ELSEVIER

INTEGRATION, the VLSI journal 32 (2002) 5–40

INTEGRATION
theVLSI journal

www.elsevier.com/locate/vlsi

To Booth or not to Booth

Wolfgang J. Paul[a,*], Peter-Michael Seidel[b]

[a] *Computer Science Department, University of the Saarland, 66041 Saarbruecken, Germany*
[b] *Computer Science and Engineering Department, Southern Methodist University, Dallas TX 75275, USA*

Abstract

Booth Recoding is a commonly used technique to recode one of the operands in binary multiplication. In this way the implementation of a multipliers' adder tree can be improved in both cost and delay. The improvement due to Booth Recoding is said to be due to improvements in the layout of the adder tree especially regarding the lengths of wire connections and thus cannot be analyzed with a simple gate model. Although conventional VLSI models consider wires in layouts, they usually neglect wires when modeling the delay. To make the layout improvements due to Booth recoding tractable in a technology-independent way, we introduce a VLSI model that also considers wire delays and constant factors. Based on this model we consider the layouts of binary multipliers in a parametric analysis providing answers to the question whether to use Booth Recoding or not.

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 $\frac{4}{2}$-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 [1–8]. Booth recoding [9–13] 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 $\frac{1}{4}$. In [9,10,14–16], however, the usefulness of Booth recoding is challenged

*Corresponding author.
E-mail addresses: wjp@cs.uni-sb.de (W.J. Paul), seidel@seas.smu.edu (P.-M. Seidel).

0167-9260/02/$ - see front matter
PII: S0167-9260(02)00040-8

altogether. Finally, in [15] an ambitious case study of around 1000 concrete designs 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 [17] and VLSI complexity [18,19] are only trusted to be exact up to constant factors.
- The standard theoretical VLSI model ignores wire delays.
- Conway/Mead style rules [20,21] *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 [2,6,7] and even $\frac{4}{2}$-trees [3,4,22,23] 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 v 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 $\frac{4}{2}$-trees. We show that Booth encoding improves for *all* reasonable values of v both the delay and the area of the resulting VLSI layouts by constant factors between 26% and 50% minus low-order terms which depend on v and the length n of the operands. For practical operand length in the range $8 \leqslant n \leqslant 64$ we specify the gain exactly.

The paper is organized in the following way. In Section 2, we review Booth recoding as explained in [10,24]. Section 3 provides a combinatorial lemma which will subsequently permit to count the number of full adders in certain $\frac{4}{2}$-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 [25] is combined with a linear delay model for nets of wires. Section 6 contains the specification and analysis of layouts. The layouts for $\frac{4}{2}$-trees follow partly a suggestion from [26]. In Section 7, the savings due to Booth recoding are evaluated. We analyze both the asymptotical behavior and the situation for practical n. We conclude in Section 8 and list some further work.

2. Preliminaries

For $a = a[n-1:0] = (a[n-1], \ldots, a[0]) \in \{0,1\}^n$ we denote by

$$\langle a \rangle = \sum_{i=0}^{n-1} a[i] 2^i$$

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

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

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 $\langle a\rangle\langle b\rangle = \langle p\rangle$ holds.

2.1. Multiplication

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

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

Then, we have

$$S_{j,1} = \langle a\rangle b[j]2^j,$$

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

and

$$\begin{aligned} \langle a\rangle\langle b\rangle &= S_{0,m} \\ &= S_{0,m-1} + S_{m-1,1}. \end{aligned}$$

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 recoding or Booth recoding radix 4) 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

$$\langle b\rangle = 2\langle b\rangle - \langle b\rangle = \sum_{j=0}^{m'-1} B_{2j}4^j,$$

where

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

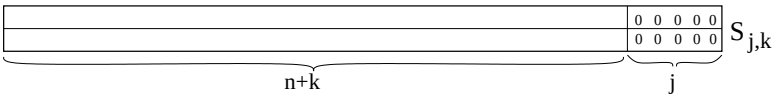


Fig. 1. $S_{j,k}$ in carry–save representation.

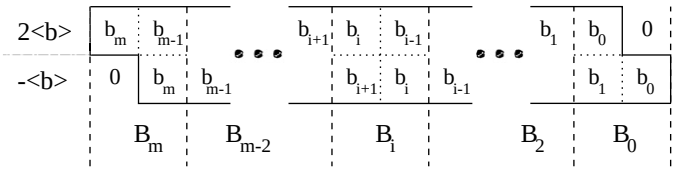


Fig. 2. Booth digits B_{2j}.

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

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

With

$$C_{2j} = \langle a \rangle B_{2j} \in \{-2^{n+1} + 2, \ldots, 2^{n+1} - 2\},$$

$$D_{2j} = \langle a \rangle |B_{2j}| \in \{0, \ldots, 2^{n+1} - 2\},$$

$$d_{2j} = bin_{n+1}(D_{2j}),$$

the product can be computed from the sums

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

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

$$E_{2j} = C_{2j} + 3\,2^{n+1}, \quad e_{2j} = bin_{n+3}(E_{2j}),$$

$$E_0 = C_0 + 4\,2^{n+1}, \quad e_0 = bin_{n+4}(E_0).$$

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

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

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

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

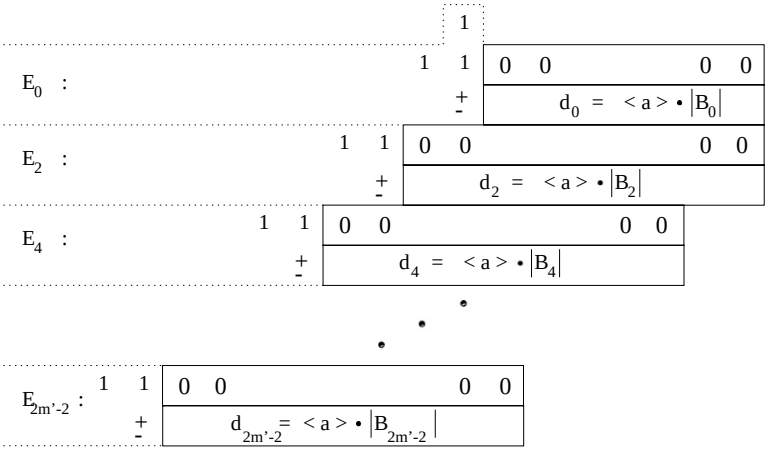


Fig. 3. Summation of the E_{2j}.

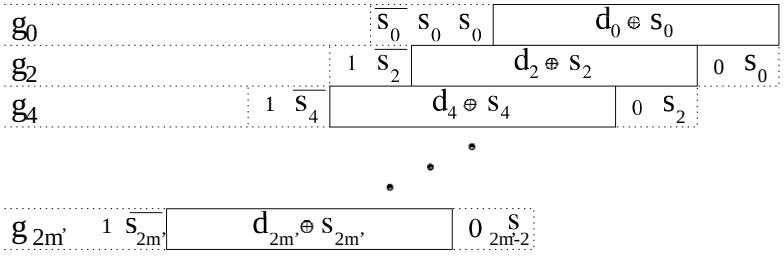


Fig. 4. Partial product construction.

With respect to the standard subtraction algorithm for binary numbers, the e_{2j} can be computed by

$$\langle e_{2j} \rangle = \langle 1\overline{s_{2j}}, d_{2j} \oplus s_{2j} \rangle + s_{2j},$$

$$\langle e_0 \rangle = \langle \overline{s_0} s_0 s_0, d_0 \oplus s_0 \rangle + s_0.$$

Based on these equations, the computation of the numbers

$$F_{2j} = E_{2j} - s_{2j},$$

$$f_{2j} = bin_{n+3}(F_{2j}),$$

$$f_0 = bin_{n+4}(F_0)$$

is easy, namely

$$f_{2j} = (1\overline{s_{2j}}, d_{2j} \oplus s_{2j}),$$

$$f_0 = (\overline{s_0} s_0 s_0, d_0 \oplus s_0).$$

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 nonnegative. Formally, let

$$g_{2j} = (f_{2j}, 0s_{2j-2}) \in \{0,1\}^{n+5},$$

$$g_0 = (f_{2j}, 00) \in \{0,1\}^{n+6}.$$

Then

$$\langle g_{2j} \rangle = 4\langle f_{2j} \rangle + s_{2j-2}$$

and hence we get the product by

$$\langle a \rangle \langle b \rangle = \sum_{j=0}^{m'-1} E_{2j} 4^j = \sum_{j=0}^{m'-1} \langle g_{2j} \rangle 4^{j-1}$$

with the g_{2j} as partial products. We define

$$S'_{2j,2k} = \sum_{t=j}^{j+k-1} \langle g_{2t} \rangle 4^{t-1}.$$

Then, we have analogously to the nonBooth case

$$S'_{2j,2} = \langle g_{2j} \rangle 4^{j-1},$$

$$S'_{2j,2(k+h)} = S'_{2j,2k} + S'_{2(j+k),2h}.$$

One can easily show, that $S'_{2j,2k}$ is a multiple of 2^{2j-2} and $S'_{2j,2k} < 2^{n+2j+2k+2}$. Therefore, at most $n + 2k + 4$ nonzero positions are necessary to represent $S'_{2j,2k}$ in both carry-save or binary form.

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 non-decreasing 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 1. $H \leqslant (\mu m)/2$.

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

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

and the lemma follows. □

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 $\{k2^\ell, (k+1)2^\ell\}$. 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

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

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

4. Gates

In this section, we use a simple gate model [25] to analyze delay and cost of the multiplication circuits considering only the influence of gates. This model does not consider fanout restrictions or wire effects. 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

Gate	*Not*	*Nand* *Nor*	*And* *Or*	*Xor* *Xnor*
delay	1	1	2	2
cost	1	2	2	4

Fig. 5. Motorola technology parameters.

basic gate delays and costs can be extracted from any design system. We will consider the Motorola technology [27] (Fig. 5), but the reader might choose his favorite technology in our parametrical analysis.

4.1. Partial product generation

4.1.1. NonBooth

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

$$\langle a\rangle b[j] = \langle a[n-1]\wedge b[j], \ldots, a[0]\wedge b[j]\rangle.$$

The nm *AND*-gates have a cost of $2nm$. 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

$$g_{2j} = (1\overline{s_{2j}}, d_{2j}\oplus s_{2j}, 0s_{2j-2}),$$

$$g_0 = (\overline{s_0}s_0s_0, d_0\oplus s_0, 00),$$

shifted by $2j-2$ bit positions.

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

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

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

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

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

$b[2j+1:2j-1]$	B_{2j}	$b[2j+1:2j-1]$	B_{2j}
000	0	100	-2
001	1	101	-1
010	1	110	-1
011	2	111	-0

(a)

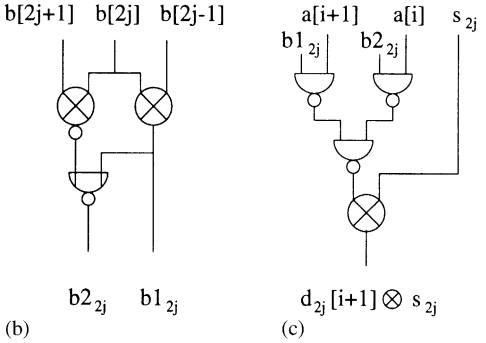


Fig. 6. (a) Booth digit signals, (b) Booth decoder and (c) selection logic.

The selection logic SL from Fig. 6(c) directs either $a[i]$ or $a[i+1]$ or 0 to position $[i+1]$ of d_{2j} and the inversion depending on s_{2j} yields $g_{2j}[i+3]$. 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 $m'C_{BD} = 11m'$. The selection logic occurs for each partial product $(n+1)$-times. Therefore, the selection logics have altogether a cost of

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

4.2. Redundant partial product addition

We study two standard constructions for the reduction of the partial products to a carry-save representation of the product: plain multiplication arrays and $\frac{4}{2}$-trees. Each construction uses *k-bit carry-save adders* (k-CSA, $\frac{3}{2}$-adders) [3–5]. They are composed of k full adders working in parallel and hence they have cost kC_{FA} and delay D_{FA}. Cascading two k-CSAs one constructs a *k-bit* $\frac{4}{2}$*-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

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

holds. The k-bit $\frac{4}{2}$-adders constructed in this way have cost $2kC_{FA}$ and delay $2D_{FA}$. We consider full adder implementations with cost $C_{FA} = 14$ and delay $D_{FA} = 6$. There are optimized implementations of $\frac{4}{2}$-adders like described in [23], we do not use them in this analysis to keep the presentation simple.

In order to introduce techniques of analysis we review quite formally the construction of standard multiplication arrays [3,21]. A carry-save representation of $S_{0,3}$ can be computed by a single n-CSA (see Fig. 7(a)). 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. 7(c)). The $m-2$ many n-CSAs which are cascaded this way have cost $n(m-2)C_{FA}$ and delay $(m-2)D_{FA}$. The combined cost and delay for (nonBooth) partial product generation and the multiplication array are

$$\begin{aligned} C_{Array} &= n(m-2)C_{FA} + nmC_{AND} \\ &= 16nm - 28n, \end{aligned}$$

$$\begin{aligned} D_{Array} &= (m-2)D_{FA} + D_{AND} \\ &= 6m - 10. \end{aligned}$$

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')-CSAs suffice to sum the partial

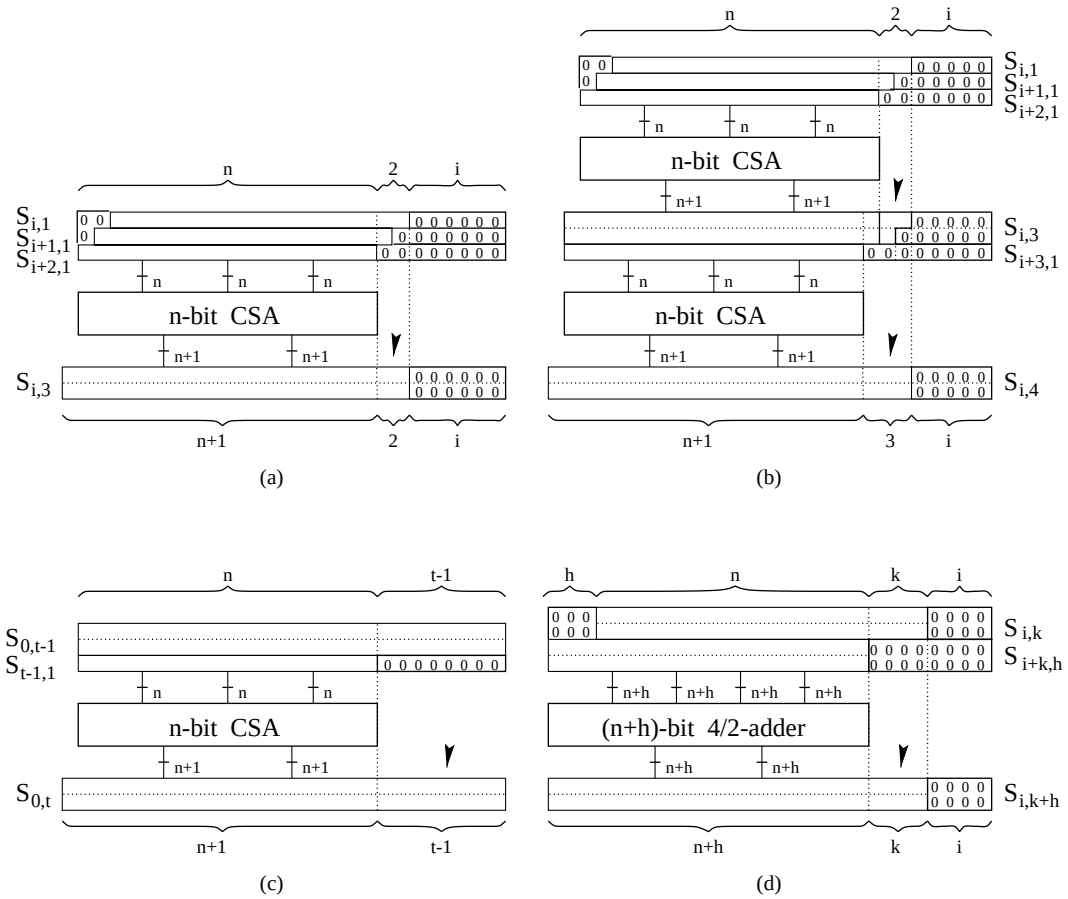


Fig. 7. Partial compression of (a) $S_{i,3}$; (b) $S_{i,4}$; (c) $S_{0,t}$ from a carry–save representation of $S_{0,t-1}$ and $S_{t-1,1}$; and (d) $S_{i,k+h}$ from carry–save representations of $S_{i,k}$ and $S_{i+k,h}$.

products in a multiplication array with Booth-2 recoding. Taking into account the partial product generation one obtains cost and delay

$$C'_{Array} = (n')(m'-2)C_{FA} + (n+1)m'C_{SL} + m'C_{BD}$$
$$= 24nm' + 91m' - 28n - 140,$$

$$D'_{Array} = (m'-2)D_{FA} + D_{SL} + D_{BD}$$
$$= 6m' - 5.$$

For $n = m = 53$ (double precision) we get

$$C'_{Array}/C_{Array} = 35457/43460 = 81.6\%,$$

$$D'_{Array}/D_{Array} = 157/308 = 50.9\%.$$

For $n = m = 24$ (single precision) we get

$$C'_{Array}/C_{Array} = 8139/8544 = 95.2\%$$

$$D'_{Array}/D_{Array} = 73/134 = 54.5\%.$$

Asymptotically, C'_{Array}/C_{Array} tends to $\frac{12}{16} = 75\%$ and D'_{Array}/D_{Array} tends to $\frac{3}{6} = 50\%$.

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 $\frac{4}{2}$-trees

The situation changes in two respects when we consider addition trees with logarithmic delay like Wallace trees [2,6,7] or $\frac{4}{2}$-trees [1,3,4,22,23]. 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. NonBooth

We construct a specific family of $\frac{4}{2}$-trees which happens to be accessible to analysis. The nodes of the tree are either $\frac{3}{2}$-adders or $\frac{4}{2}$-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 $\mu = \log(M/4) = \lceil \log(m) \rceil - 2$. We will construct the entire addition tree T by two parts as shown in Fig. 8.

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

If $M/2 < m < 3M/4$ we use in the top level only $b = m - M/2$ many $\frac{3}{2}$-adders (and we feed $m - 3b$ representations of partial products directly into the lower part). We arrange the $\frac{3}{2}$-adders at the *right* end of the tree. For $m = 24$ we have $m \geqslant 24$, and use 8 $\frac{3}{2}$-adders as leaves. For $m = 53$ we have $m \geqslant 48$, and we use 11 $\frac{3}{2}$-adders and 5 $\frac{4}{2}$-adders as leaves. We only analyze the first case explicitly.

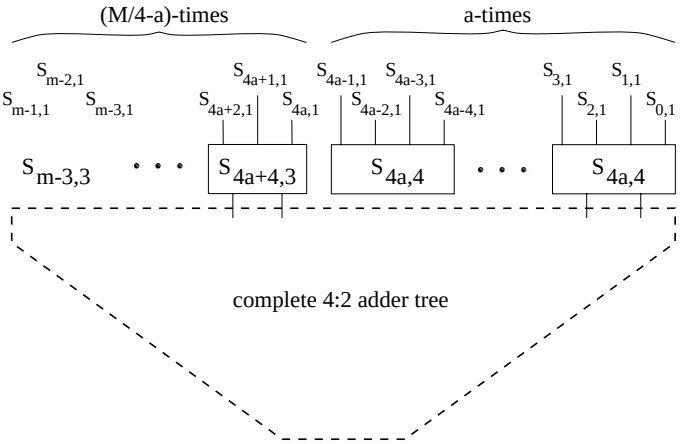


Fig. 8. Adder tree construction.

If all $\frac{3}{2}$- and $\frac{4}{2}$-5adders in the tree would consist of exactly n resp. $2n$ full adders, then the tree would have a total of

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

full adders, because every $\frac{3}{2}$-adder reduces the number of partial products by 1 and every $\frac{4}{2}$-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 $\frac{3}{2}$-adder computes a sum $S_{i,3}$. Thus, n full adders suffice (see Fig. 7(a)). A leaf of the tree which is a $\frac{4}{2}$-adder computes a sum $S_{i,4}$. It can be simplified as shown in Fig. 7(b) such that $2n$ full adders suffice. Thus, in the top level of the tree there are no excess full adders.

Each $\frac{4}{2}$-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. 7(d). 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 $\frac{3}{2}$-adder and with $W(u) = 4$ if it is a $\frac{4}{2}$-adder, then $h = L(u)$ and for the number $E = 2H$ of excess full adders in the tree we have

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

This implies, that the $\frac{4}{2}$-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

$$\begin{aligned} C_{Tree} &= (n(m-2) + E)C_{FA} + nmC_{AND} \\ &= 16nm + o(nm), \end{aligned}$$

$$\begin{aligned} D_{Tree} &= 2(\mu+1)D_{FA} + D_{AND} \\ &= 12\mu + 14 \\ &= 12\lceil \log(m) \rceil - 10. \end{aligned}$$

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 $\frac{3}{4}M' \leqslant m' < M'$. We proceed as above and only analyze this case. The standard

length of $\frac{3}{2}$-adders and $\frac{4}{2}$-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

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

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

$$\begin{aligned} C'_{Tree} &= (n'(m' - 2) + E')C_{FA} + (n + 1)m'C_{SL} + m'C_{BD} \\ &= 24nm' + \mathrm{o}(nm'), \end{aligned}$$

$$\begin{aligned} D'_{Tree} &= 2(\mu' + 1)D_{FA} + D_{BD} + D_{SL} \\ &= 12(\mu' + 1) + 7. \end{aligned}$$

For $n = m = 53$ we get

$$C'_{Tree}/C_{Tree} = 37305/46288 = 80.6\%,$$

$$D'_{Tree}/D_{Tree} = 55/62 = 88.7\%$$

and for $n = m = 24$ we get

$$C'_{Tree}/C_{Tree} = 8531/9552 = 89.3\%,$$

$$D'_{Tree}/D_{Tree} = 43/50 = 86.0\%.$$

Asymptotically, C'_{Tree}/C_{Tree} tends again to $\frac{12}{16} = 75\%$. Unless m is a power of two, we have $\mu = \mu' + 1$ and $D_{Tree} - D'_{Tree} = 7$. Hence, D'_{Tree}/D_{Tree} 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.

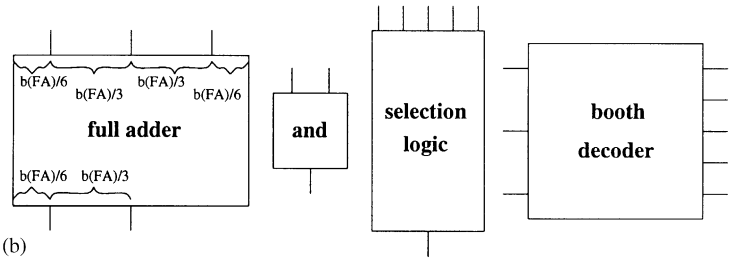
5. VLSI model

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

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	$\sqrt{11}$	$\sqrt{11}$

(a)



(b)

Fig. 9. (a) Table of basic circuit geometries; (b) shapes of basic circuits.

the same simple circuit in order to keep the analysis simple. We require

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

i.e. area equals gate count and

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

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 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

$$TIME(S,N) = D(S) + v|N|.$$

The parameter v weights the influence of wire delay on the total delay. If $v = 0$ 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 $v \geqslant 1$. Then

$$v|N| \geqslant 1 = D(NOT),$$

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 v to the interval $[0,1]$. We do not restrict the fanout of circuits, but within limits the parameter v 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. 9(a) happen to make the layouts of addition trees particularly simple. We place the pins of the full adder and the other basic circuits as specified in Fig. 9(b).[1] The exact position of the input pins of *AND* gates and the selection logic contributes only

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

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 $\frac{4}{2}$-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 decoders *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 $\frac{3}{2}$-adder and occupies one row of the matrix or it is a $\frac{4}{2}$-adder and occupies 2 consecutive rows of the matrix. Every bit position occupies a column of the matrix. Between neighboring full adders 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. 10). 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. 11(a)–(c):

- Box3: this is one full adder which is part of a leaf v of the tree with weight $L = 3$. The whole $\frac{3}{2}$-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 routed down to the next leaf of the tree. This—unfortunately—consumes horizontally distance 2δ.
- Box4: this is one pair of full adders which is part of a leaf v of the tree with weight $L = 4$. The whole $\frac{4}{2}$-adder v has as inputs 4 bits of b which are routed in the b-channel $b[2:0]$ of the first $\frac{3}{2}$-adder and in one b-channel of the second $\frac{3}{2}$-adder. Similarly, 4 a-channels are used. Each signal $a[i]$ is fed into 3 full adders of consecutive bit positions in the first $\frac{3}{2}$-adder level and then into

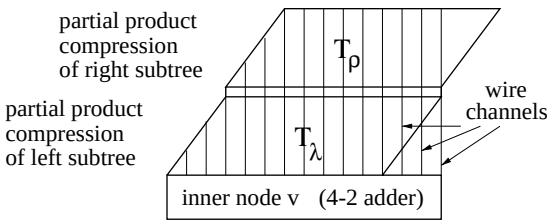


Fig. 10. Recursive partition of the tree layout.

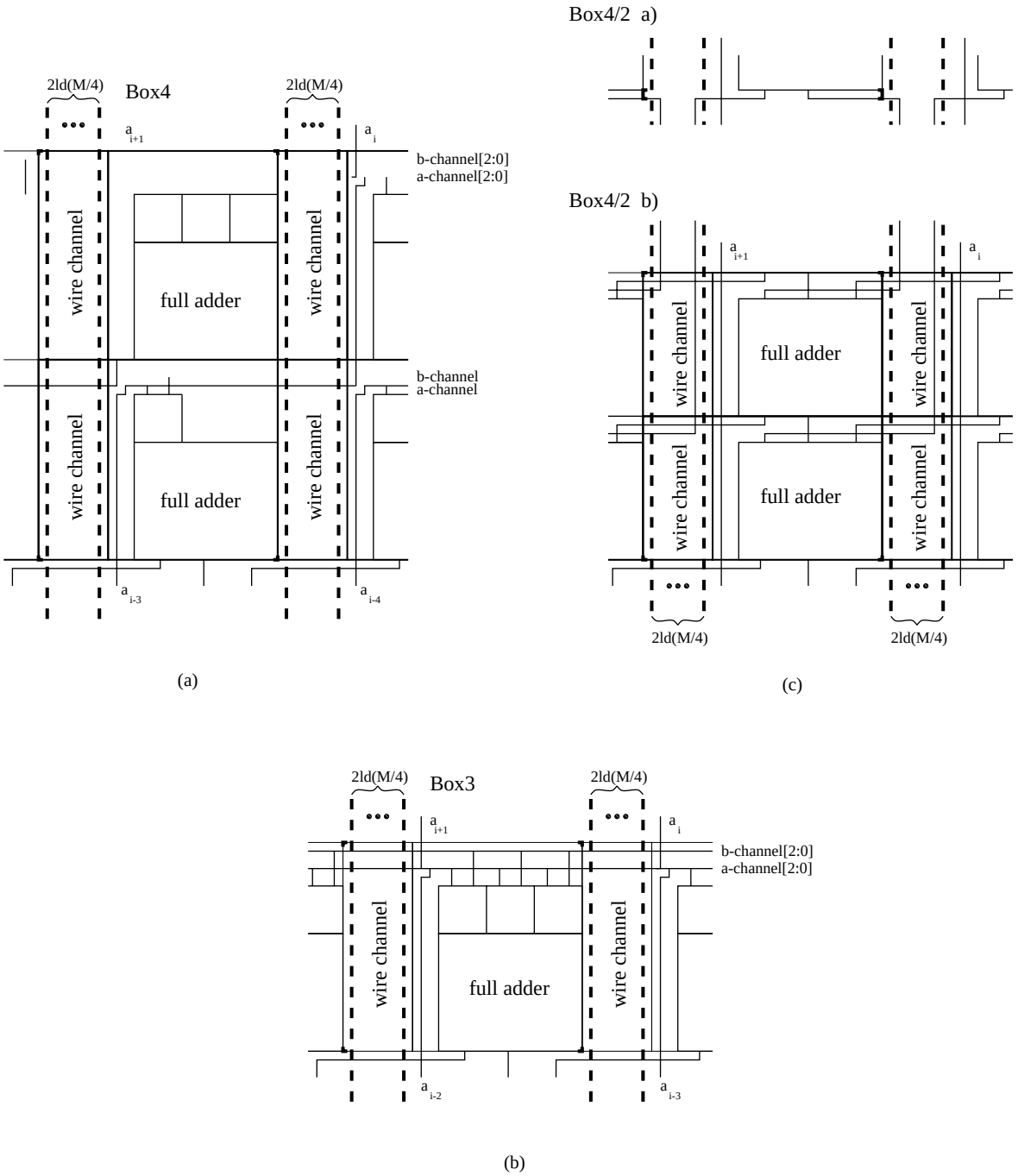


Fig. 11. Basic boxes for non-booth multiplier construction.

one full adder in the next bit position of the second $\frac{3}{2}$-adder. After that $a[i]$ is routed down to the next leaf of the tree.

- Box4/2: 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,

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

$Box4/2a$ 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

$$h(box4/2) = 2h(FA) + 7\delta,$$

$$h(box3) = h(FA) + h(AND) + 7\delta,$$

$$h(box4) = 2h(FA) + 2h(AND) + 11\delta.$$

The height of the whole layout is

$$h(T) = (M/4 - 1)h(box4/2) + (M/4)h(box3) + a(h(box4) - h(box3)).$$

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

- The horizontal extension $mb(box)$.
- 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

$$|a[i]| = mb(box) + h(T) - \mu(h(box4/2) - \delta) - h(FA) - h(AND) + 3m\delta.$$

For $m \leqslant n$ 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 $\frac{2}{3}$ 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—first full adder in v—carry output—second 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:

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

- 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

$$Sum_{4/2} = \mu(4b(FA)/3 + 4\delta) + \sum_{k=1}^{\mu} 2k\delta$$
$$= \mu(4b(FA)/3 + 4\delta) + \delta(\mu^2 + \mu).$$

- 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:

$$Channel = h(T) - h(box4) - \log(M/4)h(box4/2).$$

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 v.

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

$$TIME_{Tree}(n, v) = D_{AND} + 2D_{FA}(1 + \mu) + v(|a[i]| + |carry4| + Carry_{4/2} + Sum_{4/2} + Channel).$$

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. 12(a)–(c). 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 $\mu'\delta$.

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

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:

$$h(box4/2') = 2h(FA) + 7\delta,$$

$$h(box3') = h(FA) + h(SL) + b(SL) + 17\delta,$$

$$h(box4') = 2h(FA) + 2h(SL) + 26\delta.$$

Most of the other equations only change slightly:

$$h(T') = (M'/4 - 1)h(box4/2') + (M'/4)h(box3') + a'(h(box4') - h(box3')),$$

$$|a'[i]| = m'b(box') + h(T') - \mu'(h(box4/2' - \delta) - h(FA) - h(SL) + 4m'\delta,$$

$$|carry4'| = b(box') - 2b(FA)/3 + 3\delta,$$

$$Carry'_{4/2} = \mu'(b(box') - 2b(FA)/3 + 3\delta),$$

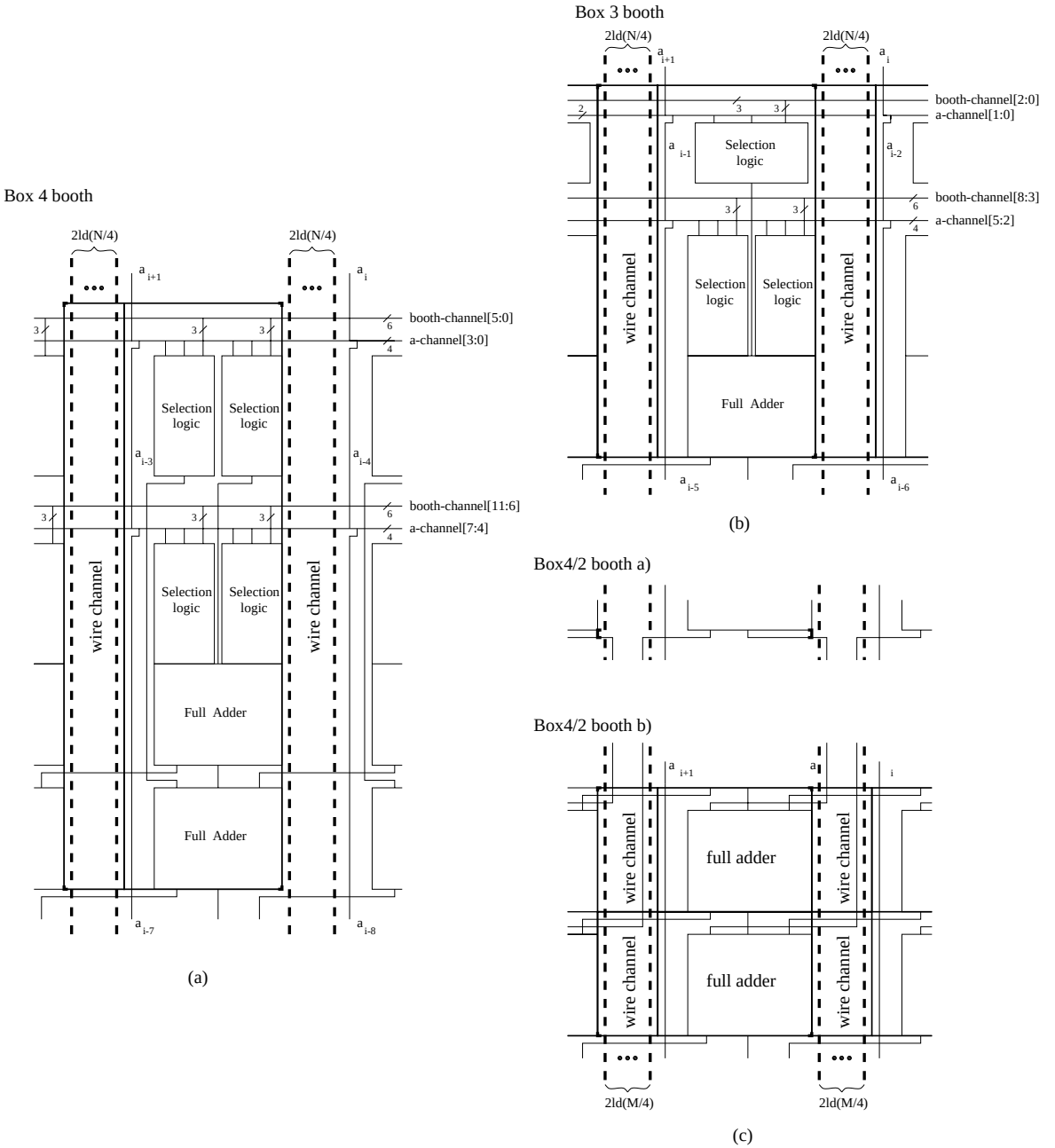


Fig. 12. Basic boxes for booth multiplier construction.

$$Sum'_{4/2} = \mu'(4b(FA)/3 + 4\delta) + \sum_{k=1}^{\mu'} 2k\delta,$$

$$= \mu'(4b(FA)/3 + 4\delta) + \delta(\mu'^2 + \mu'),$$

$$Channel' = h(T') - h(box4') - \log(M'/4)h(box4/2').$$

Additionally, to these wires we must consider the wire between one selection logic output in the upper row and the full-adder 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 v which one is slower. 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.

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

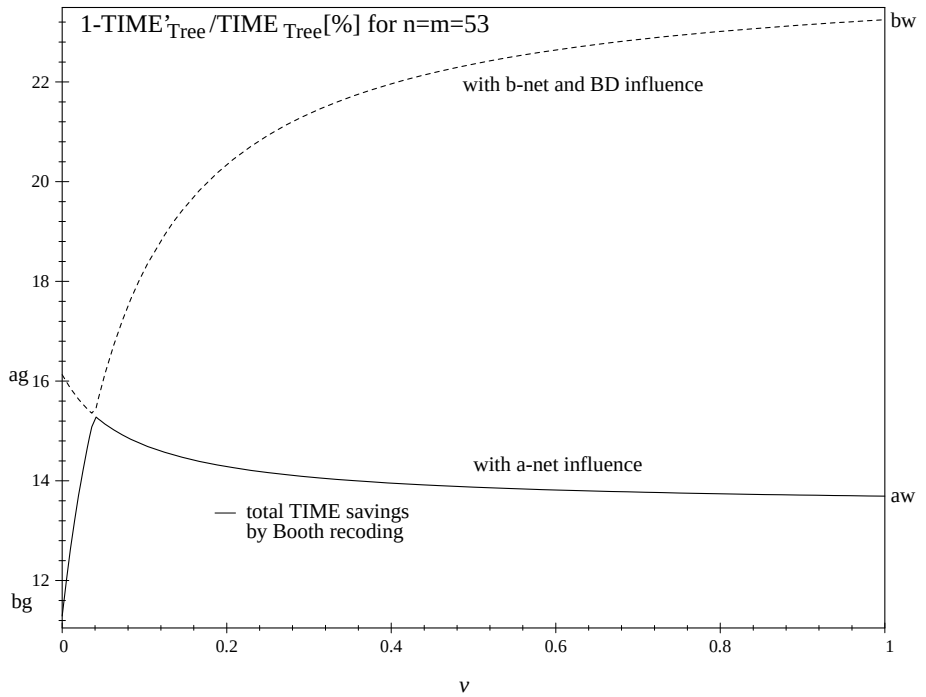


Fig. 13. Relative TIME improvement by Booth recoding for $n = m = 53$.

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

$$TIME'_{Tree}(n,v) = D_{SL} + 2D_{FA}(1+\mu') + v\left(max\left(|a'[i]|, |b'(i)| + \frac{D_{BD}}{v}\right) + |sum4'| + |carry4'| + Carry'_{4/2} + Sum'_{4/2} + Channel'\right).$$

In Fig. 13 the relative improvement $(TIME_{Tree} - TIME'_{Tree})/TIME_{Tree}$ is plotted for $n = m = 53$ as a function of v. This figure is surprisingly complex and counter intuitive. In particular we see gains due to Booth recoding *decrease* with increasing v, i.e. with slower wires. For small values of v until $v \approx 0.1$ 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 v we have

$$TIME'_{Tree} \approx 55 + 537v,$$

whereas for large v we have

$$TIME'_{Tree} \approx 52 + 614v.$$

We always have

$$TIME_{Tree} = 62 + 710v.$$

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

$$TIME'_{Tree}/TIME_{Tree} \approx 55/62 \approx 89\%.$$

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 v. For $v = 0$ the branch starts at $1 - \frac{52}{62} \approx 100\% - 84\% = 16\%$. For large v 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 recoding. 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 $v = 1$:

$$wire(T') \approx w(T')/2 + 2h(T'),$$

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

$$w(T) \approx w(T') \approx 2nw(box) \approx 2n5.7$$

If we would use as leaves only $\frac{4}{2}$-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, $h(T) \approx (n/4)(9.9 + 6.7) = 4.15n$. Similarly, we have $h(T') \approx (n/8)(h(box'4) + h(box4/2) \approx (n/8)(17.5 + 6.7) = 3.025n$. For $n = 53$ we get $wire(T') \approx 623$ and $wire(T) \approx 742$.

Note that we have $h(T)/w(T) \approx \frac{1}{3}$ and $h(T)/w(T') \approx \frac{1}{4}$. 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*

[2] This is common practice.

become slower!. This incidentally shows, that the common practice of making the layouts of addition trees roughly square is not always a good idea.

7. Evaluations

In this section, we evaluate the savings due to Booth recoding for a wide range of n and also analyze the asymptotical behaviour. The evaluations are presented in two parts: In the first part the gate model from Section 4 is considered, in the second part we consider the layout model from Section 6. To be able to compare the four different designs for every particular n, on the one hand layouts had to be analyzed also for the multiplication arrays. On the other hand, the delay, cost and area formulae of the multiplication tree designs had to be extended for the cases where $M/2 \leqslant m < 3/4M$. These extensions have been done following the presented designs in a straightforward way.

We do not only focus on the savings in cost, area and delay, but we consider the more general quality metrics, the InverseGateQuality (IGQ) and the InverseLayoutQuality (ILQ), that are defined below:

Definition 2. Depending on a quality paramater $0 \leqslant q \leqslant 1$, we define the *InverseGateQuality* (*IGQ*) of a design X, that has the cost $C_X(n,m)$ and the delay $D_X(n,m)$, by

$$IGQ_X(n,q) = C_X(n,n)^q D_X(n,n)^{(1-q)}.$$

Note, that we get the delay by $IGQ_X(n,0) = D_X(n,n)$ and the cost by $IGQ_X(n,1) = C_X(n,n)$. Because the delay $D_X(n,n)$ of a design X is the inverse of the performance of the design $P_X(n,n) = 1/D_X(n,n)$, for $q = \frac{1}{2}$ the IGQ metric relates to cost-performance ratio:

$$\begin{aligned} IGQ_X(n,0.5) &= \sqrt{C_X(n,n)D_X(n,n)} \\ &= \sqrt{C_X(n,n)/P_X(n,n)}. \end{aligned}$$

Definition 3. We define the IGQ of the best among our designs, that do not use Booth recoding by

$$bestIGQ(n,q) = min(IGQ_{Array}(n,q), IGQ_{Tree}(n,q)).$$

Analogously, the IGQ of the best among our designs using Booth recoding is defined by

$$bestIGQ'(n,q) = min(IGQ'_{Array}(n,q), IGQ'_{Tree}(n,q)).$$

Moreover, we define the relative savings of the IGQ due to Booth recoding:

$$IGQsave(n,q) = 1 - bestIGQ'(n,q)/bestIGQ(n,q).$$

Definition 4. Depending on a quality paramater $0 \leqslant q \leqslant 1$, we define the *InverseLayoutQuality* (*ILQ*) of a design X, that requires the area $AREA_X(n,m)$ and has the delay $TIME_X(n,m,v)$, given by

$$ILQ_X(n,v,q) = AREA_X(n,n)^q TIME_X(n,n,v)^{(1-q)}.$$

Note, that we get the *TIME* metric by $ILQ_X(n,v,0)$ and the *AREA* metric by $ILQ_X(n,v,1)$. Moreover, for $q=\frac{1}{2}$ and $q=\frac{2}{3}$ the ILQ metric relates to the metrics commonly known as AT and AT^2, respectively.

Definition 5. We define the ILQ of the best among our designs, that do not use Booth recoding by

$$bestILQ(n,v,q) = min(ILQ_{Array}(n,v,q), ILQ_{Tree}(n,v,q)).$$

Accordingly, the ILQ of the best among our designs using Booth recoding is defined by

$$bestILQ'(n,v,q) = min(ILQ'_{Array}(n,v,q), ILQ'_{Tree}(n,v,q)).$$

Moreover, we define the relative savings of the ILQ due to Booth recoding:

$$ILQsave(n,v,q) = 1 - bestILQ'(n,v,q)/bestILQ(n,v,q).$$

7.1. Gate analysis

7.1.1. Asymptotic considerations

The following lemma states, that for large n, the multiplication arrays have the best IGQ for $q=1$, whereas, multiplication trees have the best IGQ for $q<1$.

Lemma 6. *For large n*:

$$bestIGQ(n,q) = \begin{cases} IGQ_{Array}(n,q) & if\ \ q=1, \\ IGQ_{Tree}(n,q) & if\ \ 0\leqslant q<1, \end{cases}$$

$$bestIGQ'(n,q) = \begin{cases} IGQ'_{Array}(n,q) & if\ \ q=1, \\ IGQ'_{Tree}(n,q) & if\ 0\leqslant q<1. \end{cases}$$

Proof. We separate the proof for the two cases: (a) $q=1$; and (b) $0\leqslant q<1$:

(a) For $q=1$, the quality metric becomes the cost metric $IGQ_X(n,1)=C_X(n,n)$, so that from the cost formulae $C_{Tree}(n,n)=C_{Array}(n,n)+EC_{FA}$ and $C'_{Tree}(n,n)=C'_{Array}(n,n)+E'C_{FA}$ the proof of part (a) of the lemma already follows.

(b) For $0\leqslant q<1$ and large n, we consider the fraction

$$\begin{aligned} \frac{IGQ_{Array}(n,q)}{IGQ_{Tree}(n,q)} &= \frac{C_{Array}(n,n)^q D_{Array}(n,n)^{(1-q)}}{C_{Tree}(n,n)^q D_{Tree}(n,n)^{(1-q)}}. \\ &\geqslant 1^q \left(\frac{n}{2\log(n)} + o\left(\frac{n}{2\log(n)}\right)\right)^{1-q} \\ &\geqslant 1. \end{aligned}$$

The same argumentation can be used for the Booth designs, so that also part (b) of the proof is completed. □

Theorem 7. *For large n, Booth recoding improves the best IGQ by*

$$IGQsave(n,q) = 1 - \left(\frac{C_{FA} + C_{SL}}{2(C_{FA} + C_{AND})}\right)^q \pm o(1).$$

Proof. We separate the proof for the two cases: (a) $q = 1$; and (b) $0 \leqslant q < 1$:

(a) For $q = 1$, it follows from Lemma 6 that for large n

$$IGQsave(n,1) = 1 - IGQ'_{Array}(n,1)/IGQ_{Array}(n,1).$$

Because $IGQ_{Array}(n,1) = C_{Array}(n,n) = (C_{FA} + C_{AND})n^2 + o(n^2)$ and

$$\begin{aligned} IGQ'_{Array}(n,1) &= C'_{Array}(n,n) \\ &= (C_{FA} + C_{SL})n\left\lceil\frac{n+1}{2}\right\rceil + o(n^2) \\ &= \frac{C_{FA} + C_{SL}}{2}n^2 + o(n^2), \end{aligned}$$

we get $IGQ'_{Array}(n,1)/IGQ_{Array}(n,1) = (C_{FA} + C_{SL})/2(C_{FA} + C_{AND}) \pm o(1)$, so that for large n

$$IGQsave(n,1) = 1 - \frac{C_{FA} + C_{SL}}{2(C_{FA} + C_{AND})} \pm o(1).$$

(b) For $0 \leqslant q < 1$, it follows from Lemma 6 that for large n

$$IGQsave(n,q) = 1 - IGQ'_{Tree}(n,q)/IGQ_{Tree}(n,q).$$

Using $IGQ'_{Tree}(n,q) = C'_{Tree}(n,n)^q D'_{Tree}(n,n)^{(1-q)}$ and $IGQ_{Tree}(n,q) = C_{Tree}(n,n)^q D_{Tree}(n,n)^{(1-q)}$, we get

$$IGQsave(n,q) = 1 - \frac{C'_{Tree}(n,n)^q D'_{Tree}(n,n)^{(1-q)}}{C_{Tree}(n,n)^q D_{Tree}(n,n)^{(1-q)}}.$$

Because for large n, $D'_{Tree}(n,n)^{(1-q)}/D_{Tree}(n,n)^{(1-q)} = \mu'/\mu \pm o(1) = 1 \pm o(1)$, the equation for $IGQsave(n,q)$ can be simplified to

$$\begin{aligned} IGQsave(n,q) &= 1 - \frac{C'_{Tree}(n,n)^q}{C_{Tree}(n,n)^q} + o\left(\frac{C'_{Tree}(n,n)^q}{C_{Tree}(n,n)^q}\right) \\ &= 1 - \left(\frac{(C_{FA} + C_{SL})n\lceil (n+1)/2 \rceil}{(C_{FA} + C_{AND})n^2}\right)^q \pm o(1) \\ &\geqslant 1 - \left(\frac{C_{FA} + C_{SL}}{2(C_{FA} + C_{AND})}\right)^q \pm o(1). \end{aligned}$$

This completes the proof of the theorem. □

Corollary 8. *Asymptotically, Booth recoding saves 25% of the cost of the partial product generation and reduction (Theorem 7 with $q = 1$ and the presented basic circuit designs). The asymptotic relative IGQ savings are depicted in Fig. 14 as a function of the quality parameter q.*

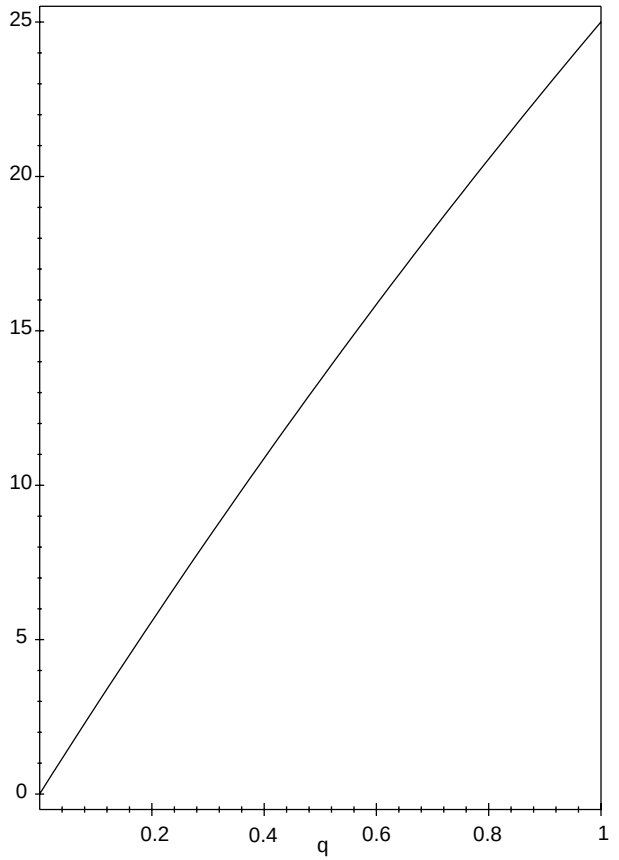


Fig. 14. Asymptotic relative savings (%) of the best IGQ (regarding gate model) due to Booth recoding as a function of q: $IGQsave(\infty, q)$.

7.1.2. Considerations for practical n

The relative savings of the best IGQ according to the use of Booth Recoding are depicted in Fig. 15 for $8 \leqslant n \leqslant 64$ and $0 \leqslant q \leqslant 1$. This figure shows that Booth recoding is useful in most practical situations, except for small n and $q \approx 1$. For $q = 1$ the savings of the IGQ due to Booth recoding are analyzed in detail. The following lemma states, that regarding the gate count ($IGQ_X(n, 1)$), Booth recoding is useful, iff $n = 13, 15$ or $n \geqslant 17$.

Lemma 9.

$$bestIGQ'(n, 1) < bestIGQ(n, 1) \Leftrightarrow ((n = 13)\ OR\ (n = 15)\ OR\ (n \geqslant 17)).$$

Proof. From Lemma 6 we get $bestIGQ'(n, 1) = C'_{Array}(n, n)$ and $bestIGQ(n, 1) = C_{Array}(n, n)$. We distinguish two cases: (a) n is even; and (b) n is odd:

(a) If $n = m$ is even, then we can substitute m' by $m' = n/2 + 1$ in $C'_{Array}(n, n)$. The condition $C'_{Array}(n, n) \leqslant C_{Array}(n, n)$ is then equivalent to $n^2 - \frac{139}{8}n + 49 \geqslant 0$, so that for even $n > 0$ we get the

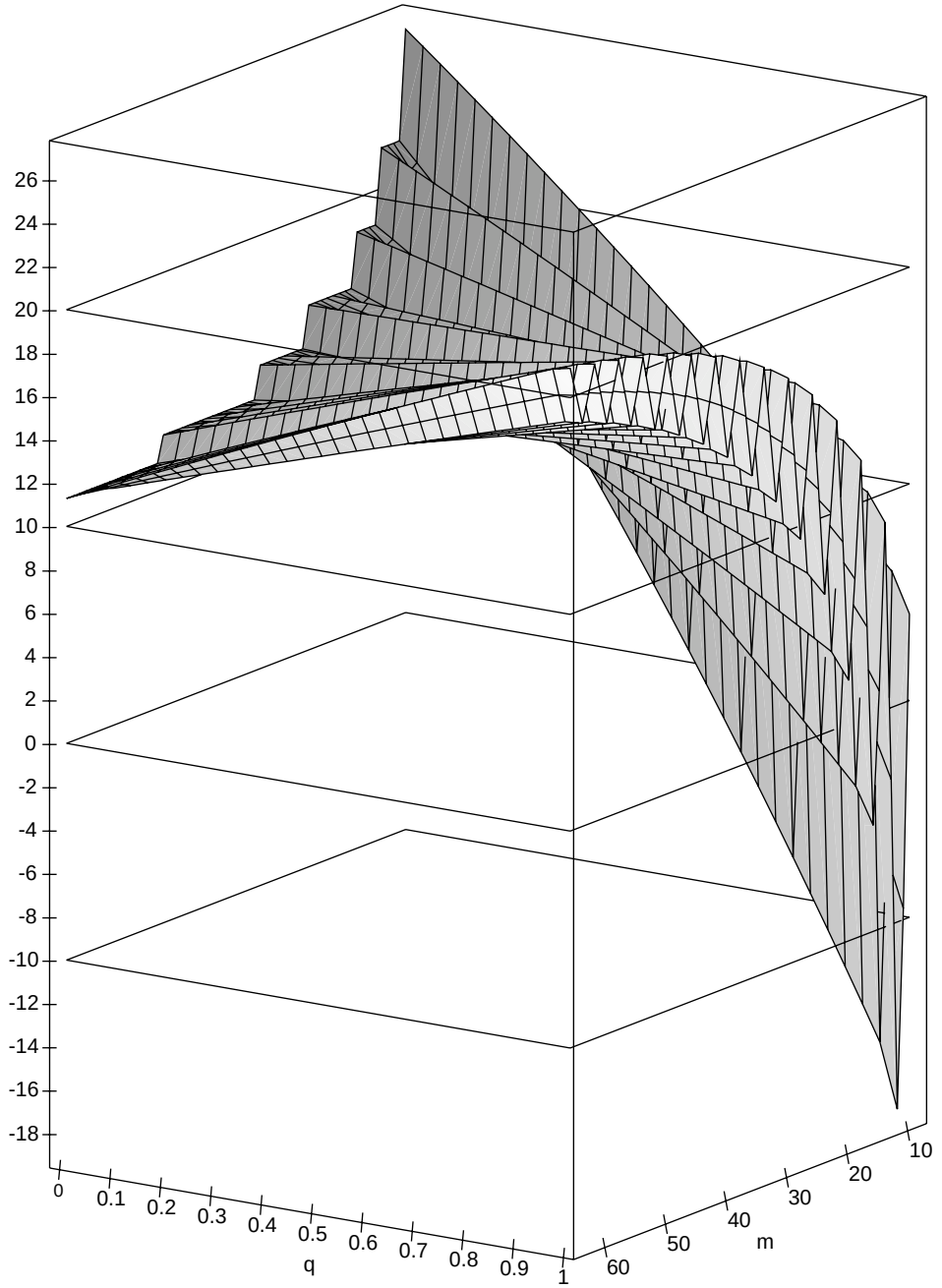


Fig. 15. Relative savings (%) of the best IGQ (regarding gate model) due to Booth recoding: $IGQsave(n, q)$.

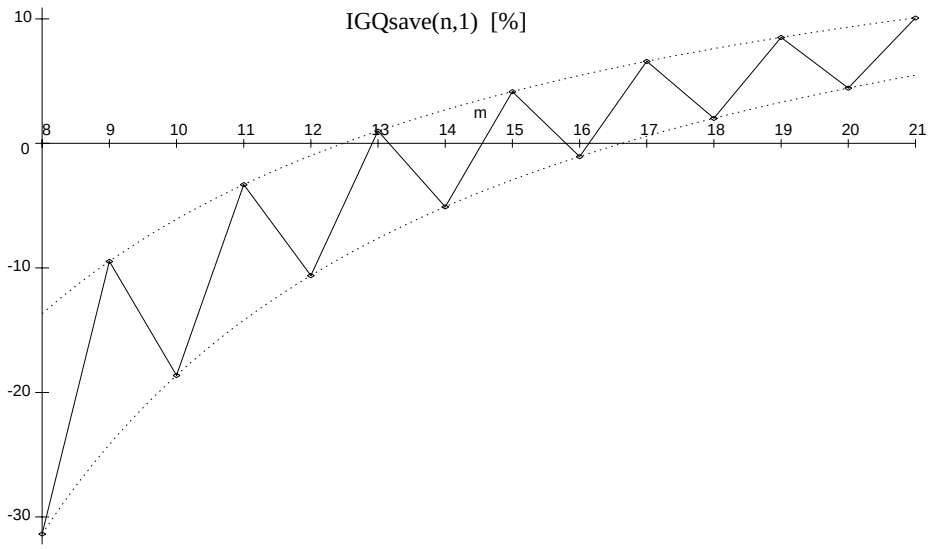


Fig. 16. Relative Savings (%) of the best IGQ due to Booth recoding for $q = 1$: $IGQsave(n, 1)$.

solution $n \geqslant 16.63$, which is equivalent to even $n \geqslant 18$. (b) If $n = m$ is odd, then we can substitute m' by $m' = n/2 + \frac{1}{2}$ in $C'_{Array}(n, n)$. The condition $C'_{Array}(n, n) \leqslant C_{Array}(n, n)$ is then equivalent to $n^2 - \frac{115}{8}n + \frac{189}{8} \geqslant 0$, so that for odd $n > 0$ we get the solution $n \geqslant 12.48$, which is equivalent to odd $n \geqslant 13$. Thus, the solutions of part (a) and part (b) combine to the lemma. The situation of the IGQ savings due to Booth recoding $IGQsave(n, 1)$ is depicted in Fig. 16 for $8 \leqslant n \leqslant 20$. □

Finally, Fig. 17 depicts which of the 4 different designs should be chosen according to the IGQ for $0 \leqslant q \leqslant 1$, $8 \leqslant n \leqslant 64$ and $10 \leqslant n \leqslant 10^{10}$.

7.2. Layout analysis

7.2.1. Asymptotic considerations

Lemma 10. *With the definition of the condition* $COAR \Leftrightarrow ((0 \leqslant q < 1)\ AND\ (v \neq 0))\ OR\ (q = 1)$, *the multiplication array designs have the asymptotically best ILQ for* $(COAR = 1)$. *For* $(COAR = 0)$ *the multiplication tree designs have the asymptotically best ILQ, so that for large n*

$$bestILQ(n, v, q) = \begin{cases} ILQ_{Array}(n, v, q) & if\ \ COAR, \\ ILQ_{Tree}(n, v, q) & otherwise. \end{cases}$$

$$bestILQ'(n, v, q) = \begin{cases} ILQ'_{Array}(n, v, q) & if\ \ COAR, \\ ILQ'_{Tree}(n, v, q) & otherwise. \end{cases}$$

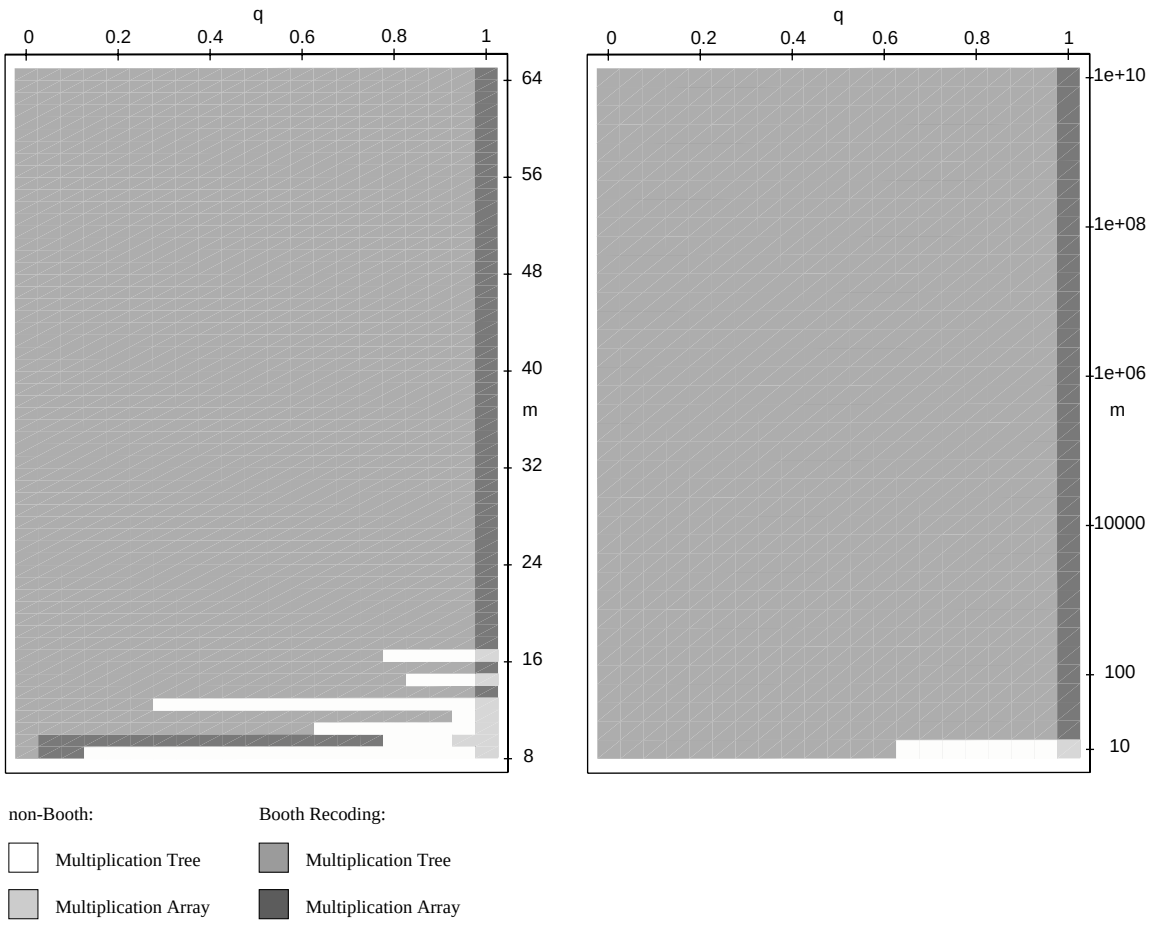


Fig. 17. Value ranges of q and n for that the 4 designs have the best IGQ (regarding gate model).

Proof. We separate the proof for the two cases: (a) ($v = 0$); and (b) ($0 < v \leqslant 1$):

(a) The proof for ($v = 0$) is again partitioned into two parts for the cases: (i) ($q = 1$); and (ii) ($0 \leqslant q < 1$):

(i) From ($q = 1$) and

$$ILQ_{Tree}(n, 0, 1) = AREA_{Tree}(n, n) = \Omega(n^2 \log(M)),$$
$$ILQ'_{Tree}(n, 0, 1) = AREA'_{Tree}(n, n) = \Omega(n^2 \log(M)),$$
$$ILQ_{Array}(n, 0, 1) = AREA_{Array}(n, n) = \mathrm{O}(n^2),$$
$$ILQ'_{Array}(n, 0, 1) = AREA'_{Array}(n, n) = \mathrm{O}(n^2),$$

it follows, that for large n

$$bestILQ(n, 0, 1) = ILQ_{Array}(n, 0, 1),$$
$$bestILQ'(n, 0, 1) = ILQ'_{Array}(n, 0, 1).$$

(ii) For ($v = 0$),

$$\begin{aligned} ILQ_{Array}(n,0,q) &= AREA_{Array}(n,n)^q TIME_{Array}(n,n,0)^{(1-q)} \\ &= \Theta(n^{(q+1)}), \end{aligned}$$

$$\begin{aligned} ILQ_{Tree}(n,0,q) &= AREA_{Tree}(n,n)^q TIME_{Tree}(n,n,0)^{(1-q)} \\ &= \Theta(n^{(2q)}\log(n)). \end{aligned}$$

Because for $0 \leqslant q < 1$:

$$\begin{aligned} \frac{n^{(q+1)}}{n^{(2q)}\log(n)} &= n^{1-q}/\log(n) \\ &\geqslant 1 + \Omega(1). \end{aligned}$$

We get for large n

$$bestILQ(n,0,q) = ILQ_{Tree}(n,0,q).$$

In the same way we can show for $0 \leqslant q < 1$, that for large n:

$$bestILQ'(n,0,q) = ILQ_{Tree}(n,0,q).$$

This completes part (a) of the proof.

(b) In the following, we assume for the proof of case (b), that ($0 < v \leqslant 1$): Because $TIME_{Tree}(n,n,v) = \Omega(n\log(n))$, $TIME'_{Tree}(n,n,v) = \Omega(n\log(n))$, $TIME_{Array}(n,n,v) = O(n)$ and $TIME'_{Array}(n,n,v) = O(n)$, we get for large n

$$bestILQ(n,v,0) = TIME_{Array}(n,n,v), \tag{1}$$

$$bestILQ'(n,v,0) = TIME'_{Array}(n,n,v). \tag{2}$$

Accordingly, from $AREA_{Tree}(n,n) = \Omega(n^2\log(n))$, $AREA'_{Tree}(n,n) = \Omega(n^2\log(n))$, $AREA_{Array}(n,n) = O(n^2)$ and $AREA'_{Array}(n,n) = O(n^2)$, it follows for large n that

$$bestILQ(n,v,1) = AREA_{Array}(n,n), \tag{3}$$

$$bestILQ(n,v,1)' = AREA'_{Array}(n,n). \tag{4}$$

Because for $v > 0$, the multiplication arrays have both, the asymptotically smallest AREA and the asymptotically fastest TIME, we get for large n

$$\begin{aligned} bestILQ(n,v,q) &= AREA_{Array}(n,n)^q TIME_{Array}(n,n,v)^{(1-q)} \\ &= ILQ_{Array}, \end{aligned}$$

$$\begin{aligned} bestILQ'(n,v,q) &= AREA'_{Array}(n,n)^q TIME'_{Array}(n,n,v)^{(1-q)} \\ &= ILQ'_{Array}. \end{aligned}$$

This completes case (b) of the proof. □

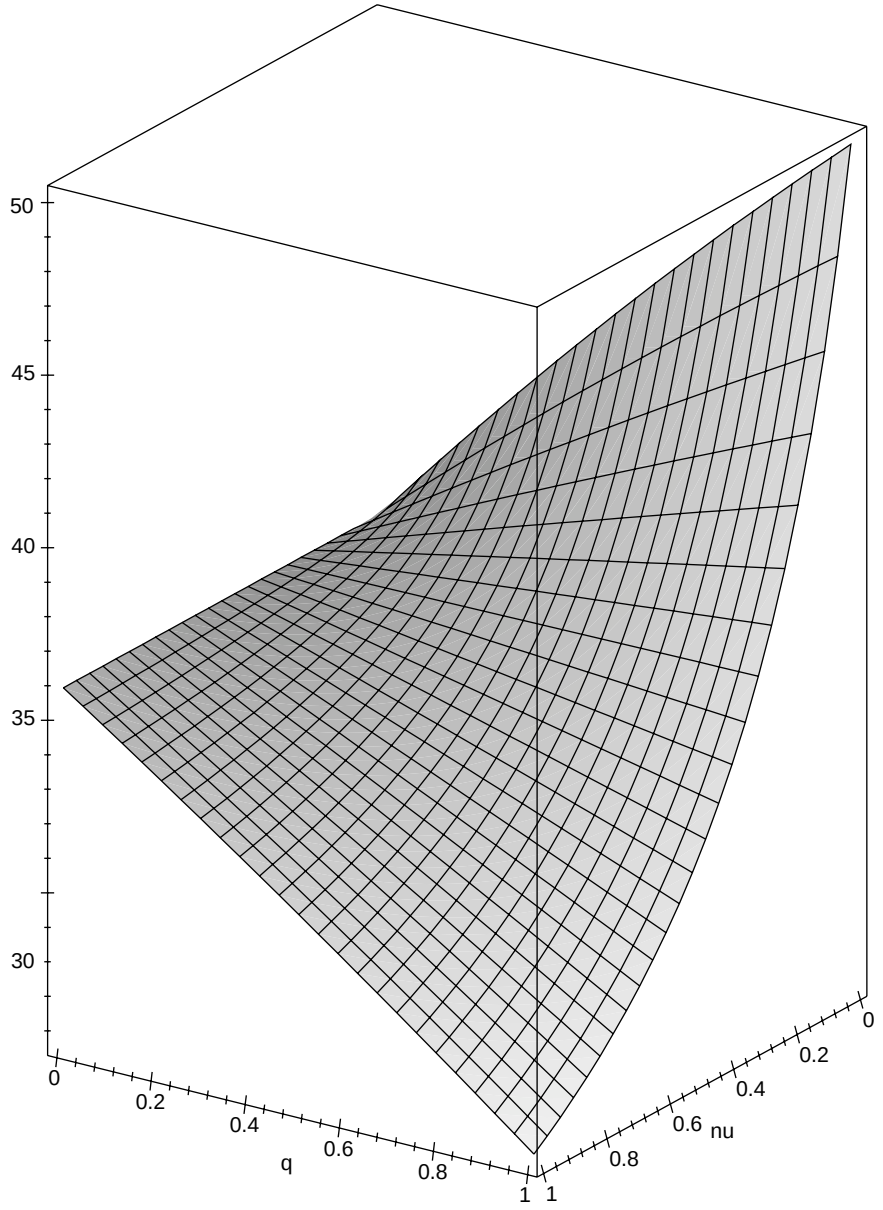


Fig. 18. Asymptotic relative savings (%) of the best ILQ (regarding layout model) due to Booth recoding for $0<v\leqslant 1$ and $0\leqslant q\leqslant 1$.

Theorem 11. *We use the previous definition of the condition* $COAR \Leftrightarrow ((0\leqslant q<1)$ *AND* $(v\neq 0))$ *OR* $(q=1)$. *Then, for large n the use of Booth recoding relatively saves the best ILQ by*

$$ILQsave(n,v,q) = \begin{cases} 1 - 0.641^q \left(\dfrac{11.66v+3}{13.75v+6}\right)^{(1-q)} \pm o(1) & \text{if } COAR, \\ 1 - 0.728^q \pm o(1) & \text{otherwise.} \end{cases}$$

The asymptotic relative ILQ savings are depicted in Fig. 18 *as a function of q and v.*

Proof. First, we summarize the *TIME* and *AREA* formulae for the different designs:

$$AREA'_{Array}(n,n) = 15.49n^2 + o(n^2),$$

$$AREA_{Array}(n,n) = 23.92n^2 + o(n^2),$$

$$AREA'_{Tree}(n,n) = 1.26n^2 \log(M) + o(n^2 \log(M)),$$

$$AREA_{Tree}(n,n) = 1.73n^2 \log(M) + o(n^2 \log(M)),$$

$$TIME'_{Array}(n,n,v) = 11.66vm + o(vm) + 3m + o(m),$$

$$TIME_{Array}(n,n,v) = 13.75vm + o(vm) + 6m + o(m),$$

$$TIME'_{Tree}(n,n,v) = 0.2vn \log(M) + o(vn \log(M) + 12 \log(M) + o(\log(M)),$$

$$TIME_{Tree}(n,n,v) = 0.2vn \log(M) + o(vn \log(M) + 12 \log(M) + o(\log(M)).$$

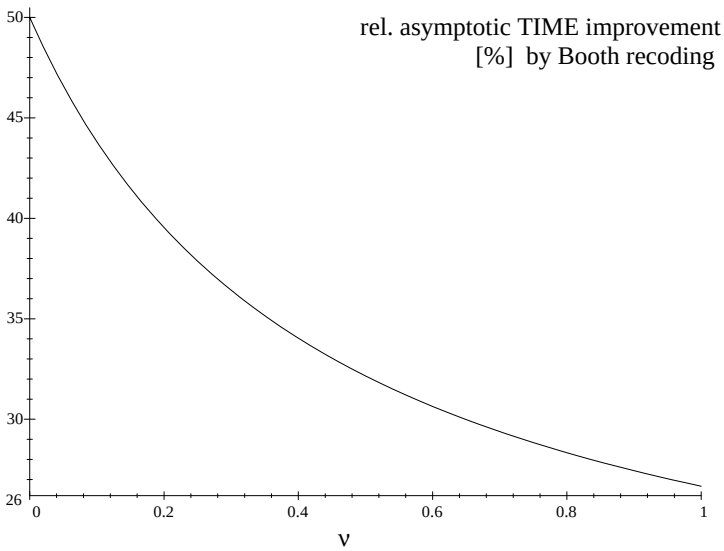


Fig. 19. Asymptotic TIME savings due to Booth recoding.

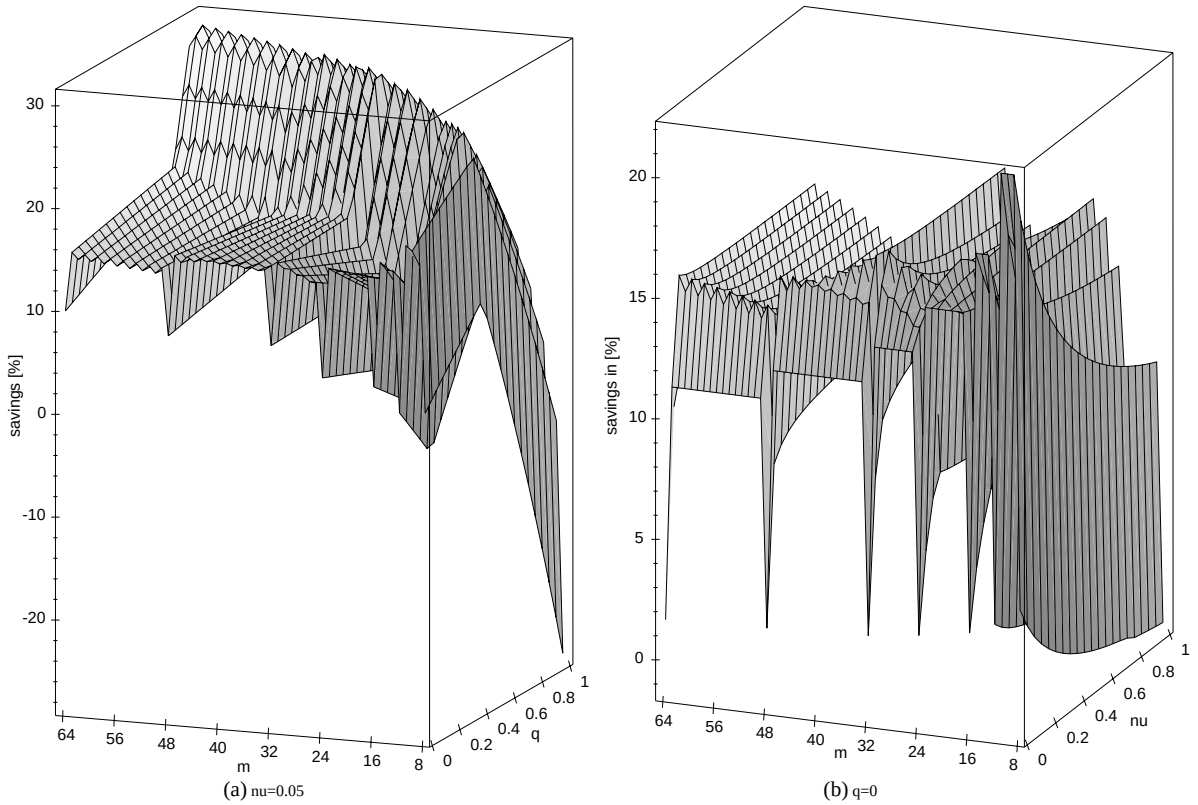


Fig. 20. Relative savings (%) of the best ILQ (regarding layout model) due to Booth recoding for (a) $0<q\leqslant 1$, $8\leqslant m\leqslant 64$ and $v=0.05$; and (b) $0\leqslant v\leqslant 1$, $8\leqslant m\leqslant 64$ and $q=0$.

From these equations, we can easily derive

$$\frac{ILQ'_{Array}(n,v,q)}{ILQ_{Array}(n,v,q)}=\frac{AREA'_{Array}(n,n)^q}{AREA_{Array}(n,n)^q}\frac{TIME'_{Array}(n,n,v)^{(1-q)}}{TIME_{Array}(n,n,v)^{(1-q)}}$$

$$=\left(\frac{15.49}{23.92}\right)^q\left(\frac{11.66v+3}{13.75v+6}\right)^{(1-q)}\pm o(1)$$

$$\frac{ILQ'_{Tree}(n,v,q)}{ILQ_{Tree}(n,v,q)}=\frac{AREA'_{Tree}(n,n)^q}{AREA_{Tree}(n,n)^q}\frac{TIME'_{Tree}(n,n,v)^{(1-q)}}{TIME_{Tree}(n,n,v)^{(1-q)}}$$

$$=\left(\frac{1.26}{1.73}\right)^q 1^{(1-q)}\pm o(1).$$

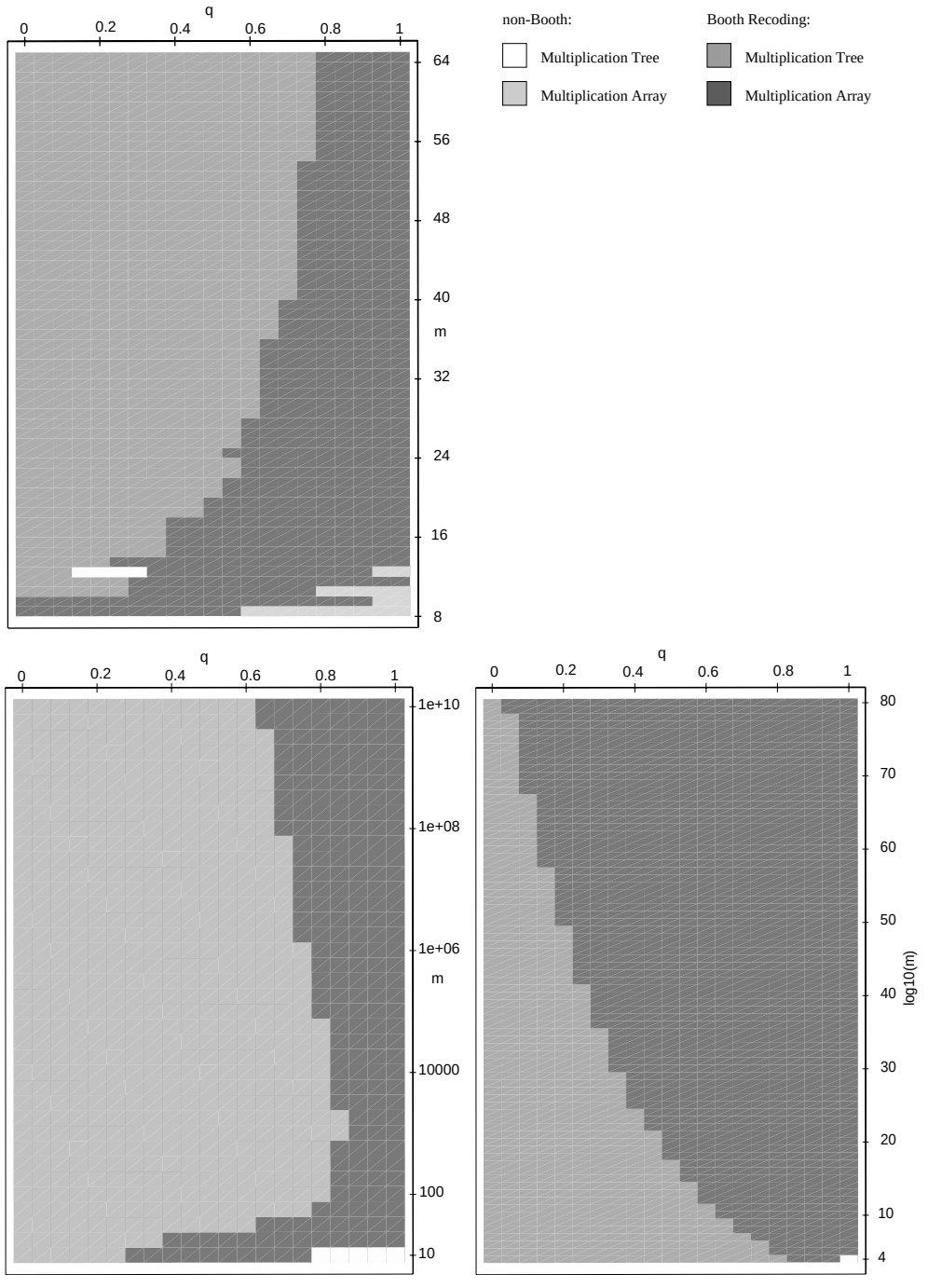


Fig. 21. Value ranges of q and n for that the 4 designs have the best ILQ (regarding layout model).

Finally, by the use of Lemma 10, we get, that for large n

$$ILQsave(n,v,q) = \begin{cases} 1 - \frac{ILQ'_{Array}(n,v,q)}{ILQ_{Array}(n,v,q)} & \text{if } COAR, \\ 1 - \frac{ILQ'_{Tree}(n,v,q)}{ILQ_{Tree}(n,v,q)} & \text{otherwise.} \end{cases}$$

$$= \begin{cases} 1 - 0.641^q \left(\frac{11.66v+3}{13.75v+6}\right)^{(1-q)} \pm o(1) & \text{if } COAR, \\ 1 - 0.728^q \pm o(1) & \text{otherwise.} \end{cases} \quad \square$$

Corollary 12. *From Theorem* 11 *we can extract the AREA savings* ($q = 1$) *for large n*:

$$ILQsave(n,v,1) = 35.9 \pm o(1).$$

Moreover, we can extract the TIME savings ($q = 0$) *for large n*:

$$ILQsave(n,v,0) = \begin{cases} \left(\frac{11.66v+3}{13.75v+6}\right) \pm o(1) & \textit{if } (v > 0), \\ \pm o(1) & \textit{otherwise.} \end{cases}$$

The asymptotic TIME savings are depicted in Fig. 19.

7.2.2. Considerations for practical n

The relative savings of the best ILQ according to the use of Booth recoding are depicted in Fig. 20(a) for $8 \leqslant n \leqslant 64$, $0 \leqslant q \leqslant 1$ and $v = 0.05$, and in Fig. 20(b) for $8 \leqslant n \leqslant 64$, $q = 0$ and $0 \leqslant v \leqslant 1$. These figures show that Booth recoding is useful for most practical parameters regarding the ILQ.

Fig. 21 depicts which of the 4 different layouts should be chosen according to the ILQ for $0 \leqslant q \leqslant 1$ and $8 \leqslant n \leqslant 64$. Because the best practical ILQ layouts in this range do not seem to converge to the best asymptotical ILQ layouts from Lemma 10, the choice of the best ILQ layout is also depicted for a larger range of n with $10 \leqslant n \leqslant 10^{10}$ and an even larger range of n with $10^4 \leqslant n \leqslant 10^{80}$ in Fig. 21.

8. Conclusions

We formally investigate the complexity of Booth recoding for fixed point multiplication. As main contribution of this investigation we provide a formal version of the folklore theorem, that Booth recoding saves the delay and the area of the multipliers' adder tree by constant factors between 26% and 50% minus low order terms which depend on the length n of the operands and a parameter v for the wire delay. For the cost analysis we provide a formal version of the folklore theorem that balanced addition trees in n-bit multipliers have $n^2 + O(n \log n)$ full adders. This required the combinatorial argument from Section 3. In our investigations we did not only analyze the costs and delays considering gates, but we also considered layouts including the effects of wire delays. For practical n in the range $8 \leqslant n \leqslant 64$ we specify the delay and the area gain due to

Booth recoding exactly for the proposed designs. Parts of our gate analysis are already used and referenced by [28].

Our analysis are not only parametrical considering the operand size n, but also the relative delay of the wires, the costs and delays of basic gates and the geometries of the basic circuits can be adjusted by parameters, so that the interested reader may choose his favorite technology/implementation in the analysis. Regarding the wire delay parameter v, we have demonstrated the surprising effect, that for particular multiplier designs the relative delay savings due to Booth recoding might decrease as wires become slower. Empirical studies on a limited number of designs so far have suggested that Booth recoding should be particularly helpful if wires become slow [16]. We have analyzed this effect in some more detail.

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 [2,6,8–10,22,23,29–32].
- 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.

Acknowledgements

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

References

[1] L. Dadda, Some schemes for parallel multipliers, Alta Frequenza 34 (1965) 349–356.
[2] B.C. Drerup, E.E. Swartzlander, Fast multiplier bit-product matrix reduction using bit-ordering and parity generation, J. VLSI Signal Process. 7 (1994) 249–257.
[3] J. Keller, W.J. Paul, Hardware Design, 2nd Edition, Teubner Verlagsgesellschaft, Stuttgart, Leipzig, 1997.
[4] I. Koren, Computer Arithmetic Algorithms, Prentice-Hall International, Englewood Cliffs, NJ, 1993.
[5] A.R. Omondi, Computer Arithmetic Systems; Algorithms, Architecture and Implementations, Series in Computer Science, Prentice-Hall International, Englewood Cliffs, NJ, 1994.
[6] N. Takagi, H. Yasuura, S. Yajima, High-speed VLSI multiplication algorithm with a redundant binary addition tree, IEEE Trans. Comput. C-34 (9) (1985) 217–220.
[7] C.S. Wallace, A suggestion for parallel multipliers, IEEE Trans. Electron. Comput. EC-13 (1964) 14–17.
[8] Z. Wang, A. Jullien, C. Miller, A new design technique for column compression multipliers, IEEE Trans. Comput. 44 (8) (1995) 962–970.

[9] H. Al-Twaijry, Area and performance optimized CMOS multipliers, Ph.D. Thesis, Stanford University, August 1997.
[10] G.W. Bewick, Fast multiplication: algorithms and implementation, Ph.D. Thesis, Stanford University, March 1994.
[11] A.D. Booth, A signed binary multiplication technique, Q. J. Mech. Appl. Math. 4 (2) (1951) 236–240.
[12] P.E. Madrid, B. Millar, E.E. Swartzlander, Modified Booth algorithm for high radix multiplication, IEEE Computer Design Conference, 1992, pp. 118–121.
[13] L.P. Rubinfeld, A proof of the modified Booth's algorithm for multiplication, IEEE Trans. Comput. (October 1975), pp. 1014–1015.
[14] H. Al-Twaijry, M. Flynn, Multipliers and datapaths, Technical Report CSL-TR-94-654, Stanford University, December 1994.
[15] H. Al-Twaijry, M. Flynn, Performance/area tradeoffs in Booth multipliers, Technical Report CSL-TR-95-684, Stanford University, November 1995.
[16] H. Al-Twaijry, M. Flynn, Technology scaling effects on multipliers, Technical Report CSL-TR-96-698, Stanford University, July 1996.
[17] I. Wegener, The Complexity of Boolean Functions, Wiley, New York, 1987.
[18] C.D. Thompson, Area-time complexity for VLSI. Proceedings of the 11th Annual ACM Symposium on the Theory of Computing, Vol. 11, 1979, pp. 81–88.
[19] J.D. Ullman, Computational Aspects of VLSI, Computer Science Press, Rockville, MD, 1984.
[20] L.A. Glasser, D.W. Dobberpuhl, The Design And Analysis Of VLSI Circuits, Addison-Wesley, Reading, MA, 1985.
[21] C. Mead, L. Conway, Introduction to VLSI Systems, Addison-Wesley, Reading, MA, 1980.
[22] L. Kuehnel, H. Schmeck, A closer look at VLSI multiplication, INTEGRATION, VLSI J. 6 (1988) 345–359.
[23] R.K. Yu, G.B. Zyner, 167 MHz Radix-4 floating point multiplier, Proceedings of the 12th Symposium on Computer Arithmetic, Vol. 12, 1995, pp. 149–154.
[24] P. Kornerup, private communication.
[25] S.M. Mueller, W.J. Paul, The Complexity of Simple Computer Architectures, in: Lecture Notes in Computer Science, Vol. 995, Springer, Berlin, 1995.
[26] J. Vuillemin, A very fast multiplication algorithm for VLSI implementation, INTEGRATION, VLSI J. 1 (1983) 39–52.
[27] C. Nakata, J. Brock, H4C Series: Design Reference Guide, CAD, 0.7 Micron L_{eff}, Motorola Ltd., 1993, Preliminary.
[28] S.M. Mueller, W.J. Paul, Computer Architecture—Complexity and Correctness, Springer, Berlin, 2000 ISBN# 3-540-67481-0.
[29] Z.-J. Mou, F. Jutand, Overturned-stairs adder trees and multiplier design, IEEE Trans. Comput. 41 (8) (1992) 940–948.
[30] 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, IEEE Trans. Comput. 45 (3) (1996) 294–306.
[31] R.M. Owens, R.S. Bajwa, M.J. Irwin, Reducing the number of counters needed for integer multiplication, Proceedings 12th Symposium on Computer Arithmetic, Vol. 12, 1995, pp. 38–41.
[32] K.F. Pang, R. Soong, H.-W. Sexton, P.H. Ang, Generation of high speed CMOS multiplier-accumulators, Proceedings IEEE International Conference on Computer Design: VLSI in Computers and Processors, 1988, pp. 217–220.