Binary Multiplication Radix-32 and Radix-256

Peter-Michael Seidel, Lee D McFearin, David W Matula

Department of Computer Science and Engineering
Southern Methodist University
Dallas, TX, 75275
{seidel,mcfearin,matula}@seas.smu.edu

Abstract

Multipliers are used at many different places in microprocessor design. As the non-memory sub-blocks of the microprocessor with the largest size and delay, multipliers have a big impact on the cycle time of the microprocessor. Targeting deeper pipelines and higher clock frequencies, there is a growing demand for multiplier designs that can be split into shorter stages. For this purpose, the use of Booth recoding has been a popular method to cut down the number of partial products in a multiplier, to reduce the delay of the partial product accumulation and to simplify the partition of the multiplier into several shorter stages. The complexity to pre-compute an increasing number of digit multiples of the multiplicand within the multiplier unit limits the use of Booth recoding mainly to radices 4 and 8.

We propose novel encoding schemes for the implementation of higher radix multiplication. In particular, we consider multiplication radix-32 and radix-256. In the high-radix representations each digit of the multiplier is represented in a secondary radix which is 7 in the case of radix-32 and which is 11 in the case of radix-256, so that the multiplier is represented by roughly $2p/5$ *resp.* $3p/8$ *terms. All non zero digits of the secondary radix system are a power of two, simplifying partial product generation. The partial products depending on multiples of the radices 7 or 11 can be separately accumulated, with multiplication by the radix a pre- or post-computation option. These features provide more flexible multiplier designs that can be implemented in shorter pipeline stages. We compare the proposed designs with multipliers that use traditional Booth recoding.*

1 Introduction

Binary multiplication is one of the most frequently used arithmetic operations in microprocessors. The implementation of binary multiplication is complex and multipliers are the largest and slowest non-memory sub-blocks in most processors. For this reason their implementations have a big impact on the cycle time of the processors. Targeting higher clock frequencies and therefore deeper pipelines for these processors creates a growing demand for multiplier designs that can be split into shorter stages.

We focus in this paper on the generation and reduction of partial products, which is the first and most costly sub-step in binary multiplication. Booth recoding [3, 10] is a commonly used technique to encode the multiplier by fewer digits, and hence, reduce the number of partial products. Booth radix-4 reduces the number of partial products from p to $\lceil \frac{p+1}{2} \rceil$. Booth radix-8 reduces this number to $\lceil \frac{p+1}{3} \rceil$ at the cost of pre-computing 3 times the multiplicand and routing two values into each partial product generator. Our principal result is the demonstration that the number of partial products can be reduced to $\lceil \frac{2(p+1)}{5} \rceil$ and further to $\lceil \frac{3(p+1)}{8} \rceil$ while maintaining the simpler Booth radix-4 properties that no odd multiple of the multiplicand need be pre-computed and that each PPG receives only the multiplicand for selective shift and/or complementation.

We obtain these properties by novel representation schemes for the implementation of higher radix multiplication. In particular we are targeting the design of multipliers radix-32 and radix-256. For these 'high-radix' multipliers each of the 'high-radix' digits of the p-bit multiplier is represented in a secondary radix. We will show that each radix-32 digit can be represented by two radix-7 digits, and that each radix-256 digit can be represented by three radix-11 digits, hence the multiplier is represented by roughly $2p/5$ resp. $3p/8$ terms. Moreover, we show that all the non zero digits in our secondary radix system are a power of two, which simplifies the implementation of the partial product generation. The partial products depending on multiples of the radices 7 or 11 can be separately accumulated, with multiplication by the radix a pre- or post-computation option. These features provide more flexible multiplier designs that can be implemented in shorter pipeline stages.

0-7695-1150-3/01 $10.00 © 2001 IEEE

Section 2 of this paper introduces notation and briefly reviews conventional multiplier designs and higher radix representations. Section 3 describes the foundations of secondary radix representation schemes. Section 4 introduces multiplier designs based on the representation schemes from Section 3 and compares them with multipliers that use conventional Booth recoding. We conclude in Section 5 and propose further work.

2 Preliminaries

For a bit-string $a = a[p-1:0] = (a[p-1], \ldots, a[0]) = (a_{p-1}, \ldots, a_0) \in \{0,1\}^p$ we denote by $\langle a \rangle = \sum_{i=0}^{p-1} a[i] \cdot 2^i$ the binary number represented by a.

A $p \times p$-*multiplier* is a circuit with p inputs $a = a[p-1:0]$, p inputs $b = b[p-1:0]$ and $2p$ outputs $c = c[2p-1:0]$ such that $\langle a \rangle \cdot \langle b \rangle = \langle c \rangle$ holds.

2.1 Binary Multiplication

In general binary $p \times p$-*multipliers* are implemented with a first step of generating the product in a carry-save representation and a second step of compressing the carry-save representation to a binary representation of the product. We focus on implementation of the first step. In the simplest form the product computation is based on the sum:

$$\langle c \rangle = \sum_{i=0}^{p-1} \langle a \rangle \cdot b[i] \cdot 2^i \qquad (1)$$

with the *partial products*: $S_i = \langle a \rangle \cdot b[i] \cdot 2^i$. These partial products have to be generated and compressed to a carry-save representation. The generation of the partial products corresponding to (1) simply consists of logical AND-gates. Except for optimizing the logical and physical implementation of the partial product reduction, the main approach to decrease the delay and size of the partial product reduction is to decrease the number of partial products in (1) by representing one of the operands in a higher radix.

2.2 Higher Radix Partial Product Generation

Let a p-digit string in radix β denote the radix polynomial

$$(d_{p-1}, d_{p-2}, \ldots, d_0)_\beta \equiv d_{p-1}\beta^{p-1} + d_{p-2}\beta^{p-2} + \ldots + d_0 \in P[\beta, D].$$

Here $\beta \geq 2$ is the *radix*, D is the *digit set* with $d_i \in D$ for $0 \leq i \leq p-1$, and the *radix system* $P[\beta, D]$ denotes the set of all radix polynomials with radix β and digits from D.

For the product $A \cdot B$ of the p-bit integer multiplicand $A = \langle a \rangle$ and the p-bit integer multiplier $B = \langle b \rangle$, let the multiplier be represented by a $p' = \lceil \frac{p+1}{k} \rceil$ digit polynomial in the balanced minimally redundant (Booth digit) system $P[2^k, \{-2^{k-1}, -2^{k-1}+1, \ldots, 2^{k-1}\}]$. Each term of the product

$$A \cdot B = \sum_{i=0}^{p'-1} A \cdot d_i \cdot 2^{ki}$$

is a higher radix partial product of the form

$$A \cdot d_i \cdot 2^{ki} = (-1)^s \cdot 2^e \cdot (jA), j \in \{1, 3, \ldots, 2^{k-1}-1\}.$$

Radix	*Primary Part. Prods* (jA)	# *Partial Products*	*PPG Fanin*	*Fanout Primary PP's*	*Total PPG Fanin*
2	1	64	1	64	64
4	1	33	1	33	33
8	2	22	2	22	44
16	4	17	4	17	68
32	8	13	8	13	124

Table 1. Complexity of partial product generation for various Booth recodings (p=64).

This allows each higher radix partial product to be created from a set of 2^{k-2} *primary* partial products (jA) by a conditional shift and/or complement. Five metrics are provided in Table 1 for comparing the consequences of employing higher radix Booth recodings on a 64-bit operand. The number of primary partial products that must be computed and routed to each partial product generator (PPG) grows linearly with the base β while the number of partial products that must be driven to each PPG decreases inversely with $lg\beta$. A measure of multiplier circuit routing complexity is the *total PPG-fanin* given by the sum of the number of primary partial products that must be routed into each PPG. The necessity of routing each primitive partial product (jA) to each of the PPG's causes the total PPG-fanin to grow for $\beta \geq 4$.

- Radix-4 has a clear advantage in these metrics over the host radix-2. The reduction by one half in the number of partial products and total PPG-fanin is obtained simply by the facility of the PPG units to conditionally complement and/or perform a one bit shift.
- Moving to radix-8 further reduces the number of partial products by a third while adding the complexity and delay of a 2-1 add to pre-compute the primitive value $(3A)$. This tradeoff is more acceptable for higher precisions in terms of adder complexity, but the routing of the two primitive values to each PPG increases the total PPG fanin.
- Moving to radices 16 and 32 only marginally reduces the number of partial products while greatly increasing the partial product complexity both in number of primary partial products and total fanin to the PPG's.

Our contribution here is a new multiplier recoding procedure that significantly reduces the total PPG-fanin while maintaining the number of higher radix partial products at a level between that of Booth recoding with radices 4 and 8.

3 Secondary Radix Operand Conversion

Consider a minimally redundant (Booth) radix polynomial representation $B = \sum_{i=0}^{p'-1} d_i 2^{ki}$ for very high k, in particular $5 \leq k \leq 12$. The digit ranges expand from $-16 \leq d_i \leq 16$ for $k = 5$ up to $-2048 \leq d_i \leq 2048$ for $k = 12$. To reduce the partial product complexity these large digit ranges are represented by a two to four digit number in a *secondary radix* γ where the non-zero digits $d \in D$ for the secondary radix system $P[\gamma, D]$ are exclusively restricted to signed binary powers $d = (-1)^s \cdot 2^n$ for $n = 0, 1, 2, \ldots$. The possibility of such systems will first be illustrated by a simple example.

Example 1. Our primary radix is $\beta = 32$ with Booth digit set $D = \{-16, -15, \ldots, 16\}$. Our secondary radix is $\gamma = 7$ with a digit set $D = \{-4, -2, -1, 0, 1, 2, 4\}$ having only signed binary power or zero digits. Note from Table 2 that every digit $-16 \leq d_i \leq 16$ of the primary radix system can be represented as a two digit radix-7 number

$$d_i = d_{i,1} \cdot 7 + d_{i,0}, \qquad d_{i,1}, d_{i,0} \in \{-4,-2,-1,0,1,2,4\}.$$

Our 64×64 bit product $A \cdot B$ using a secondary radix representation for B can be expressed as

$$A \cdot B = (7A) \cdot \sum_{i=0}^{12} (d_{i,1} 32^i) + (A) \cdot \sum_{i=0}^{12} (d_{i,0} 32^i). \quad (2)$$

The right hand side of (2) has 26 partial products, achieving a reduction more than halfway between that of Booth radix 4 and 8. These 26 partial products are partitioned into two groups, 13 of which employ the primary partial product $(7A)$ and 13 of which employ (A) giving a total PPG fanin of only 26 (table 5). Two options are possible with these simplified partial products noting that $d_{i,j} 32^i = (-1)^s 2^n$ or 0 for all $0 \leq i \leq 12, 0 \leq j \leq 1$.

- *Pre-compute (7A).* The primary partial product can be pre-computed by a shift and add $(7A = 8A - A)$ while the $d_{i,j}$ are obtained from a recoder or recoding table.
- *Post-compute (7A).* The higher order summation can utilize a 13:2 adder tree compressing $\sum_{i=0}^{12} A(d_{i,1} 32^i)$ to a redundant (e.g. carry save) sum z. Then the post computation can add $8z - z$ to the low order sum $y = \sum_{i=0}^{12} A(d_{i,0} 32^i)$ output from a second 13:2 adder tree. The value of $8z - z + y$ is completed by a 6:2 compressor and a 2-1 addition.

□

digit radix-32	$b[5i+4 : 5i-1]$	digits radix-7
0	111111,000000	00
1	000001,000010	01
2	000011,000100	02
3	000101,000110	$1\bar{4}$
4	000111,001000	04
5	001001,001010	$1\bar{2}$
6	001011,001100	$1\bar{1}$
7	001101,001110	10
8	001111,010000	11
9	010001,010010	12
10	010011,010100	$2\bar{4}$
11	010101,010110	14
12	010111,011000	$2\bar{2}$
13	011001,011010	$2\bar{1}$
14	011011,011100	20
15	011101,011110	21
16	011111	22

Table 2. Radix-7 representations and selection signals for all positive digits radix-32.

Note that the post-computation option of Example 1 utilizes only two more partial products and one additional level of 3-to-2 adder delay to avoid the complexity of a 2-1 adder to pre-compute $(7A)$. If multiplier digit recoding is performed in the first cycle of a pipelined multiplier, the post-computation option allows the product to be fed back as the multiplicand of a dependent multiply operation entering on the second cycle. This effectively reduces pipeline stall by one cycle on dependent multiplications.

The following shows that any primary radix system $\beta = 2^k$ with $D = \{-2^{k-1}, -2^{k-1}+1, \ldots, 2^{k-1}\}$ can be coupled with the secondary radix system $P[7, \{-4,-2,-1,0,1,2,4\}]$.

Lemma 1 *The radix system $P[7, \{-4,-2,-1,0,1,2,4\}]$ contains a unique radix polynomial P_B of value B for any integer B. Furthermore if $|B| \leq \frac{1}{3}(7^\ell - 1)$, then $P_B = (d_{\ell-1} d_{\ell-2} \ldots d_0)_7$ has at most ℓ digits.*

Proof: Given B let P'_B be the unique radix 7 polynomial with digits $-3 \leq d_i \leq 3$. Starting from the least digit d_0, replace any digit of magnitude 3 with a carry out of a unit of the same sign leaving behind a digit of magnitude 4 of the opposite sign. Sequentially, each carry is absorbed forming a range $[-4, 4]$ in each successive position before the next carry is rippled out for the magnitude 3 digit values. The process determines a suitable P_B which must be unique since the digit set is a non-redundant residue system modulo 7 [7]. Any positive initial value less than or equal to the ℓ digit number $(22 \ldots 2)_3$ will not have a positive carry out to

val	radix-11	val	radix-11	val	radix-11	val	radix-11
0	$0\,0\,0$	32	$1\,\overline{8}\,\overline{1}$	64	$2\,\overline{G}\,\overline{2}$	96	$0\,8\,8$
1	$0\,0\,1$	33	$1\,\overline{8}\,0$	65	$2\,\overline{G}\,\overline{1}$	97	$1\,\overline{2}\,\overline{2}$
2	$0\,0\,2$	34	$1\,\overline{8}\,1$	66	$2\,\overline{G}\,0$	98	$1\,\overline{2}\,\overline{1}$
3	$0\,1\,\overline{8}$	35	$1\,\overline{8}\,2$	67	$2\,\overline{G}\,1$	99	$1\,\overline{2}\,0$
4	$0\,0\,4$	36	$0\,4\,\overline{8}$	68	$2\,\overline{G}\,2$	100	$1\,\overline{2}\,1$
5	$0\,\overline{1}\,G$	37	$1\,\overline{8}\,4$	69	$1\,\overline{4}\,\overline{8}$	101	$1\,\overline{2}\,2$
6	$0\,2\,\overline{G}$	38	$0\,2\,G$	70	$2\,\overline{G}\,4$	102	$1\,\overline{1}\,\overline{8}$
7	$0\,1\,\overline{4}$	39	$\overline{1}\,G\,\overline{G}$	71	$\overline{1}\,G\,G$	103	$1\,\overline{2}\,4$
8	$0\,0\,8$	40	$0\,4\,\overline{4}$	72	$0\,8\,\overline{G}$	104	$0\,8\,G$
9	$0\,1\,\overline{2}$	41	$1\,\overline{8}\,8$	73	$1\,\overline{4}\,\overline{4}$	105	$1\,0\,\overline{G}$
10	$0\,1\,\overline{1}$	42	$0\,4\,\overline{2}$	74	$2\,\overline{G}\,8$	106	$1\,\overline{1}\,\overline{4}$
11	$0\,1\,0$	43	$0\,4\,\overline{1}$	75	$1\,\overline{4}\,\overline{2}$	107	$1\,\overline{2}\,8$
12	$0\,1\,1$	44	$0\,4\,0$	76	$1\,\overline{4}\,\overline{1}$	108	$1\,\overline{1}\,\overline{2}$
13	$0\,1\,2$	45	$0\,4\,1$	77	$1\,\overline{4}\,0$	109	$1\,\overline{1}\,\overline{1}$
14	$0\,2\,\overline{8}$	46	$0\,4\,2$	78	$1\,\overline{4}\,1$	110	$1\,\overline{1}\,0$
15	$0\,1\,4$	47	$\overline{1}\,G\,\overline{8}$	79	$1\,\overline{4}\,2$	111	$1\,\overline{1}\,1$
16	$0\,0\,G$	48	$0\,4\,4$	80	$0\,8\,\overline{8}$	112	$1\,\overline{1}\,2$
17	$1\,\overline{8}\,\overline{G}$	49	$1\,\overline{8}\,G$	81	$1\,\overline{4}\,4$	113	$1\,0\,\overline{8}$
18	$0\,2\,\overline{4}$	50	$2\,\overline{G}\,\overline{G}$	82	$2\,\overline{G}\,G$	114	$1\,\overline{1}\,4$
19	$0\,1\,8$	51	$\overline{1}\,G\,\overline{4}$	83	$1\,\overline{2}\,\overline{G}$	115	$1\,\overline{2}\,G$
20	$0\,2\,\overline{2}$	52	$0\,4\,8$	84	$0\,8\,\overline{4}$	116	$1\,1\,\overline{G}$
21	$0\,2\,\overline{1}$	53	$\overline{1}\,G\,\overline{2}$	85	$1\,\overline{4}\,8$	117	$1\,0\,\overline{4}$
22	$0\,2\,0$	54	$\overline{1}\,G\,\overline{1}$	86	$0\,8\,\overline{2}$	118	$1\,\overline{1}\,8$
23	$0\,2\,1$	55	$\overline{1}\,G\,0$	87	$0\,8\,\overline{1}$	119	$1\,0\,\overline{2}$
24	$0\,2\,2$	56	$\overline{1}\,G\,1$	88	$0\,8\,0$	120	$1\,0\,\overline{1}$
25	$1\,\overline{8}\,\overline{8}$	57	$\overline{1}\,G\,2$	89	$0\,8\,1$	121	$1\,0\,0$
26	$0\,2\,4$	58	$2\,\overline{G}\,\overline{8}$	90	$0\,8\,2$	122	$1\,0\,1$
27	$0\,1\,G$	59	$\overline{1}\,G\,4$	91	$1\,\overline{2}\,\overline{8}$	123	$1\,0\,2$
28	$0\,4\,\overline{G}$	60	$0\,4\,G$	92	$0\,8\,4$	124	$1\,1\,\overline{8}$
29	$1\,\overline{8}\,\overline{4}$	61	$1\,\overline{4}\,\overline{G}$	93	$1\,\overline{4}\,G$	125	$1\,0\,4$
30	$0\,2\,8$	62	$2\,\overline{G}\,\overline{4}$	94	$1\,\overline{1}\,\overline{G}$	126	$1\,\overline{1}\,G$
31	$1\,\overline{8}\,\overline{2}$	63	$\overline{1}\,G\,8$	95	$1\,\overline{2}\,\overline{4}$	127	$1\,2\,\overline{G}$
32	$1\,\overline{8}\,\overline{1}$	64	$2\,\overline{G}\,\overline{2}$	96	$0\,8\,8$	128	$1\,1\,\overline{4}$

Table 3. Radix-11 representations for all positive digits radix-256

position $\ell + 1$ by this process, and it follows that P_B is an ℓ digit number whenever $|B| \leq \frac{1}{3}(7^\ell - 1)$. □

Employing radix 7 requires numerous powers 7^i to be computed when the primary radix $\beta = 2^k$ has a large k. It is then useful to consider larger radices where all non zero digits are of the binary power form $(-1)^s 2^n$. Let $D_q^* = \{-2^q, -2^{q-1}, \ldots, -1, 0, 1, 2, 4, \ldots, 2^q\}$ be termed the *qth order binary power digit set*, with $P[2q+3, D_q^*]$ then a *binary power radix system*. Using theory on complete and unique radix representations developed in [7] the following can be shown for values $3 \leq \gamma \leq 59$.

Theorem 2 *The binary power radix system $P[\gamma, D_q^*]$ with $\gamma = 2q + 3$ contains a unique radix polynomial of value B for every integer B for any of the radices $\gamma = 3, 5, 7, 11, 13, 19, 23, 29, 37, 47$ and 59, and for no other radix less than 59.* □

Proof: Note that D_q^* is a symmetric complete residue system modulo $\gamma = 2q+3$ only for prime γ. Employing a Theorem from [7], $P[\gamma, D_q^*]$ will contain a unique radix polynomial of value i for every integer i if it contains a radix polynomial of value i for any i from zero up to the maximum digit magnitude 2^q divided by the base minus one $(\gamma - 1)$, that is for $0 \leq i \leq \frac{2^{q-1}}{q+1}$. This can be tested in time linear in $\frac{2^{q-1}}{q+1}$ by an algorithm described in [7]. Exhaustive use of this algorithm for each prime $3 \leq \gamma \leq 59$ can be employed to confirm the theorem. □

The three digit numbers of the radix 11 binary power radix system include all integers B with $|B| \leq 170$. Table 3 shows the representations for the range $0 \leq B \leq 128$. This provides that the primary radix 256 can be coupled with the secondary radix 11 using using only the secondary radix powers 1, 11 and 121.

Observation 3 *Let the p-bit integer $B = \sum_{i=0}^{p'-1} d_i \cdot 2^{ki}$, where $-2^{k-1} \leq d_i \leq 2^{k-1}$, with $p' = \lceil \frac{p+1}{k} \rceil$. Let $P[\gamma, D_q^*]$ be a binary power system with $\gamma = 2q + 3$ where $d_i = \sum_{j=0}^{p''-1} d_{i,j} \cdot \gamma^j$ for any $-2^{k-1} \leq d_i \leq 2^{k-1}$. Then the $p \times p$ bit product $A \cdot B$ is the sum of $p'' \cdot p'$ partial products. The partial products can be partitioned into p'' sets of p' binary power terms $(\gamma^j A)(d_{ij} 2^{ki})$, each term being zero or a conditionally complemented and/or shifted partial product of a single primitive partial product.* □

More simply stated the p-bit integer may be partitioned into p' high radix digits, each of which is represented by p'' secondary radix binary power digits. Each of the $p' \cdot p''$ terms needs only a single input primitive partial product to shift and conditionally complement to obtain the final partial product. This significantly simplifies the PPG's of a secondary radix system implementation for a high primary radix.

It is also possible to use secondary radix systems that are not "pure" radix systems. For example, the mixed radix system $d_{i,2} \cdot (5 \cdot 11) + d_{i,1} \cdot 11 + d_{i,0}$ with binary power digits provides a useful foundation for implementing radix-256 with some advantages over the radix-11 system of Table 3. A secondary radix system might also be simply a weighted system with binary power digits. It can be shown that the four weights 91, 15, 7 and 1 allow a secondary system with binary power digits that covers the integer range $[-2048, 2048]$. This secondary system can be coupled with the primary radix 2^{12} to achieve a 3-fold reduction in partial products.

The variety of secondary radix systems provides ample opportunities for new multiplier organizations as we shall show in the next section.

4 Multiplier Designs

In this section we propose multiplier designs on the basis of the encoding schemes from the previous section. The proposed encoding schemes share the following features, that will be utilized in the designs:

- There are very few digits to be considered in the secondary radix representations.
- All digits in the secondary radix representation are powers of two.
- All the multiples of all weights in the secondary radix system can be computed by simple sums.

If p'' denotes the number of digits that are required in the secondary radix representation of a higher radix digit, then $\lceil (p+1)/k \rceil \cdot p''$ digits are required to represent the multiplier $\langle b \rangle$. Thus, in comparison with binary the number of digits is reduced by roughly k/p'' which is $5/2, 8/3$ or 3 in the cases we consider. Additionally these digits are simple multiples, and even the multiplication by the weights can be computed by simple sums. This is not very different from the properties of Booth recoding radix-8. The main new flexibility for the implementations is given by the following properties:

- Each digit depends on only one odd multiple which could be 1 and is known at design time.
- Only some of the digits in the secondary radix have to be weighted by a 'hard' multiple.
- These 'hard' multiples are computed unconditionally, they do not depend on the value of the digits in the secondary radix.
- The low order digits do not have to deal with 'hard' multiples.

Based on these properties we suggest two basic architectures for the design of partial product generation and reduction:

- *Architecture I using pre-computed 'hard' multiples:* the multiplications by the weights of the secondary representation are computed on the multiplicand $\langle a \rangle$. In parallel the multiplier $\langle b \rangle$ is recoded and the partial products corresponding to the low order digits (which do not have to deal with any hard multiples) are generated and can already be partially reduced. These are combined with the remaining partial products in a second partial product reduction step.
- *Architecture II using post-computed 'hard' multiples:* After recoding the multiplier $\langle b \rangle$ into the digits of the secondary radix system, the multiples of the multiplicand $\langle a \rangle$ by these digits (note that these are only multiples by powers of two) are generated. The terms that we get from this selection are accumulated separately in groups that share the same weights of the corresponding digits in the secondary radix system. The carry-save representation of the sum of each of these groups is then multiplied by the corresponding weight (note that these multiples can be computed by simple sums). The results are accumulated to get the carry-save representation of the product in a final partial product reduction step.

We discuss the implementation of Architectures I and II for both radix-32 and radix-256 in the following.

4.1 Radix-32 Architecture of the Multiplier

The encoding scheme for the proposed implementations is based on a radix-32 signed digit representation of the multiplier:

$$\langle b \rangle = \sum_{i=0}^{p'-1} d_i \cdot 32^i,$$

so that the multiplier is represented by $p' = \lceil (p+1)/5 \rceil$ radix-32 digits $d_i \in \{-16, -15, \cdots, 16\}$. Corresponding to Booth recoding a canonical choice for the digits d_i is computed from the binary representation of $b = \langle b[p-1:0] \rangle$ by:

$$\begin{aligned} d_i \;=\; & -16b[5i+4] + 8b[5i+3] + 4b[5i+2] + \\ & + 2b[5i+1] + b[5i] + b[5i-1]. \end{aligned}$$

As suggested in the previous section each radix-32 digit d_i can be represented by two digits in the secondary radix-7:

$$d_i = d_{i,1} \cdot 7 + d_{i,0},$$

where both $d_{i,1}$ and $d_{i,0}$ are a power of two. The high order radix-7 digits $d_{i,1}$ can only have values from the set $\{-2, -1, 0, 1, 2\}$ and the low order radix-7 digits $d_{i,0}$ can only have the values $\{-4, -2, -1, 0, 1, 2, 4\}$ (see Table 2). Note that the selection from $\{-2, -1, 0, 1, 2\}$ times the multiplicand $\langle a \rangle$ compares to the simple selection for Booth digits radix-4. For the low order digits $d_{i,0}$ additionally $4 \times \langle a \rangle$ (a shift by 2 bit positions) has to be considered.

Without considering the weight of 7 this gives us two groups of $\lceil (p+1)/5 \rceil$ partial products, each of which can be generated very easily. For the group of partial products generated by the low order digits $d_{i,0}$ these are already the final values for the partial products.

The group of partial products generated by the high order digits $d_{i,1}$ additionally have to be multiplied by 7. There are two options where this multiplication could be computed: On one hand the multiplicand $\langle a \rangle$ can be multiplied by 7

before the partial product reduction which leads to Architecture I. Figure 1c) depicts a block diagram corresponding to this implementation radix-32 using the pre-computation of the $7 \times \langle a \rangle$ multiple. The multiplication of $\langle a \rangle$ by 7 is computed by the following sum:

$$\begin{aligned} 7 \cdot \langle a \rangle &= 8 \cdot \langle a \rangle - \langle a \rangle \\ &= \langle a[p-1:0], 00 \rangle + \langle 11, \overline{a[p-1:0]} \rangle + 1. \end{aligned}$$

On the other hand the group of partial products generated by the high order digits could be multiplied by 7 after these partial products already have been compressed to a carry-save representation which corresponds to Architecture II. Figure 1e) depicts a block diagram corresponding to this implementation radix-32 using the post-computation of the $7x$ multiple. Also in this case the formula $7 \cdot x = 8 \cdot x - x$ is used to compute the $7x$ multiple, but this time it is not computed using $\langle a \rangle$, but it is computed using the carry-save representation of the sum of the terms that have been generated by the high order digits. In this way the 2 partial products from the carry-save representations of the two groups are extended to 6 partial products, which are then reduced to the carry-save representation of the product in a final 6:2 reduction step. Note, that for the implementation of Architecture II the input of the multiplicand $\langle a \rangle$ is required later than the input of the multiplier $\langle b \rangle$. With the partitioning suggested in Figure 1e) this makes a difference of a whole cycle in which the second operand is not needed. An operand that is fed back from a multiplier result only requires one cycle in the partial product generation and reduction for this proposed partitioning.

4.2 Radix-256 Architecture of the Multiplier

Although there are several choices to represent signed radix-256 digits from the previous section, we only propose multiplier designs using fixed radix-11 representations in some detail. We then briefly discuss alternative choices using mixed radices and other weighting systems.

The encoding scheme for the proposed implementations is based on a radix-256 signed digit representation of the multiplier:

$$\langle b \rangle = \sum_{i=0}^{p'-1} d_i \cdot 256^i,$$

so that the multiplier is represented by $p' = \lceil (p+1)/8 \rceil$ radix-256 digits $d_i \in \{-128, -127, \cdots, 128\}$. As previously suggested each radix-256 digit d_i can be represented by three digits in the secondary radix-11:

$$d_i = d_{i,2} \cdot 121 + d_{i,1} \cdot 11 + d_{i,0},$$

where each of the digits $d_{i,2}$, $d_{i,1}$ and $d_{i,0}$ in the secondary radix representation is a power of two (see Table 3 where a digit with the value of 16 is represented by the letter G). The high order radix-11 digits $d_{i,2}$ can only have values from the set $\{-4, -2, -1, 0, 1, 2, 4\}$ and the middle and the low order radix-11 digits $d_{i,1}$ and $d_{i,0}$ can only have the values $\{-16, -8, -4, -2, -1, 0, 1, 2, 4, 8, 16\}$. In this case the partial products corresponding to the high order digits and the middle digits have to be weighted by 121 and 11 respectively. Again we consider the two options of either pre-computing these multiples corresponding to Architecture I or post-computing the multiples corresponding to Architecture II. The block diagram for the pre-computing version is depicted in Figure 1d) and the block diagram for the post-computing version is depicted in Figure 1f). For the computation of the hard multiples $121x$ and $11x$ we use the formula $121 = 128 - 8 + 1$ and $11 = 8 + 2 + 1$ in this case, which can be implemented with the combination of a carry-save and a carry-propagate adder for Architecture I. For Architecture II the post-computation extends the 6 terms from the three carry-save representations of the three groups to 14 terms, which then have to be reduced to a carry-save representation in a last step.

Again the main differences between the two architectures is that in Architecture I part of the partial product reduction can be started early in parallel to the generation of the hard multiples for the other partial products. In Architecture II the second operand is required later than the first operand. The three parts of the recoding, the first group-wise partial product reduction step and the second partial product reduction step suggest a partitioning of Architecture II into three pipeline stages as depicted in Figure 1 by the two dotted lines. Corresponding to the later delay analysis the three pipeline stages seem to be balanced in this design. Such a partitioning reduces the delay that is required between latches. Because the second operand only has to go through two of these stages, there is the potential for a fast feedback of a result for dependent multiplications.

Alternatively to representing the signed radix-256 digits with the fixed secondary radix-11, one could also choose a mixed radix representation for the signed digits d_i as suggested in the previous section. For example also in the mixed secondary radix system 55-11-1, where the three digits $d_{i,2}$, $d_{i,1}$ and $d_{i,0}$ have the weights 55, 11 and 1, all digits can be chosen to be either a power of two or zero. It would be the main advantage of the of this mixed radix system that there are several choices of $d_{i,2}$, $d_{i,1}$ and $d_{i,0}$ for the representation of most d_i. This could be used to optimize the implementation of the recoder. Moreover, the digit set for the middle digit could be reduced to the values $\{-8, -2, -1, 0, 1, 2, 8\}$ by an appropriate choice of encodings.

There are even more choices for the three weights in the secondary representation to consider for multiplication radix-256. Just to mention a few promising candidates: $11-7-1, 13-7-1, 29-9-1, 13-11-1, 53-11-1$

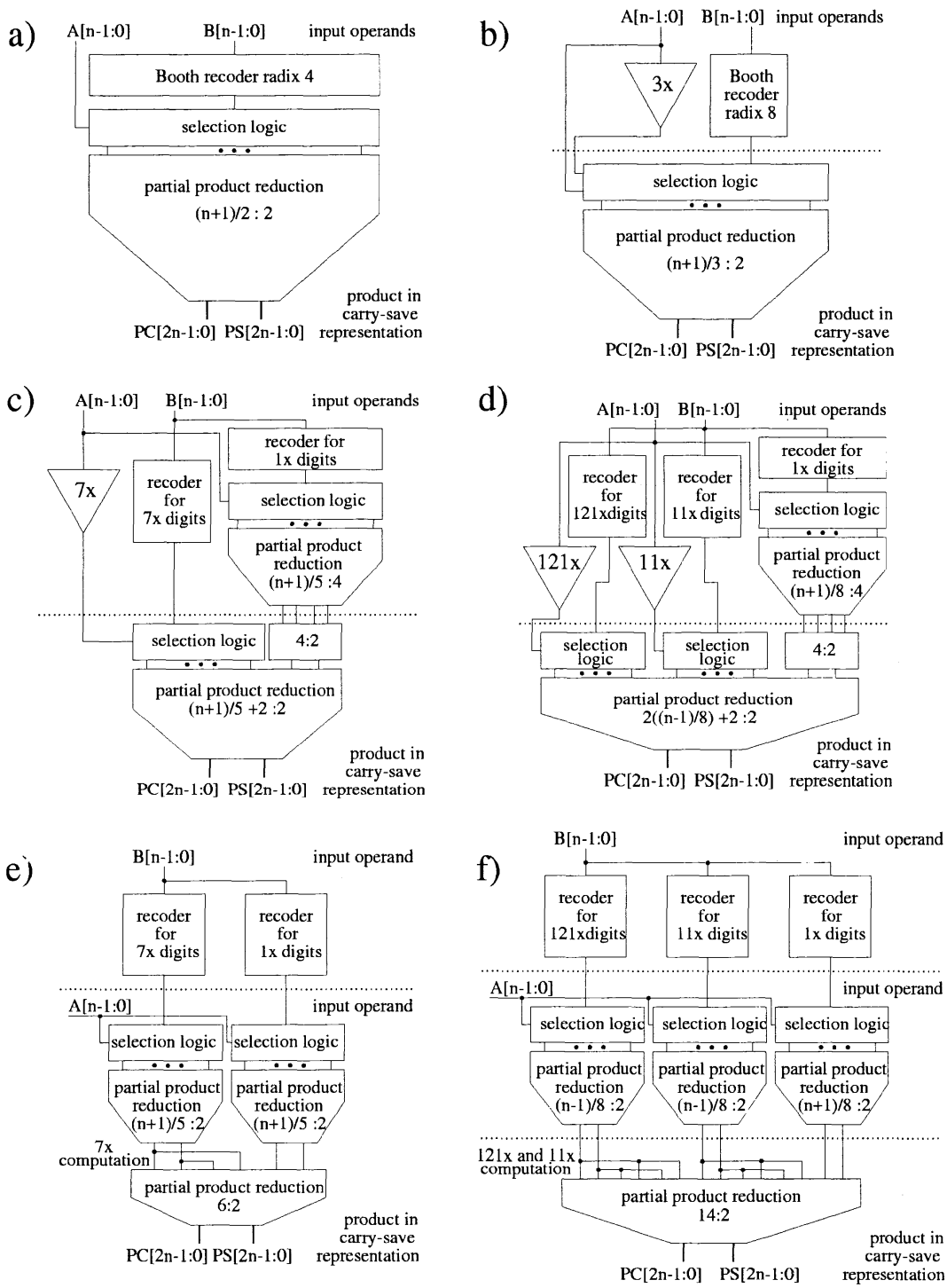

Figure 1. Block diagrams of partial product generation and reduction for multiplier designs using: a) conventional Booth recoding radix-4; b) conventional Booth recoding radix-8; c) radix-32 encoding with pre-computation of $7x$; d) radix-256 encoding with pre-computation of $11x$ and $121x$; e) radix-32 encoding with post-computation of $7x$; f) radix-256 encoding with post-computation of $11x$ and $121x$.

design	p=24 bit				p=32 bit				p=53 bit				p= 64 bit				p= 113 bit			
	# cycles 1st operand	# cycles 2nd operand	throughput / cycle	delay between latches	# cycles 1st operand	# cycles 2nd operand	throughput / cycle	delay between latches	# cycles 1st operand	# cycles 2nd operand	throughput / cycle	delay between latches	# cycles 1st operand	# cycles 2nd operand	throughput / cycle	delay between latches	# cycles 1st operand	# cycles 2nd operand	throughput / cycle	delay between latches
a) Booth radix-4	1	1	1	13LL	1	1	1	14LL	1	1	1	16 LL	1	1	1	17LL	1	1	1	19LL
b) Booth radix-8	2	2	1	9LL	2	2	1	10LL	2	2	1	13 LL	2	2	1	13LL	2	2	1	16LL
c) radix-32 pre	2	2	1	9LL	2	2	1	10LL	2	2	1	12 LL	2	2	1	12LL	2	2	1	15LL
d) radix-256 pre	2	2	1	9LL	2	2	1	11LL	2	2	1	12 LL	2	2	1	13LL	2	2	1	15LL
e) radix-32 post	2	1	1	12LL	2	1	1	13LL	2	1	1	15 LL	2	1	1	16LL	2	1	1	18LL
f) radix-256 post	3	2	1	9LL	3	2	1	9LL	3	2	1	9LL	3	2	1	9LL	3	2	1	11LL

Table 4. Latencies, cycle times and throughput for the multiplier designs.

and $77 - 11 - 1$, all allow secondary digits that are a power of two and that are smaller than or equal to 16. There are some differences in the properties of these encodings for the multiplier implementations. One advantage of the systems using 7 or 9 as one of their weights is that these multiples can be computed by a sum of 2 terms instead of a sum of 3 terms. For Architecture II this would reduce the reduction in the third stage from a $14 : 2$ reduction to a $12 : 2$ reduction, which is a significant change regarding the delay of the corresponding adder tree.

4.3 Delay Analysis and Comparison

In this section we analyze the delays of the proposed designs and compare the implementations with the partial product generation and reduction using conventional Booth recoding radix-4 and radix-8. In the delay analysis we mainly look at the implementation at gate level and count the delay on the critical paths of the implementations in logic levels.

We assume that a full adder has a delay of 2 logic levels and a 4-to-2 adder is implemented with a delay of 3 logic levels using an optimized version as described in [6, 14]. We assume the selection of the partial products to have a delay of 2 logic levels, which is mainly one logic level to select the absolute (shifted) values of the partial products and one logic level to conditionally complement it.

The recoding hardware is assumed to have a delay of 2 logic levels for Booth recoding radix-4 [1, 2, 9, 13], to have a delay of 3 logic levels for Booth recoding radix-8 [1, 2], to have a delay of 4 logic levels for our radix-32 encodings and to have a delay of 6 logic levels for our radix-256 encodings. We assume the carry propagate addition of 64 bit numbers to be implemented in 8 logic levels. Based on these basic delay assumptions we suggest the partitionings of the implementations into pipeline stages as shown in Figure 1a)-f) by the dotted lines. Tables 4 then summarizes the latency from both the input of the first and the second operand in pipeline stages and the number of logic levels that are required between latches in a pipeline stage for each design.

For implementations using conventional Booth recoding [3, 10] radix-4 the partial product generation and reduction is usually not partitioned, but the implementation is supposed to fit into one pipeline stage (see Figure 1a). For large operand sizes this can increase the delay of a pipeline stage between latches to up to 19 logic levels(LL) like for $p = 113$. Also the area of the implementation might become very large with the wire lengths causing additional delay. The fanout for the selection logic for this implementation is about $n/2$.

For implementations using conventional Booth recoding radix-8 the implementation is split into two pipeline stages as depicted in Figure 1b) by the dotted line [1, 2, 8, 11]. The hard multiple has to be computed on the multiplicand and must be known for the partial product reduction. The delay is reduced in comparison to Booth radix-4 by eliminating the recoding from the second cycle and by reducing the number of partial products to roughly $n/3$ providing a smaller and faster partial product reduction. The gate delay analysis suggests that the delay between latches is reduced by $3 - 4$ logic levels depending on the operand size. Because either the multiplicand or $3\times$ the multiplicand could occur in each partial product the maximum fanout in this implementation is about $n/3$.

In contrast to the designs using Booth recoding radix-8, the proposed designs for Architecture I reduce some partial products in parallel to the computation of the hard multiples during the first stage. This decreases the number of partial products that have to be reduced in the second stage. The amount of work which can be completed in the first stage depends on the delay of the digit recoder for the low order digits. The partitioning of the implementation into two

Radix	# Primary PP's (jA)	# Partial Products	PPG Fanin	Total PPG Fanin	Fanout Primary PP's (jA)
32pre	2	26	1	26	13
32post	1	26	1	26	26
256pre	3	24	1	24	8
256post	1	24	1	24	24
4096pre	4	22	1	22	6
4096post	1	22	1	22	22

Table 5. Complexity of partial product generation for proposed designs (p=64).

stages should be chosen so that the delay that is required in the first and the second stage is balanced. To balance the delays of these two stages, part of the reduction in the first stage could be shifted to the second stage. Our delay analysis suggested that in many cases a 4:2 reduction step should be computed at the beginning of the second stage in parallel with the generation of the partial products for the higher digits as depicted in Figure 1c)-d). In some of the designs exchanging this 4:2 compression to a 3:2 compression and the shifting of the 3:2 compression step to the first stage is feasible and has been used to balance the designs.

The delay analysis from Table 4 suggests that when only looking at gate level the proposed designs have shorter pipeline stages than the designs that use Booth recoding radix-8 by about $1-2$ logic levels.

Additionally the block sizes of the partial product reductions are reduced by separating the generations for the groups that have a different weight in the secondary radix. Since the input for the partial product generation (selection logic) is a single, different multiple of the multiplicand for each of the digit groups, each of these multiples only has a fanout of n/k, which is the maximum fanout of the implementation for Architecture I. Comparing with the maximum fanout of $n/3$ for Booth radix-8, our radix-32 implementation has a maximum fanout of only $n/5$ and our radix-256 implementation has a maximum fanout of only $n/8$. For the smaller group-wise partial product reductions the connection lengths for these inputs are reduced allowing implementations that have less wire delay or lower power consumption. Table 5 gives an overview of the number of primary partial products, the number of partial products, the PPG fanin, the total PPG fanin and the maximum fanout for primary PPG's for our proposed designs for $p = 64$. A comparison with Table 1 shows the differences to multipliers using conventional Booth recoding.

The main difference of the proposed designs is that in Architecture II, the multiplicand is required one stage later than the multiplier as an input. With the partitionings suggested in Figures 1e)-f), the latency of dependent multiplications in the proposed radix-32 design compares with the latency of Booth recoding radix-4 and the latency of the proposed radix-256 design compares with the latency of Booth recoding radix-8 in this way.

The delay analysis for our radix-32 implementation indicates slightly faster pipeline stages that when using Booth recoding radix-4, also the size of the partial product reduction is smaller and the maximum fanout is reduced from $n/2$ to $2n/5$. In this case the pipeline stages of our design are not very balanced, and there is much more time available in the first stage than is needed for the recoding implementation. This implementation would suggest a partitioning into three stages: 1/2 stage for recoding, a full stage for the first partial product reduction and 1/2 stage for the second partial product reduction.

The proposed radix-256 implementation can nicely be partitioned into three balanced pipeline stages. The delay analysis for the stages of this implementation are much shorter than for Booth recoding radix-8. Still the latency for the second operand is two stages. In this case the maximum fanout is about $3n/8$. In the last stage 14 partial products must be reduced when using the fixed radix-11 for the secondary radix representation. When choosing some of the alternative weights for the secondary number system that we previously suggested for radix-256, these 14 terms could be reduced to 12.

Often a difference of one partial product can significantly change the delay of adder tree reduction [5, 12]. Because the proposed designs provide some flexibility to shift parts of the reductions from one stage to another, optimizations based on the boundaries for the number of partial products for a given delay can be utilized. Additionally the alternative choices for the radix-256 and radix-4096 representation suggested in this section provide even more flexibility.

5 Conclusions and Further Work

We propose implementations for high-radix multiplication of radix-32, radix-256 and also point out options for multiplication radix-4096. The multiplier designs are targeting the implementation in deeper pipelines with shorter pipeline stages and smaller fanout. The basis for our implementations is the introduction of novel representation schemes for the recoding of the multiplier. We represent each conventional 'high radix' Booth digit in a secondary radix system with digits that are powers of 2.

This proposed encoding scheme significantly reduces the number of hard multiples typically required for higher radix multiplication using conventional Booth recoding. It permits the implementation of radix-32 multiplication with only one hard multiple calculation; radix-256 multiplication with two hard multiples; and radix-4096 with three hard multiples. Furthermore, the partial products depending on

these hard multiples are known in advance reducing the fanout requirements for these multiples and providing greater design flexibility to tailor the multiplier to the specific implementation constraints.

We demonstrated several designs of different radices where the hard multiples were calculated in parallel to the recoding (pre-computation) as typically done today, or calculated in redundant form (post-computation) after the partial product reduction creating a new paradigm.

Each design demonstrated a different balance of work and may be practical depending on the time budget of each pipeline stage. The radix-256 post-computation multiplier did particularly well under our implementation constraints of balanced pipeline stages. They allow for a smaller cycle time than conventional Booth recoding radix-8, and still maintain the two cycle feedback latency for one operand.

Our designs also reduce the maximum fanout of p-bit multiplication to $p/5$ for radix-32 and $p/8$ for radix-256. Note that this fanout for Booth recoding radix-4 is $p/2$ and for Booth recoding radix-8 is $p/3$.

Further investigation for base and digit selection of the secondary radix is continuing in our lab. Various bases are possible and may provide an easier calculation of the hard multiples. Additionally, redundancy in the binary power digit set of the secondary radix can improve the recoding logic and provide fewer shifting alternatives for the shifting multiplexors. It would also be of interest to compare the design from [4] using half radix-4 and half radix-8 digits with our radix-32 system. Our design avoids the necessity of pre-computing $3x$ along with the higher PPG fanin, while requiring a more complex recoder. Optimizing a recoder design for our secondary radix systems is a goal that would allow a better comparison of these mixed radix systems.

References

[1] H. Al-Twaijry. *Area and Performance Optimized CMOS Multipliers*. PhD thesis, Stanford University, August 1997.

[2] G. Bewick. *Fast Multiplication: Algorithms and Implementation.* PhD thesis, Stanford University, March 1994.

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

[4] J. Clouser, M. Matson, R. Badeau, R. Dupcak, S. Samudrala, R. Allmon, and N. Fairbanks. A 600-MHz superscalar floating-point processor. *IEEE Journal of Solid-state circuits*, 32(7):1026–1029, 1999.

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

[6] D. Galhi and A. Chan. Four-to-two adder cell for parallel multiplication. U.S. patent 4901270, Intel Corporation, 1990.

[7] D. Matula. Basic digit sets for radix representations. *Journal of the ACM*, 29(4):1131–1143, 1982.

[8] S. Oberman. Floating point division and square root algorithms and implementation in the AMD-K7^{TM} microprocessor. *Proceedings of the 14th Intl. Symposium on Computer Arithmetic*, 14:106–115, 1999.

[9] W. Paul and P.-M. Seidel. On the Complexity of Booth Recoding. *Proceedings of the 3rd Conference on Real Numbers and Computers(RNC3)*, pages 199–218, 1998.

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

[11] E. Schwarz, R. Averill, and L. Sigal. A radix-8 CMOS S/390 multiplier. *Proceedings of the 13th Intl. Symposium on Computer Arithmetic*, 13:2–9, 1997.

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

[13] W.-C. Yeh and C.-W. Jen. High-speed booth encoded parallel multiplier design. *IEEE Transactions on Computers, Special Issue on Computer Arithmetic*, 49(7):692–701, 2000.

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