(v_\ell)$, if it exists. Otherwise we set $h(\ell) = 0$. It follows that

$$H = (m \cdot \mu)/2 - \sum_{\ell=0}^{\mu-1} h(\ell)$$

In the above example we have $h_0 = 1/2, h_1 = 1/2, h_2 = 3/2, h_3 = 5/2$ and $H = 101$. For $\mu = 3, m = 27$ and weights 33333444 we have $h_0 = 1/2, h_1 = 1/2, h_2 = 3/2$ and $H = 38$.

Motorola	Not	$Nand$ Nor	And Or	Xor $Xnor$
delay	1	1	2	2
cost	1	2	2	4

Figure 5: Motorola technology parameters

b[i+1:i-1]	B[i]	b[i+1:i-1]	B[i]
000	0	100	-2
001	1	101	-1
010	1	110	-1
011	2	111	-0

Figure 6: Booth digit representations.

4 Gates

In this section we use a simple gate model [18] to analyze delay and cost of the multiplication circuits considering only the influence of gates. The delay is the maximum gate delay of all paths from input bits $a[i]$ and $b[j]$ to output bits $p[k]$. The cost is computed as the cumulative cost of all gates used. For this purpose the basic gate delays and costs can be extracted from any design system and will be used for the Motorola technology [19] (Fig. 5).

4.1 Partial Product Generation

4.1.1 non-Booth

One has to compute and shift the binary representations of the numbers

$$< a > \cdot b[j] = < a[n-1] \wedge b[j], \ldots, a[0] \wedge b[j] >$$

The $n \cdot m$ *And*-gates have a cost of $2 \cdot n \cdot m$. As they are used in parallel, they have a total delay of 2.

4.1.2 Booth

The binary representations of the numbers $S'_{2j,2}$ must be computed (see Fig. 4). These are

$$\begin{aligned} g_{2j} &= (1\overline{s_{2j}}, d_{2j} \oplus s_{2j}, 0s_{2j-2}) \\ g_0 &= (\overline{s_0}s_0s_0, d_0 \oplus s_0, 00) \end{aligned}$$

shifted by $2j-2$ bit positions.

The $d_{2j} = bin_{n+1}(< a > \cdot |B_{2j}|)$ are easily determined from B_{2j} and a by

$$d_{2j} = \begin{cases} (0,\ldots,0) & \text{if} \quad B_{2j} = 0 \\ (0,a) & \text{if} \quad |B_{2j}| = 1 \\ (a,0) & \text{if} \quad |B_{2j}| = 2. \end{cases}$$

For this computation two signals indicating $|B_{2j}| = 1$ and $|B_{2j}| = 2$ are necessary. We denote these by

$$b1_{2j} = \begin{cases} 1 & \text{if } |B_{2j}| = 1 \\ 0 & otherwise \end{cases}$$
$$b2_{2j} = \begin{cases} 1 & \text{if } |B_{2j}| = 2 \\ 0 & otherwise \end{cases}$$

and calculate them by the Booth decoder logic BD from Fig. 7a, that can be developed from Fig. 6. A Booth decoder has cost $C_{BD} = 11$ and delay $D_{BD} = 3$.

The selection logic SL from Fig. 7b directs either $a[i]$ or $a[i+1]$ or 0 to position $i+1$ and the inversion depending on s_{2j} yields $g_{2j}[i+1]$. In the first 2 (3 for rightmost partial product) and last 2 bit positions this logic is replaced by the simple signal of a sign bit, its inverse, a zero or a one. The selection logic has cost $C_{SL} = 10$ and delay $D_{SL} = 4$. As m' booth decoders are necessary, the decoding logic costs $m' \cdot C_{BD} = 11 \cdot m'$. The selection logic occurs for each partial product $n+1$-times. Therefore, the selection logics altogether have a cost of

$$m' \cdot (n+1) \cdot C_{SL} = 10 \cdot m' \cdot (n+1).$$

4.2 Redundant partial product addition

We study two standard constructions for the reduction of the partial products to a carry save

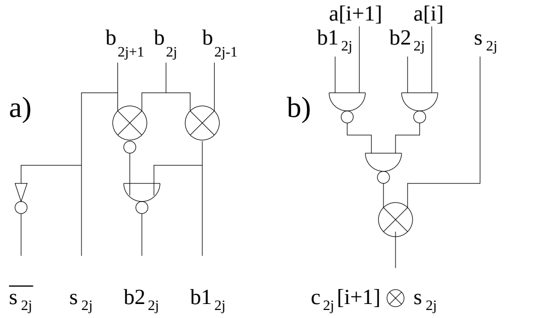

Figure 7: a) booth decoder b) selection logic

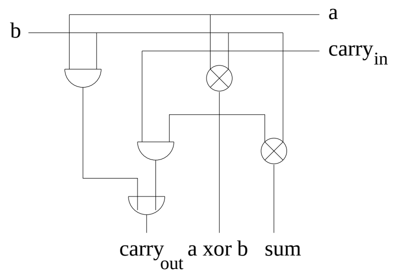

Figure 8: Circuit of a full adder.

representation of the product: plain multiplication arrays and 4/2-trees.

Each construction uses *k-bit carry save adders* (k-CSA) [11] [12] [21]. They are composed of k full adders working in parallel and hence they have cost $k \cdot C_{FA}$ and delay D_{FA}.

Cascading two k-CSA's one constructs a *k-bit 4/2-adder*, i.e. a circuit with $4k$ inputs $a[k-1:0]$, $b[k-1:0]$, $c[k-1:0]$, $d[k-1:0]$ and $2(k+1)$ outputs $s[k:0], t[k:0]$ such that

$$< a > + < b > + < c > + < d > \equiv$$
$$< s > + < t > \bmod 2^{n+1}$$

holds. The k-bit 4/2-adders constructed in this way have cost $2k \cdot C_{FA}$ and delay $2 \cdot D_{FA}$.

We use the full adders from Fig. 8. For these full adders we have $C_{FA} = 14$ and $D_{FA} = 6$.

4.2.1 Multiplication Arrays

In order to introduce techniques of analysis we review quite formally the construction of standard multiplication arrays [11] [12]. A carry save representation of $S_{0,3}$ can be computed by a single n-CSA (see Fig. 9). Exploiting

$$S_{0,t} = S_{0,t-1} + S_{t-1,1},$$

one can compute a carry save representation of $S_{0,t}$ from a carry save representation of $S_{0,t-1}$ and the binary representation of $S_{t-1,1}$ by an n-CSA. This works because both $S_{0,t-1}$ and $S_{t-1,1}$ can be represented with $n+t-1$ bits and because the binary representation of $S_{t-1,1}$ has $t-1$ trailing zeros (see Fig. 10). The $m-2$ many n-CSA's which are cascaded this way have cost $n(m-2) \cdot C_{FA}$ and delay $(m-2) \cdot D_{FA}$. The combined cost and delay for (non booth) partial product generation and the multiplication array are

$$\begin{aligned} C_1 &= n(m-2) \cdot C_{FA} + nm \cdot C_{AND} \\ &= 16nm - 28n \\ D_1 &= (m-2) \cdot D_{FA} + D_{AND} \\ &= 6m - 10. \end{aligned}$$

With Booth recoding one has only to sum the m' (representations of) partial products g_{2j}. Each partial product has length $n' = n + 5$ except g_0 which has $n'+1$ bits. Arguing with the sums $S'_{0,2t}$ in place of the sums $S_{0,t}$ one shows that $(m'-2)$ many (n')-CSA's suffice to sum the partial products in a multiplication array with Booth2 recoding. Taking into account the partial product generation one obtains cost and delay

$$\begin{aligned} C'_1 &= (n') \cdot (m'-2) \cdot C_{FA} + \\ &+ (n+1)m' \cdot C_{SL} + m' \cdot C_{BD} \\ &= 24nm' + 91m' - 28n - 140 \\ D'_1 &= (m'-2) \cdot D_{FA} + D_{SL} + D_{BD} \\ &= 6m' - 5. \end{aligned}$$

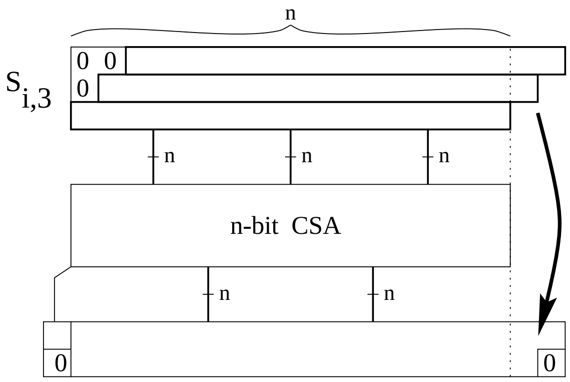

Figure 9: Partial compression of $S_{i,3}$.

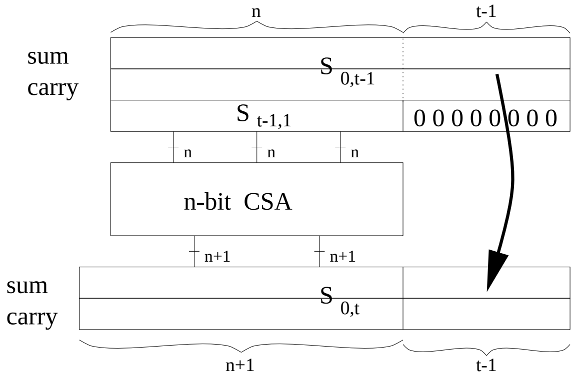

Figure 10: Partial compression of $S_{0,t-1}$ and $S_{t-1,1}$.

For $n = m = 53$ we get

$$\begin{array}{rcccl} C_1'/C_1 & = & 35457/43460 & = & 81,6\% \\ D_1'/D_1 & = & 157/308 & = & 50,9\%. \end{array}$$

For $n = m = 24$ we get

$$\begin{array}{rcccl} C_1'/C_1 & = & 8139/8544 & = & 95,2\% \\ D_1'/D_1 & = & 73/134 & = & 54,5\%. \end{array}$$

Asymptotically C_1'/C_1 tends to $12/16 = 75\%$ and D_1'/D_1 tends to $3/6 = 50\%$.

Thus, if multiplication arrays would yield the best known multipliers the case for Booth recoding would be easy and convincing (and hardly worth a paper).

4.3 Analysis 4/2-trees

The situation changes in two respects when we consider addition trees with logarithmic delay like Wallace trees [29] [8] [25] or 4/2-trees [11] [12] [14] [7] [32]. First, counting gates in such a tree becomes a somewhat nontrivial problem. Second, Booth recoding only yields a marginal saving in time.

4.3.1 Non Booth

We construct a specific family of 4/2 trees which happens to be accessible to analysis. The nodes of the tree are either 3/2-adders or 4/2-adders. The representations of the m partial products $S_{0,1}, \ldots S_{m-1,1}$ will be fed into the leaves of the tree *from right to left.* Let $M = 2^{\lceil \log m \rceil}$ be the smallest power of two greater or equal to m. Let $\mu = \log(M/4)$. We will construct the entire addition tree T by two parts as shown in Fig. 11.

The lower regular part is a complete binary tree of depth $\mu - 1$ consisting entirely of 4/2-adders. It has $M/8$ many 4/2-adders as leaves. For the top level of the tree we distinguish two cases. If $3M/4 \leq m \leq M$ we use $a = m - 3M/4$ many 4/2 adders and $M/4 - a$ many 3/2 adders. We arrange the 3/2-adders of the top level of the tree at the *left* of the 4/2-adders.

If $M/2 < m < 3M/4$ we use in the top level only $b = m - M/2$ many 3/2-adders (and we feed $m - 3b$ representations of partial products directly into the lower part). We arrange the 3/2-adders at the *right* end of the tree.

For $m = 24$ we have $m \geq 24$, and use 8 3/2 adders as leaves. For $m = 53$ we have $m \geq 48$, and we use 11 3/2 adders and 5 4/2 adders as leaves. We only analyze the first case explicitly.

If all 3/2-adders and 4/2-adders in the tree would consist of exactly n resp. $2n$ full adders,

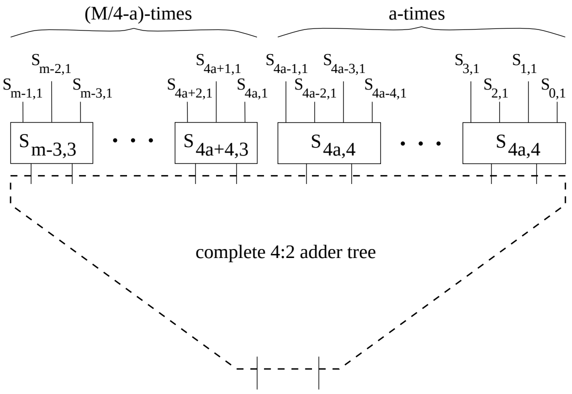

Figure 11: adder tree construction

then the tree would have a total of

$$F = n \cdot (m - 2)$$

full adders, because every 3/2-adder reduces the number of partial products by 1 and every 4/2-adder reduces the number of partial products by 2. It remains to estimate the number of *excess full adders* in the tree.

Every leaf in the tree which is a 3/2-adder computes a sum $S_{i,3}$. Thus n full adders suffice (see Fig. 9). A leaf of the tree which is a 4/2-adder computes a sum $S_{i,4}$. It can be simplified as shown in Fig. 12 such that $2n$ full adders suffice. Thus, in the top level of the tree there are no excess full adders.

Each 4/2-adder u in the lower portion of the tree performs a computation of the form

$$S_{i,k+h} = S_{i,k} + S_{i+k,h},$$

where a carry save representation of $S_{i,k}$ is provided by the right son of u and $S_{i+k,h}$ is provided by the left son of u. By the results of section 3 we are in the situation of Fig. 13. Hence node u has $2h$ excess full adders.

Referring to section 3 we label each leaf u of the tree with the number of partial products it sums, i.e. with $W(u) = 3$ if u is a 3/2-adder and with $W(u) = 4$ if it is a 4/2 adder, then

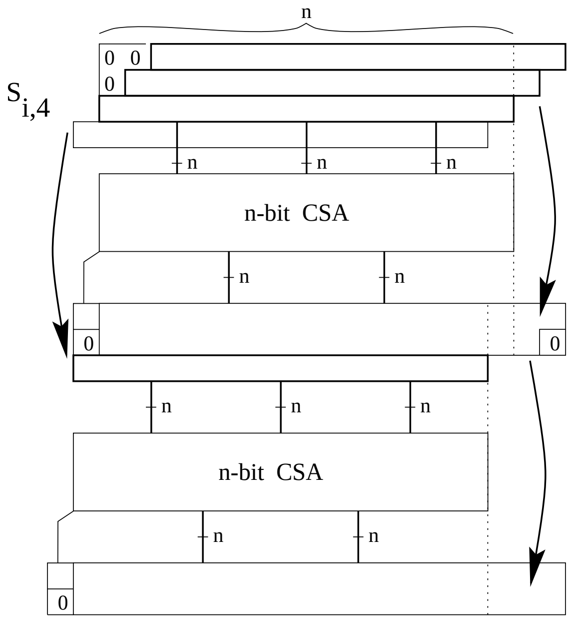

Figure 12: Partial compression of $S_{i,4}$.

$h = L(u)$ and for the number $E = 2H$ of excess full adders in the tree we have

$$\begin{aligned} E &= (m \cdot \mu) - 2 \cdot \sum_{\ell=0}^{\mu-1} h_\ell \\ &\leq (m \cdot \mu) \end{aligned}$$

This implies, that the 4/2-trees constructed above have $nm + o(nm)$ full adders.

The proof only depends on the fact that for each interior node u the left son of u sums less partial products than the right son of u. Hence it applies to many other balanced addition trees. One hardly dares to state or prove such a folklore result because it is 'obviously' known. Unfortunately we have not been able to locate it in the literature and we need it later.

With partial product generation we get cost and delay

$$\begin{aligned} C_2 &= (n \cdot (m-2) + E) \cdot C_{FA} + nm \cdot C_{AND} \\ &= 16nm + o(nm) \\ D_2 &= 2(\mu + 1) \cdot D_{FA} + D_{AND} \\ &= 12\mu + 14. \end{aligned}$$

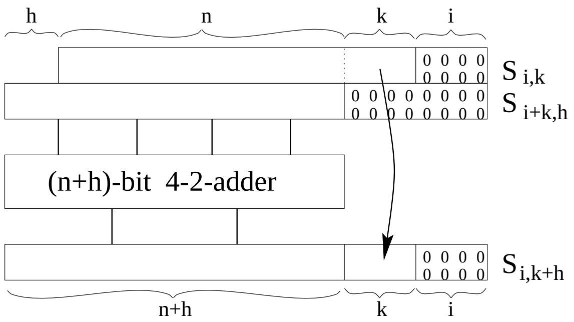

Figure 13: Partial compression of $S_{h,k}$.

4.3.2 Booth

Let $M' = 2^{\lceil \log m' \rceil}$ the smallest power of two greater or equal m' and let $\mu' = \log(M'/4)$.

For $m \in \{24, 53\}$ we have $3/4 \cdot M' \leq m' < M'$. We proceed as above and only analyze this case. The standard length of 3/2-adders and 4/2-adders is now $n' = n + 5$ bits; longer operands require excess full adders. Let E' be the number of excess full adders

Considering the sums S' instead of the sums S one shows that the top level of the tree has no excess full adders. Let H' be the sum of labels of left sons in the resulting tree and for all ℓ let h'_ℓ be the correction term for level ℓ.

Because with Booth recoding successive partial products are shifted by 2 positions, we now have

$$E' = 4H'$$

and we get

$$\begin{aligned} E' &= 2(m' \cdot \mu') - 4 \cdot \sum \ell h'_\ell \\ &\leq 2(m' \cdot \mu'). \end{aligned}$$

Taking into account the partial product generation one obtains cost and delay

$$\begin{aligned} C'_2 &= (n' \cdot (m' - 2) + E') \cdot C_{FA} \\ &+ (n+1)m' \cdot C_{SL} + m' \cdot C_{BD} \\ &= 24nm' + o(nm') \\ D'_2 &= 2(\mu' + 1) \cdot D_{FA} + D_{BD} + D_{SL} \\ &= 12(\mu' + 1) + 7. \end{aligned}$$

For $n = m = 53$ we get

$$\begin{aligned} C'_2/C_2 &= 37305/46288 &= 80,6\% \\ D'_2/D_2 &= 55/62 &= 88,7\% \end{aligned}$$

and for $n = m = 24$ we get

$$\begin{aligned} C'_2/C_2 &= 8531/9552 &= 89,3\% \\ D'_2/D_2 &= 43/50 &= 86,0\%. \end{aligned}$$

Asymptotically C'_2/C_2 tends again to $12/16 = 75\%$. Unless m is a power of two, we have $\mu = \mu' + 1$ and $D_2 - D'_2 = 7$. Hence D'_2/D_2 tends to one as n grows large.

This contradicts the common opinion that Booth recoding saves a constant fraction of the delay (independent of n). We try to resolve this contradiction in the next section by considering layouts.

5 VLSI Model

In the circuit complexity model we could only show, that a constant fraction of the cost is saved by Booth recoding. In order to explain, why Booth recoding also saves a constant fraction of the run time we have to consider wire delays in VLSI layouts.

The layouts that we consider consist of simple rectangular circuits S connected by nets N. For circuits S we denote by $b(S)$ the breadth, $h(S)$ the height, $C(S)$ the gate count and $D(S)$ the combinatorial delay of S. We do not consider different delays for different inputs and outputs of the same simple circuit in order to keep the analysis simple.
We require

$$b(S) \cdot h(S) = C(S),$$

i.e. area equals gate count and

$$h(S)/2 \leq b(S) \leq 2h(S),$$

i.e. layouts of simple circuits are not too slim or too flat. In order to keep drawings simple we will place pins quite liberally at the borders

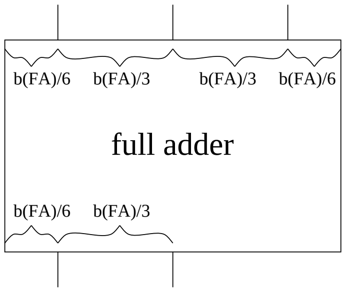

Figure 14: shape of full adder

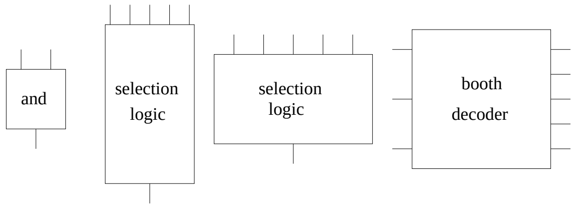

Figure 15: shapes of basic circuits

of the circuits. Input pins are at one side of the rectangle, output pins are at the opposite side.

Nets consist of horizontal and vertical lines (wires). They must have a minimal distance δ from each other and from circuits. Thus a wire channel for t lines has width $(t+1)\delta$. The *size* $|N|$ of a net N is the sum of the length of the lines that constitute the net.

We define the delay of a circuit S driving a net N as

$$D(S,N) = D(S) + \nu \cdot |N|.$$

The parameter ν weights the influence of wire delay on the total delay. If $\nu = 0$ only gate delays count. For a square inverter NOT with $C(S) = 1$ we have $b(NOT) = h(NOT) = 1$. Suppose we connect the output of such a gate with a single wire N of length $h(NOT) = 1$ and we have $\nu \geq 1$. Then

$$\nu \cdot |N| \geq 1 = D(NOT),$$

i.e. a wire which is as long as the gate contributes to the delay as much as the propagation delay of the gate or more. This does not seem reasonable. Therefore, we restrict the range of ν to the interval $[0,1]$.

We do not restrict the fanout of circuits, but within limits the parameter ν can also be used to model fanout restrictions. We will consider only four types of simple circuits S, namely AND gates, full adders FA, Booth decoders BD and the selection logic SL. We will use two geometries for the selection logic. The geometries from Fig. 16 happen to make the layouts of addition trees particularly simple. We place the pins of the full adder as specified in Fig. 14, and we place the pins of the remaining circuits as suggested by Fig. 15.[1] The exact position of the input pins of AND gates and the selection logic contributes only marginally to the run time of addition trees and will not be considered in the analysis.

We will use $\delta = 0.1$.

6 Layouts and their analysis

First we specify and analyze the layout of a plain 4/2-tree T with $M/4$ leaves where partial products are generated with simple AND-gates. The tree T has depth $\mu = \log(M/4)$ as well as μ levels of interior nodes. The number of interior nodes is $M/4 - 1$. This will then be compared with the layout of a tree T' with $M'/4$ leaves where partial products are generated by circuits SL controlled by Booth recoders BD.

All our layouts will consist of a matrix of full adders plus extra circuits and wires. Every node in the tree is either a 3/2 adder and occupies one row of the matrix or it is a 4/2 adder and occupies 2 consecutive rows of the matrix. Every bit position occupies a column of the matrix. Between neighboring full adders

[1]Strictly speaking we use three layouts for the selection logic. Two of the layouts only differ in the position of the output pin.

S	C(S)	b(S)	h(S)
SL	10	$\sqrt{5}$ $2\sqrt{5}$	$2\sqrt{5}$ $\sqrt{5}$
AND	2	$\sqrt{2}$	$\sqrt{2}$
FA	14	$(2\sqrt{5}+2\delta)$	$14/b(FA)$
BD	11	$\sqrt{11}$	$\sqrt{11}$

Figure 16: Geometries of basic circuits

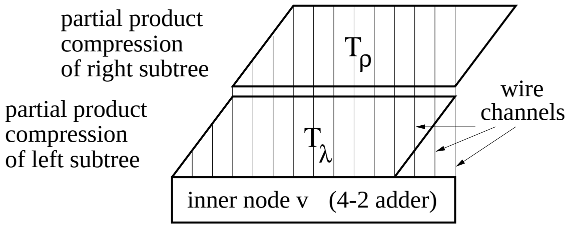

Figure 17: Recursive Partition of the tree layout.

of the same row we leave a wire channel of appropriate width.

Inputs $a[i]$ are fed into the layout at the top with indices increasing from right to left. Inputs $b[j]$ are fed into the layout from the left with indices increasing from top to bottom. Outputs are produced at the right border and at the bottom of the layout.

Let T_λ resp. T_ρ be the subtree rooted in the left resp. right son of a node v. Then, we layout T_ρ on top of T_λ. This is followed by a row for v (see Fig. 17). Between the layouts of the two subtrees we leave space for one wire. This space will be filled with boxes4/2a specified below.

It only remains to specify where to place the extra circuits and how to layout the wires. For tree T it suffices to specify the three types of *boxes* in Fig. 20a to Fig. 20c:

- box3: this is one full adder which is part of a leaf v of the tree with weight $L = 3$. The whole 3/2 adder v has as inputs 3 bits of b which are routed in the *b-channel* $b[2:0]$. Every bit $a[i]$ is needed in three consecutive bit positions of v; then it is routed to the next leaf down the tree and one position to the left. Thus, 3 lines of an *a-channel* $a[2:0]$ suffice to accommodate the a-inputs for v. For all leaves v and all i we feed input $a[i]$ into a-channel $a[i \bmod 3]$. Before $a[i]$ can be fed into the channel, bit $a[i-3]$ has to be removed and routet down to the next leaf of the tree. This -unfortunately- consumes horizontally distance 2δ.

- box 4: this is one pair of full adders which is part of a leaf v of the tree with weight $L = 4$. The whole 4/2 adder v has as inputs 4 bits of b which are routed in the b-channel $b[2:0]$ of the first 3/2-adder and in one b-channel of the second 3/2-adder. Similarly 4 a-channels are used. Each signal $a[i]$ is fed into 3 full adders of consecutive bit positions in the first 3/2-adder level and then into one full adder in the next bit position of the second 3/2-adder. After that $a[i]$ is routed down to the next leaf of the tree.

- box 4/2: This is strictly speaking a pair of boxes which is part of an interior node v of the tree with subtrees T_ρ and T_λ. Box4/2a is inserted between the layout of T_ρ and T_λ. Box4/2b is added at the bottom of the layout of T_λ. The box consumes two wires in each vertical wire channel, one for a carry output and one for the sum output of one full adder of the root of T_ρ. We place the sum lines in the left and the carry lines in the right half the wire channel. It will not matter in the analysis where we place them exactly.

All boxes have the same width $b(box)$. Because each level of interior nodes contributes 2 lines to the global wire channel we find

$$b(box) = b(FA) + (\mu + 3) \cdot \delta$$

Box4/2a is placed on top of the b-channel of the rightmost leaf of T_λ. Thus it contributes only δ to the height, and we have

$$\begin{aligned} h(box4/2) &= 2h(FA) + 7\delta \\ h(box3) &= h(FA) + h(AND) + 7\delta \\ h(box4) &= 2h(FA) + 2h(AND) + 11\delta \end{aligned}$$

The height of the whole layout is

$$\begin{aligned} h(T) &= (M/4-1)\cdot h(box42) + (M/4)\cdot h(box3) \\ &\quad + a\cdot(h(box4) - h(box3)). \end{aligned}$$

For the size of the nets $a[i]$ we consider

- the horizontal extension $m \cdot b(box)$.
- the vertical extension. This extends from the very top of the layout to the leftmost leaf. Below the leftmost leaf we have μ many boxes4/2b.
- the m connections from the a-channels to the AND-gates, each of length up to 3δ.

Thus for the largest ones of the nets $a[i]$ we have

$$\begin{aligned} |a[i]| &= m\cdot b(box) + h(T) - \mu\cdot(h(box4/2-\delta) \\ &\quad -h(FA) - h(AND) + 3m\delta. \end{aligned}$$

For $m \leq n$ the size of the nets $b[i]$ is dominated by the size of the nets $a[i]$.
The carry in box4 travels horizontally over a full box minus 2/3 of a full adder. Thus we have

$$|carry4| = b(box) - 2b(FA)/3 + h(AND) + 4\delta.$$

Next we determine the accumulated delay from a leaf to the root which follows in each box4/2 the following path: sum bit in T_ρ – 1'st full adder in v – carry output – 2'nd full adder in the box to the left – sum bit of that adder.

We estimate the accumulated length of the wires separately

- all wires connected to carry outputs:

$$Carry_{4/2} = \mu\cdot(b(box) - 2b(FA)/3 + 3\delta).$$

- all wires connected to sum outputs *not counting* vertical displacement in the global wire channels. In the wire channels every one of the distances $2k\delta$ occurs exactly once.

$$\begin{aligned} Sum_{4/2} &= \mu\cdot(4b(FA)/3 + 4\delta) \\ &\quad + \textstyle\sum_{k=1}^{\mu} 2k\delta \\ &= \mu\cdot(4b(FA)/3 + 4\delta) \\ &\quad + \delta\cdot(\mu^2+\mu) \end{aligned}$$

- total vertical displacement of the wires connected to sum outputs in the wire channels; from the bottom of rightmost leaf (a box4 if $m = 53$) to the very bottom of the layout not counting the boxes4/2 on the path:

$$\begin{aligned} Channel &= h(T) - h(box4) \\ &\quad - (\log(M/4))\cdot h(box4/2). \end{aligned}$$

The competing paths from the carry output of T_ρ to the second full adder are faster than the path considered above for realistic values of ν.

We now imagine that all a-inputs and b-inputs are the outputs of drivers whose propagation delay we ignore. Then the total delay of the tree equals

$$\begin{aligned} Time(T) &= \nu\cdot(|a[i]| + |carry4| + Carry_{4/2} \\ &\quad +Sum_{4/2} + Channel) \\ &\quad +D_{AND} + 2D_{FA}\cdot(1+\mu). \end{aligned}$$

Exactly along the same lines one specifies and analyzes the layout of the addition tree T' with booth recoding. The tree has $M'/4$ leaves and depth $\mu' = log(M'/4)$.

One uses the boxes specified in Fig. 21a to Fig. 21c. The selection logic becomes more complex and has to be placed in two rows in order not to exceed the Full adder width. For this organization in Box4' an additional horizontal input wire must be routed around the full adder. Also the channel width changes a bit and becomes $\mu' \cdot \delta$.

$$b(box') = b(FA) + \mu' \cdot \delta + 4\delta.$$

This change of width is the only change for the Box4/2'; its height stays the same. In the other two boxes there must be 5 input wires per selection logic. In Box3' the top selection logic is rotated in order not to waste area. From the figures one obtains the following equations:

$$\begin{aligned} h(box4/2') &= 2h(FA) + 7\delta \\ h(box3') &= h(FA) + h(SL) + b(SL) + 17\delta \\ h(box4') &= 2h(FA) + 2h(SL) + 26\delta \end{aligned}$$

Most of the other equations only change slightly:

$$\begin{aligned} h(T') &= (M'/4 - 1) \cdot h(box4/2') \\ &\quad + (M'/4) \cdot h(box3') \\ &\quad + a' \cdot (h(box4') - h(box3')) \\ |a'[i]| &= m' \cdot b(box') + h(T') \\ &\quad - \mu' \cdot (h(box4/2' - \delta) \\ &\quad - h(FA) - h(SL) + 4m'\delta \\ |carry4'| &= b(box') - 2b(FA)/3 + 3\delta \\ Carry'_{4/2} &= \mu' \cdot (b(box') - 2b(FA)/3 + 3\delta). \\ Sum'_{4/2} &= \mu' \cdot (4b(FA)/3 + 4\delta) \\ &\quad + \sum_{k=1}^{\mu'} 2k\delta \\ &= \mu' \cdot (4b(FA)/3 + 4\delta) \\ &\quad + \delta \cdot (\mu'^2 + \mu') \\ Channel' &= h(T') - h(box4') \\ &\quad - (\log(M'/4)) \cdot h(box4/2'). \end{aligned}$$

Additionally to these wires we must consider the wire between one selection logic output in the upper row and the fulladder input in Box4':

$$|sum4'| = h(SL) + b(SL)/2 + 12\delta.$$

Also the nets $b'[i]$ can become important. Their delay in sequence to the booth decoder delay has to compete against the nets $a'[i]$ and it depends on the constant ν which one is slower.

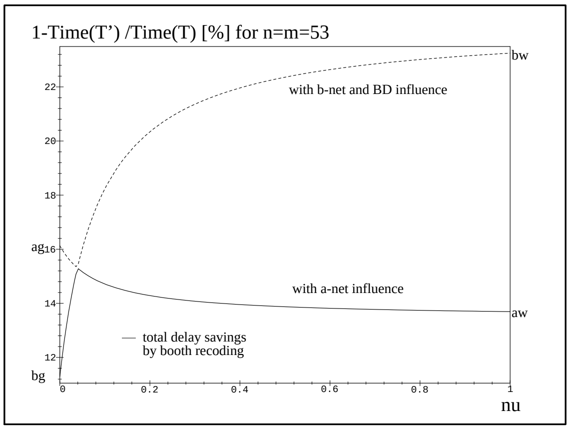

Figure 18: Relative delay improvement by Booth recoding for $n = m = 53$

Therefore, we have to consider the length of the longest $|b'[i]|$. It has to reach n' bit positions and for each connection in the worst case it crosses all the 9 other selection logic wires.

$$|b'[i]| = n' \cdot (b(box') + 10\delta).$$

With this we can evaluate the delay of the tree T':

$$\begin{aligned} Time(T') &= \nu' \cdot (max(|a'[i]|, |b'(i)| + \frac{D_{BD}}{\nu'}) \\ &\quad + |sum4'| + |carry4'| + Carry'_{4/2} \\ &\quad + Sum'_{4/2} + Channel') \\ &\quad + D_{SL} + 2D_{FA} \cdot (1 + \mu'). \end{aligned}$$

In Fig. 18 the relative improvement

$$(TIME(T) - TIME(T'))/TIME(T)$$

is plotted for $n = m = 53$ as a function of ν.

This figure is surprisingly complex and counter intuitive. In particular we see gains due to Booth recoding *decrease* with increasing ν, i.e. with slower wires. For small values of ν until $\nu \approx .1$ the delay of the Booth decoder $D_{BD} = 3$ plus the delay of the b-nets is larger than the delay of the a-nets. For small ν we have

$$TIME(T') \approx 55 + 537\nu$$

whereas for large ν we have

$$TIME(T') \approx 52 + 614\nu.$$

We always have

$$TIME(T) = 62 + 710\nu.$$

This explains why the graph has two branches. For $\nu = 0$ only gate delays count and we have

$$TIME(T')/TIME(T) \approx 55/62 \approx 89\%.$$

This explains why the branch for the b-nets starts around $100\% - 89\% = 11\%$. It remains to explain why the branch for the a-nets is falling with ν. For $\nu = 0$ the branch starts at $1 - 52/62 \approx 100\% - 84\% = 16\%$. For large ν the savings are dominated by wire delays and approach $1 - (52 + 614)/(62 + 710) \approx 14\% < 16\%$. Thus the branch falls because the wire delays have not fallen much due to Booth decoding. The reason for this is the geometry of our particular layouts which are quite wide and not very high:

Denote by $wire(T)$ resp. $wire(T')$ the wire delays for T and T'. Then we have for $\nu = 1$:

$$\begin{aligned} wire(T) &\approx w(T)/2 + 2h(T) \\ wire(T') &\approx w(T')/2 + 2h(T') \end{aligned}$$

because a-nets travel horizontally over half the layout and vertically over almost the full layout; the longest paths from the leaves to the root travel vertically over almost the full layout whereas horizontal displacement is logarithmic. We have

$$w(T) \approx w(T') \approx 2n \cdot w(box) \approx 2n \cdot 5.7$$

If we would use as leaves only 4/2–adders, then the tree T would have $n/4$ leaves each of height $h(box4/2) \approx 9.9$ and around $n/4$ interior nodes, each of height $h(box4/2) \approx 6.7$. Thus $h(T) \approx (n/4) \cdot (9.9 + 6.7) = 4.15n$. Similarly we have $h(T') \approx (n/8) \cdot (h(box'4) + h(box4/2) \approx (n/8) \cdot (17.5 + 6.7) = 3.025n$. For $n = 53$ we get $wire(T') \approx 623$ and $wire(T) \approx 742$.

Note that we have $h(T)/w(T) \approx 1/3$ and $h(T)/w(T') \approx 1/4$. If we make the layouts more square [2], say by doubling h and halving w, then the savings due to Booth recoding increase *but the layouts become slower !!*. This incidentally shows, that the common practice of making the layouts of addition trees roughly square is not always a good idea.

Asymptotic considerations If $n = m$ and n tends to infinity the whole layout consists almost exclusively of channels. The height is $\Theta(n)$ which is negligible against the width $\delta \cdot \nu \cdot n \log n + o(n \log n)$. The run time is dominated by the term $2\nu\delta \cdot n \log n$ due to the delay of the horizontal portions of nets a_i. The same holds for the tree layout without Booth recoding. Thus even in the VLSI model Booth recoding – when applied to 4/2-trees – saves asymptotically no constant factor of the delay.

However, if n grows large the layouts of plain multiplication arrays with width and height $\Theta(n)$ become eventually faster than 4/2-trees. Let A resp. A' be the straight forward layout of a multiplication array without resp. with Booth recoding. With the techniques introduced above one easily specifies and analyzes these layouts.
Then

$$\begin{aligned} Time(A) &= m(6 + \nu \cdot 13.5) + o(m) \\ Time(A') &= m(3 + \nu \cdot 11.3) + o(m). \end{aligned}$$

We see that $Time(A')/Time(A)$ tends to a constant fraction, if $n = m$ and n tends to infinity. Figure 6 depicts the relative asymptotic delay savings by booth decoding depending on the technology parameter ν. Thus asymptotically Booth recoding *does improve the delay* by 27% to 50%. This is, however, of academic interest, because even for very slow wires

[2] This is common practice

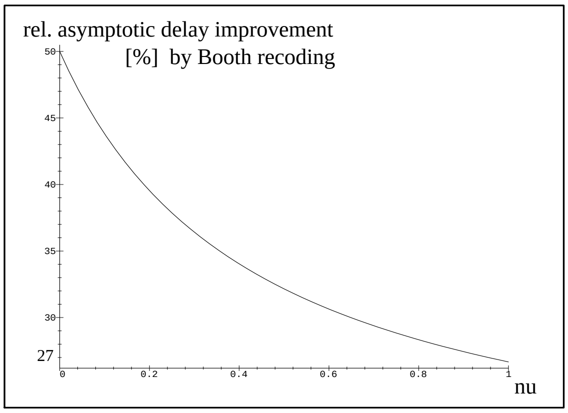

Figure 19: Relative asymptotic delay improvement by Booth recoding.

($\nu = 1$) we have $TIME(A') < TIME(T')$ only for $n \geq 10^6$.

We denote by $AREA(V)$ the area occupied by VLSI–layout V. From the results on circuit complexity one infers

$$AREA(A) = 23.9 \cdot m^2 + o(m^2)$$
$$AREA(A') = 15.5 \cdot m^2 + o(m^2).$$

Asymptotically the area savings by Booth recoding tend to $1 - AREA(A')/AREA(A) \approx 35.1\%$. For $n = 53$ Booth recoding saves $1 - AREA(T')/AREA(T) \approx 15.1\%$ of the area using the tree layout.

7 Conclusions

We have investigated formal versions of three folklore theorems about fixed point multiplication namely:

1. Balanced addition trees in n-bit Multipliers have $n^2 + O(n \log n)$ full adders. This required the combinatorial argument from section 3.

2. Booth recoding saves between 15% and 35% of the cost of partial product generation and addition tree.

3. Booth recoding saves between 11% and 50% of the delay of partial product generation and addition tree.

Moreover we have demonstrated the surprising effect, that the savings of Booth decoding might decrease as wires become slower. Empirical studies so far have suggested, that Booth recoding is particulary helpful if wires are slow [4].

Technically the main contribution of the paper is the VLSI model from section 5 and the techniques of analysis of the same section. The detailed yet tractable nature of this model opens the way for many further investigations. We list just a few

- One can systematically study where drivers should be placed in order to make nets smaller and hence speed up signal propagation.

- One can analyze hybrid layouts (arrays of small trees) and the layouts of many other multiplication designs ([1][5] [8][14][17][20] [22][23][25][30][32]).

- The layouts of various adder and shifter designs can be analyzed in a quite realistic way.

- It is common practice to 'fold' layouts of addition trees into more square layouts. But the formula

$$wire(T) \approx w(T)/2 + 2h(T)$$

suggests, that trivial folding of layouts produces slower designs. This clearly needs closer investigation.

8 Acknowledgments

For helpful and inspiring discussions the authors thank Michael Bosch and Guy Even.

References

[1] H. Al-Twaijry. *Area and Performance Optimized CMOS Multipliers.* PhD thesis, Stanford University, August 1997. accessible via ftp://umunhum.stanford.edu /tr/hesham.aug97.thesis.ps.

[2] H. Al-Twaijry and M. Flynn. Multipliers and datapaths. Technical Report CSL-TR-94-654, Stanford University, December 1994.

[3] H. Al-Twaijry and M. Flynn. Performance/area tradeoffs in booth multipliers. Technical Report CSL-TR-95-684, Stanford University, November 1995.

[4] H. Al-Twaijry and M. Flynn. Technology scaling effects on multipliers. Technical Report CSL-TR-96-698, Stanford University, July 1996.

[5] G.W. Bewick. *Fast Multiplication: Algorithms and Implementation.* PhD thesis, Stanford University, March 1994. accessible via ftp://umunhum.stanford.edu /tr/bewick.apr94.thesis.ps.Z.

[6] A.D. Booth. A signed binary multiplication technique. *Quart. Journ. Mech. and Applied Math.*, 4(2):236–240, 1951.

[7] L. Dadda. Some schemes for parallel multipliers. *Alta Frequenza*, 34:349–356, 1965.

[8] B.C. Drerup and E.E. Swartzlander. Fast multiplier bit-product matrix reduction using bit-ordering and parity generation. *Journal of VLSI Signal Processing*, 7:249–257, 1994.

[9] L.A. Glasser and D.W. Dobberpuhl. *The Design And Analysis Of VLSI Circuits.* Addison-Wesley Publishing Company, 1985.

[10] IEEE standard for binary floating-point arithmetic. ANSI/IEEE754-1985, New York, 1985.

[11] J. Keller and W.J. Paul. *Hardware Design.* Teubner Verlagsgesellschaft, Stuttgart, Leipzig, 2nd edition, 1997.

[12] Israel Koren. *Computer arithmetic algorithms.* Prentice-Hall International, 1993.

[13] P. Kornerup. private communication.

[14] L. Kuehnel and H. Schmeck. A closer look at VLSI multiplication. *INTEGRATION, the VLSI journal*, 6:345–359, 1988.

[15] P.E. Madrid, B. Millar, and E. E. Swartzlander. Modified booth algorithm for high radix multiplication. *IEEE Computer Design Conference*, pages 118–121, 1992.

[16] C. Mead and L. Conway. *Introduction To VLSI Systems.* Addison-Wesley Publishing Company, 1980.

[17] Z.-J. Mou and F. Jutand. Overturned-stairs adder trees and multiplier design. *IEEE Transactions on Computers*, 41(8):940–948, August 1992.

[18] S.M. Mueller and W.J. Paul. *The Complexity of Simple Computer Architectures.* Lecture Notes in Computer Science 995. Springer, 1995.

[19] C. Nakata and J. Brock. *H4C Series: Design Reference Guide. CAD, 0.7 Micron* L_{eff}. Motorola Ltd., 1993. Preliminary.

[20] V.G. Oklobdzija, D. Villeger, and S. S. Liu. A method for speed optimized partial product reduction and generation of fast parallel multipliers using an algorithmic approach. *IEEE Transactions on Computers*, 45(3):294–306, March 1996.

[21] A.R. Omondi. *Computer Arithmetic Systems; Algorithms, Architecture and Implementations*. Series in Computer Science. Prentice-Hall International, 1994.

[22] R.M. Owens, R.S. Bajwa, and M.J. Irwin. Reducing the number of counters needed for integer multiplication. *Proceedings 12th Symposium on Computer Arithmetic*, 12:38–41, 1995.

[23] K.F. Pang, R. Soong, H.-W.and Sexton, and P.H. Ang. Generation of high speed CMOS multiplier-accumulators. *Proceedings IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 217–220, 1988.

[24] L.P. Rubinfeld. A proof of the modified booth's algorithm for multiplication. *IEEE Transactions on Computers*, pages 1014–1015, October 1975.

[25] N. Takagi, H. Yasuura, and S. Yajima. High-speed VLSI multiplication algorithm with a redundant binary addition tree. *IEEE Transactions on Computers*, C-34(9):217–220, September 1985.

[26] C.D. Thompson. Area-time complexity for VLSI. *Proc. Eleventh Annual ACM Symposium on the Theory of Computing*, 11:81–88, 1979.

[27] J.D. Ullman. *Computational Aspects of VLSI*. Computer Science Press, 1984.

[28] J. Vuillemin. A very fast multiplication algorithm for VLSI implementation. *INTEGRATION, the VLSI journal*, 1:39–52, 1983.

[29] C.S. Wallace. A suggestion for parallel multipliers. *IEEE Trans. Electron. Comput.*, EC-13:14–17, 1964.

[30] Z. Wang, A. Jullien, and C. Miller. A new design technique for column compression multipliers. *IEEE Transaction on Computers*, 44(8):962–970, August 1995.

[31] I. Wegener. *The Complexity of Boolean Functions*. John Wiley & Sons, 1987.

[32] R.K. Yu and G.B. Zyner. 167 MHz Radix-4 floating point multiplier. *Proceedings 12th Symposium on Computer Arithmetic*, 12:149–154, 1995.

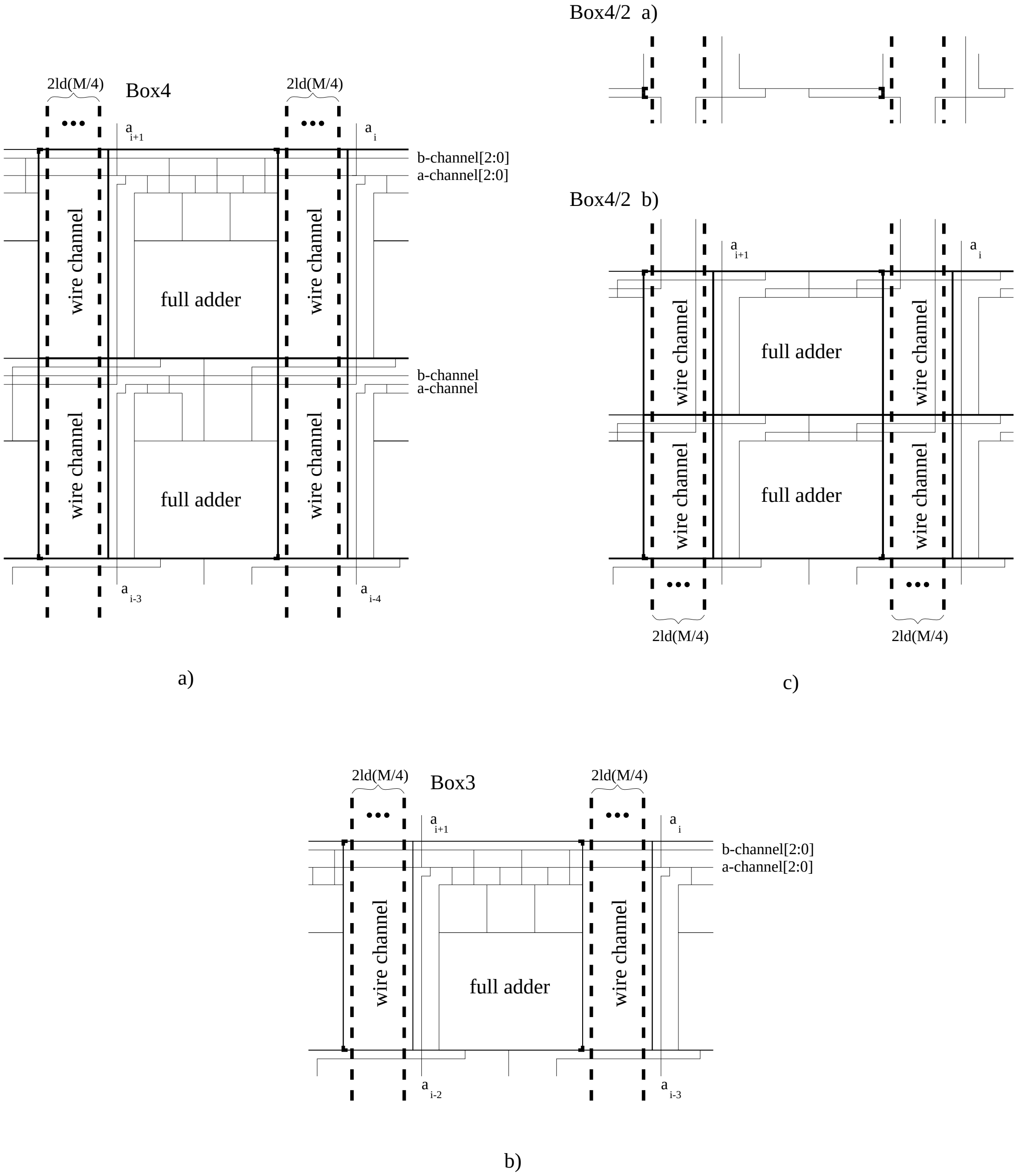

Figure 20: Basic boxes for non-booth multiplier construction

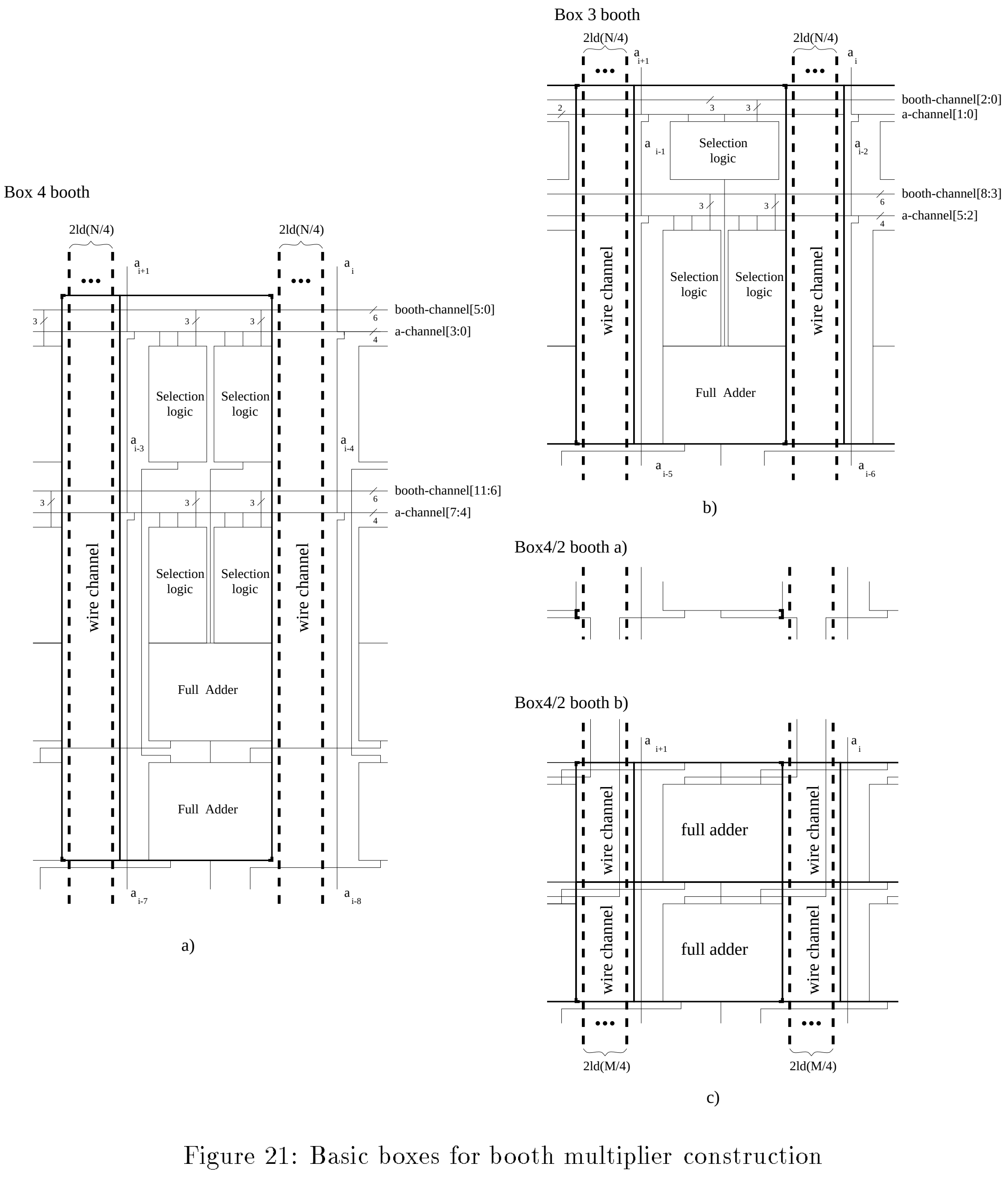

Figure 21: Basic boxes for booth multiplier construction