This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCSII.2014.2387620, IEEE Transactions on Circuits and Systems II: Express Briefs

Radix-2^r Arithmetic for Multiplication by a Constant: Further Results and Improvements

Abdelkrim K. Oudjida, Nicolas Chaillet, and Mohamed L. Berrandjia

***Abstract*—In a previous work we proposed a new sublinear-runtime recoding heuristic for the multiplication by a constant, accompanied by its upper-bound complexity. In this brief, further results are provided, namely, the analytic expressions of the average number of additions and the maximum adder-depth. Improvements to the proposed heuristic are considered as well, using a redundant recoding followed by a common-digit-elimination step.**

***Index Terms*— High-Speed and Low-Power Design, Linear-Time-Invariant (LTI) Systems, Multiplierless Single/Mutiple Constant Multiplication (SCM/MCM), Radix-2^r Arithmetic.**

I. Background and Motivation

Based on the radix-2^r arithmetic, we introduced in the preceding work [1] a new sublinear-runtime recoding heuristic (RADIX-2^r) for the multiplication by a constant with an upper-bound equal to $\lceil (N+1)/r + 2^{r-2} - 2 \rceil$, where, N is the constant bit-length, $r = 2 \cdot W\left(\sqrt{(N+1)\cdot log(2)}\right)/log(2)$, W and $\lceil\ \rceil$ are the Lambert and ceiling functions, respectively. We obtained the currently best known proved upper-bound on the exact number of additions for SCM. While RADIX-2^r shows a clear superiority over digit-recoding algorithms (CSD [2] and DBNS [3]), the comparison to non-digit-recoding algorithms (Bernstein [4], Lefèvre [5], BHM [6], Hcub [7], and MAG [8]) exhibits mitigated results. Non-recoding algorithms are better than RADIX-2^r when considering the average (*Avg*) number of additions, but not necessarily better regarding the maximum number of additions (*Upb*). Thus, we came to a *significant conclusion*: a lower *Avg* does not guarantee a lower *Upb*.

Avg, *Upb*, and adder-depth (*Ath*) are the most commonly used metrics in SCM/MCM. *Avg* informs on the compression performance of the heuristic. For a nonnegative N-bit constant, *Avg* is calculated as the mean number of additions for values varying from 0 to 2^N-1. Whereas *Upb* denotes the worst case in number of additions, as for each heuristic there exists a specific set of constants that are hard to compress. *Ath* is rather a measure of the critical path in number of cascaded adders. Reducing *Ath* not only improves the speed, but decreases the power consumption as well [9].

Developing a *predictable* heuristic, that is, with known *Avg*, *Upb,* and *Ath* complexities, gives a precise idea on how the heuristic evolves with respect to the size N. This much helps to decide early in the design process whether a given heuristic can fit one's specification requirements. To our knowledge, among all existing heuristics only CSD and RADIX-2^r are predictable. While both *Avg* and *Upb* complexities are known for CSD, only *Upb* is known so far for RADIX-2^r [1].

A.K. Oudjida (a_oudjida@cdta.dz) and M.L. Berrandjia are with "Centre de Développement des Technologies Avancées", CDTA, Cité du 20 août 1956, Baba-Hassen, Algiers, Algeria. N. Chaillet (nicolas.chaillet@femto-st.fr) is with FEMTO-ST Institute, UFC/CNRS/ENSMM/UTBM, 32 avenue de l'Observatoire, 25044 Besançon, Cedex, France.

The main purpose of this work is to make RADIX-2^r a fully predictable heuristic. In addition to *Upb*, we determine the analytic expressions for *Avg* and *Ath*. We also provide the theoretical background showing that the R3 algorithm [10] is a variant of RADIX-2^r with an improved *Avg* and the same *Upb* and *Ath*.

This brief is organized as follows. Section I outlines the necessity for a fully-predictable heuristic. RADIX-2^r *Avg* and *Ath* are introduced in Sections II and III, respectively. Section IV treats the overflow safety in the fixed-point representation, while Section V shows how RADIX-2^r can be improved using a redundant recoding. Finally, Section V provides some concluding remarks and suggestions for future work.

II. Radix-2^r: Average Number of Additions (*Avg*)

A nonnegative N-bit constant C is expressed in radix-2^r as

$$C = \sum_{j=0}^{(N+1)/r-1} \left(c_{rj-1} + 2^0 c_{rj} + 2^1 c_{rj+1} + 2^2 c_{rj+2} + \cdots + 2^{r-2} c_{rj+r-2} - 2^{r-1} c_{rj+r-1}\right) \times 2^{rj} = \sum_{j=0}^{(N+1)/r-1} Q_j \times 2^{rj}, \quad (1)$$

where $c_{-1} = c_N = 0$ and $r \in \mathrm{N}^*$. In (1), the two's complement representation of C is split into $\lceil (N+1)/r \rceil$ slices (Q_j), each of r+1 bit length. Each pair of two contiguous slices has one overlapping bit. A digit-set $DS(2^r)$ corresponds to (1), such as

$$Q_j \in DS(2^r) = \{-2^{r-1}, -2^{r-1}+1, \ldots, -1, 0, 1, \ldots, 2^{r-1}-1, 2^{r-1}\}.$$

The sign of the Q_j term is given by the c_{rj+r-1} bit, and $|Q_j| = 2^{k_j} \times m_j$, with $k_j \in \{0,1,2,\ldots,r-1\}$ and $m_j \in OM(2^r) \cup \{0,1\}$, where $OM(2^r) = \{3,5,7,\ldots,2^{r-1}-1\}$. $OM(2^r)$ is the set of odd positive digits in radix-2^r recoding, with $|OM(2^r)| = 2^{r-2}-1$.

Since each slice Q_j comprises r+1 bits, the total number of the different bit-combinations is 2^{r+1}. According to (1), only two combinations produce $Q_j = 0$: in case all the r+1 bits are equal to "0" or "1". Hence, the average number of non-null Q_j terms is equal to $(2^{r+1}-2)/2^{r+1} = 1 - 2^{-r}$. Each $Q_j \neq 0$ generates one partial product (PP). Thus, the average number of PPs in the $\lceil (N+1)/r \rceil$ slices is: $Avg_{pp} = (1-2^{-r}) \times \lceil (N+1)/r \rceil$.

For each $m_j \in OM(2^r)$ there exists an integer $k \in \{1, 2, \ldots, |OM(2^r)|\}$, such as $m_j = 2 \times k + 1$. To set the correspondence between j and k, m_j is denoted m_{jk}. The number of occurrences (O_{cc}) of m_{jk} among the 2^{r+1} combinations of Q_j is

$$O_{cc}(m_{jk}) = 4 \times \log_2 \left\lceil \frac{2^{r-1}}{2 \times k + 1} \right\rceil. \quad (2)$$

The factor 4 in (2) is due to the fact that each occurrence of m_{jk} in the positive and negative part of $DS(2^r)$ is double (see

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCSII.2014.2387620, IEEE Transactions on Circuits and Systems II: Express Briefs

Table VI in [1]). The reason is that the c_{rj-1} and c_{rj} bits in (1) have the same influence $\left(c_{rj-1}\times 2^0 + c_{rj}\times 2^0+\cdots\right)$ on the Q_j term. Therefore, the probability (P) that m_{jk} occurs among 2^{r+1} combinations is $P(m_{jk})=O_{cc}(m_{jk})/2^{r+1}$. We deliberately employ "probability" instead of the "average" to facilitate the demonstration, but actually the two notions have the same meaning. Now, the probability that m_{jk} occurs in the slice Q_j knowing that it has *not* occurred in the slices preceding the slice j is (Bayes's theorem):

$$P(m_{jk}/j)=\frac{P(m_{jk}\cap j)}{P(j)}=\frac{P(m_{jk})\times[1-P(m_{jk})]^j}{1}=P(m_{jk})\times[1-P(m_{jk})]^j.$$

The probability that any$(\forall)$ m_{jk}, for $k=1..\left|OM(2^r)\right|$, occurs in the slice Q_j knowing that it has not occurred in the slices preceding the slice j is $P(\forall m_{jk}/j)=\sum_{k=1}^{\left|OM(2^r)\right|}P(m_{jk}/j)$. Note that the $P(m_{jk}/j)$ are mutually exclusive, since one and only one odd-digit (m_{jk}) occurs in the slice j. Consequently, the average number of generated odd-digits considering all slices is

$$Avg_{om}=\sum_{j=0}^{\lceil (N+1)/r\rceil-1}P(\forall m_{jk}/j).$$

Hence, the average number of additions for RADIX-2^r is

$$Avg\geq -1+Avg_{pp}+Avg_{om} \qquad (3)$$

$$\geq -1+\left(1-2^{-r}\right)\times\lceil (N+1)/r\rceil+\sum_{j=0}^{\lceil (N+1)/r\rceil-1}\left\{\sum_{k=1}^{2^{r-2}-1}P(m_{jk})\times\left[1-P(m_{jk})\right]^j\right\}.$$

Avg_{om} does not take into account the fact that for r>4 some odd-digits require more than one addition. For instance, the digit 11 requires 2 additions. But if the digit 3 occurs in the *same* recoding, 11 will need just one addition since $11=2^3+3$. However, we proved in [1] that $Avg_{om}\leq 2^{r-2}-1$ (see Theorem (1) in [1]). Consequently, we can say that Avg is bounded by

$$-1+Avg_{pp}+Avg_{om}\leq\ Avg\ \leq -2+Avg_{pp}+2^{r-2}$$

We also proved in [1] that to get the minimum number of additions (Upb), r must be equal to

$$r=2\cdot W\left(\sqrt{(N+1)\cdot log(2)}\right)/\,log(2), \qquad (4)$$

where W is the Lambert function.

Using the two Avg limits, we have bounded the average for N varying from 64 to 8192. Results are reported in Table I. It has to be noted that for r≤4, $Avg=-1+Avg_{pp}+Avg_{om}$.

We observe that for RADIX-2^r, Avg is very close to Upb. The reason is that the average of the null Q_j digits is very low: $Avg(\forall Q_j=0)=\frac{2}{2^{r+1}}\times\lceil (N+1)/r\rceil=\frac{\lceil (N+1)/r\rceil}{2^r}$. Note that RADIX-$2^r$ provides 50% saving over CSD in Avg for N=1134.

Theorem (1) in [1] allows building the entire set of odd-digits in just r–2 stages of cascaded additions. Since there are $\lceil (N+1)/r\rceil$ slices, the total number of cascaded adders is

$$Ath=\lceil (N+1)/r\rceil-1+r-2=\lceil (N+1)/r\rceil+r-3 \qquad (5)$$

Based on the values of r given by (4), we have calculated Ath and grouped the results in Table I. For a serial implementation (adders connected in series), a saving of slightly more than 50% over CSD is achieved at N=64. While for a parallel implementation based on a tree structure, CSD Ath is lower than RADIX-2^r Ath for any value of N≥24. As for $Upb=\lceil (N+1)/r\rceil+2^{r-2}-2$, 50% saving is attained at N=128.

III. RADIX-2^r: A LOWER ADDER-DEPTH (Ath)

Equation (4) ensures a minimum Upb, whereas lower Ath values are still possible. Any value of r, such as $r<2\cdot W\left(\sqrt{(N+1)\cdot log(2)}\right)/\,log(2)$ produces both higher Upb and Ath. While the opposite, that is, $r>2\cdot W\left(\sqrt{(N+1)\cdot log(2)}\right)/log(2)$ leads to a lower Ath but a higher Upb. To garantee a reasonable balance, we set as a condition that the entire number of odd-digits must be less or equal than the total number of slices

$$\left(\left|OM(2^r)\right|\leq\lceil (N+1)/r\rceil\right). \qquad (6)$$

This condition avoids generating more odd-digits ($2^{r-2}-1$) than it is actually invoked by the recoding process. Thus, solving (6), a balanced solution for a lower Ath is found with

$$r=W(4.(N+1).\ log(2))/\,log(2). \qquad (7)$$

Table II indicates the values of r that yield a lower $Ath,$ along with its corresponding Upb and Avg. Note that both (7) and (4) provide exactly the same results for N≤20, either in Ath, Upb, or Avg. Starting from N≥21, lower Ath are obtained using (7) but at the expense of higher Upb and Avg as indicated by Table I and II. For instance, for N=256 equation

TABLE I

RADIX-2^r VERSUS CSD: *Avg*, *Ath*, and *Upb* FOR AN *N*-BIT CONSTANT

N			8	16	32*	64*	128	256	512	1024	2048	4096	8192
r			3	3	4	5	5	6	6	7	8	8	9
Avg	RADIX-2^r	min	1.86	4.51	8.96	16.44	30.37	54.00	98.11	174.19	313.43	572.41	1033.38
		max				18.59	31.18	56.32	98.65	175.85	317.99	572.99	1035.22
	CSD		2.11	4.77	10.11	20.77	42.11	84.77	170.11	340.77	682.11	1364.77	2730.11
	Saving (%)		1.19	5.45	11.37	15.69	26.92	34.92	42.16	48.63	53.71	58.03	62.11
Ath	RADIX-2^r	…	3	6	10	15	28	46	89	151	262	518	917
		//	3	4	6	7	8	10	11	13	15	16	17
	CSD	…	4	8	16	32	64	128	256	512	1024	2048	4096
		//	3	4	5	6	7	8	9	10	11	12	13
	Saving (%)	…	25.00	25.00	37.50	53.12	56.25	64.06	65.23	70.50	74.41	74.70	77.61
		//	00.00	00.00	–20.00	–16.66	–14.28	–25.00	–22.22	–30.00	–36.36	–33.33	–30.76
Upb	RADIX-2^r		3	6	11	19	32	57	100	177	319	575	1037
	CSD		4	8	16	32	64	128	256	512	1024	2048	4096
	Saving (%)		25.00	25.00	31.25	40.62	50.00	55.46	60.93	65.42	68.84	71.92	74.68

N is the bit-size of a nonnegative constant; $r=2\cdot W\left(\sqrt{(N+1)\cdot log(2)}\right)/log(2)$. For N≥64, the saving in Avg is calculated considering (min+max)/2.
*: For N=32, both r=3 and r=4 produce the same Upb, but r=4 yields lower Ath. The same holds true for N=64 with r=4 and r=5.
…: Serial implementation (adders connected in series); //: Parallel implementation based on a tree structure. For RADIX-2^r, $Ath^{\cdots}=\lceil (N+1)/r\rceil+r-3$, and $Ath^{//}=\lceil log_2\lceil (N+1)/r\rceil\rceil+r-2$. For CSD, $Avg=(N+1)/3-8/9$, $Upb=\lceil (N+1)/2\rceil-1$, $Ath^{\cdots}=\lceil (N+1)/2\rceil-1$, and CSD $Ath^{//}=\lceil log_2\lceil (N+1)/2\rceil\rceil$.
Erratum: In [1], we took CSD $Avg=(N/3)-8/9$, which is the average of a two's complement N-bit constant (see the proof in [11]).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCSII.2014.2387620, IEEE Transactions on Circuits and Systems II: Express Briefs

TABLE II

Ath, *Upb*, *Avg*, AND *r* VALUES FOR AN *N*-BIT CONSTANT USING RADIX-2^r

N	8	16	32	64	128	256	512	1024	2048	4096	8192
r	3	3	5	5	6	7	8	8	9	10	11
Ath…	3	6	9	15	25	41	70	134	234	417	753
Upb	3	6	13	19	36	67	127	191	354	664	1255
Avg	1.86	4.51	9.21 12.78	16.44 18.59	30.42 35.65	54.39 66.71	99.36 126.74	176.30 190.49	320.61 353.55	589.61 663.59	1091.70 1254.53

N is the bit-size of a nonnegative constant; $r = W\left(4.(N+1).\,log(2)\right)/log(2)$.
…: Serial implementation.

(7) achieves a reduction of 10.86% over (4) in *Ath*, while it causes an increase of 17.54% and 9.77% in *Upb* and *Avg*, respectively. Contrary to *Avg* values corresponding to (4), the ones of (7) are relatively far from *Upb*. Compared to CSD, a saving of 50% in *Ath* is obtained by (7) for *N*=56.

Finally, to decide which *r* expression to use depends actually on the design requirements. If area is targeted, (4) is used. But in case speed or power are a concern, (7) is suitable.

IV. RADIX-2^r: OVERFLOW SAFETY

In fixed-point representation, an overflow risk in SCM is possible. It might be caused by uncontrolled left-shift spans, especially for the last partial product (PP). Thus, lower bounds on the maximum left-shift must be carefully considered to ensure an overflow safety– this is more likely to the detriment of the optimization of the number of additions [3]. As far as we are aware, this issue has never been addressed in SCM despite the big number of proposed heuristics.

In RADIX-2^r, overflow safety is easy to prove. We consider two nonnegative numbers, *C* and *X*, with *n* and *m* bit-lengths, respectively. In two's complement representation, the product $P = C \times X$ needs *n*+*m*+2 bits to be complete, i.e., without truncation. We can write: $P = p_{n+m+1}\, p_{n+m} \cdots p_1\, p_0$; where p_{n+m+1} is the sign bit. To be sure there is no overflow risk; we must prove that the sign-bit of the last PP is set *at most* at the *n*+*m*+1 position. We write:

$$P = \sum_{j=0}^{\frac{n+1}{r}-1} Q_j \times X \times 2^{rj} = \sum_{j=0}^{\frac{n+1}{r}-1} (-1)^{c_{rj+r-1}} \times |Q_j| \times (-1)^{x_m} \times |X| \times 2^{rj} = \sum_{j=0}^{\frac{n+1}{r}-1} PP_j\,,$$

where the last PP is $PP_{(n+1)/r-1} = (-1)^{c_n} \times |Q_j| \times (-1)^{x_m} \times |X| \times 2^{n+1-r}$. The maximal positive values that $|Q_j|$ and $|X|$ can take are 2^{r-1} and 2^m, respectively, to which corresponds a maximal PP of $\max\left(PP_{(n+1)/r-1}\right) = (-1)^{c_n + x_m} \times 2^{n+m}$. In this case, 2^{n+m} occupies the *n*+*m* position, plus the sign bit just after at the *n*+*m*+1 position. This proves that in RADIX-2^r overflow never occurs.

V. RADIX-2^r: FURTHER IMPROVEMENTS

The objective is to decrease *Avg* without increasing *Upb*. *Avg* is successively reduced in two steps: by the utilization of a redundant recoding, followed by a Common Digit Elimination (CDE) step on the PP set. In RADIX-2^r, CDE is already applied on the odd-digits (m_j) by the recoding itself. A second order of CDE can be applied again on the Q_j terms thanks to redundancy. We present hereafter a linear runtime Redundant Radix-2^r Recoding (R3) with a better *Avg* while preserving the same *Upb* as in RADIX-2^r.

Equation (1) can be rewritten in more details as

$$C = \sum_{j=0}^{(N+1)/r-1} (-1)^{c_{rj+r-1}} \times \left(m_j \times 2^{k_j}\right) \times 2^{rj}\,, \qquad (8)$$

with $m_j \in \{0, 1, 3, 5, \ldots, 2^{r-1}-1\}$ and $k_j \in \{0, 1, 2, \ldots, r-1\}$.

To enable CDE at the Q_j level, we announce the following theorem.

Theorem 1. *Any digit* $Q_j \in DS(2^r)$ *can be represented in a combination of digits* $P_{ji} \in DS(2^s)$, *such as s is a divider of r.*

The proof of this theorem is given in [12]. When Th. (1) is applied to eq. (1), it gives: $C = \sum_{j=0}^{(N+1)/r-1} \left[\sum_{i=0}^{(r/s)-1} P_{ji}\, 2^{si} \right] 2^{rj}$ (9), where $P_{ji} \in DS(2^s) = \{-2^{s-1}, -2^{s-1}+1, \ldots, 0, \ldots, 2^{s-1}-1, 2^{s-1}\}$, $OM(2^s) = \{1, 3, \ldots, 2^{s-1}-1\}$ such as $|OM(2^r)|/|OM(2^s)| = 2^{(k-1)s}$ with *r*/*s*=*k*. The major advantage of Theorem (1) is that it yields an exponential reduction ($1/2^{(k-1)s}$) of the number of odd-digits in (9) in comparison to (1), but at the expense of a linear increase (*k*–1) in the number of additions. Theorem (1) allows a *recursive* recoding which enabled to design efficient variable multipliers [12] and multi-precision multipliers [13].

Corollary 1. *In radix-*2^r, $|Q_j| = u_j \times 2^{l_j} + (-1)^{e_j} \times v_j \times 2^{h_j}$, *where*: $u_j,\, v_j \in \{0, 1, 3, 5, \ldots, 2^{(r/2)-1}-1\}$; $l_j \in \{0, 1, 2, \ldots, r-1\}$; $h_j \in \{0, 1, 2, \ldots, (r/2)-1\}$; *and* $e_j \in \{0, 1\}$.

Proof. This corollary is a direct consequence of Theorem (1) applied for *r*/*s*=2. This means that Q_j digit, which is *r*+1 bit-length, is split into two overlapping sub-digits P_{j0} and P_{j1}, each of *r*/2+1 bit-length. This assumes that *r* is even. If *r* is odd, Theorem (2) in [12] is applied instead of Theorem (1). For *r*/*s*=2, equation (9) becomes: $C = \sum_{j=0}^{(N+1)/r-1} \left(P_{j0} + P_{j1} \times 2^{r/2}\right) \times 2^{rj}$. Note that $Q_j = P_{j0} + P_{j1} \times 2^{r/2}$, and that P_{j0} and P_{j1} have exactly the same properties as Q_j, which means that they can be expressed in the same way Q_j is written in (8). Thus, we get

$$C = \sum_{j=0}^{(N+1)/r-1} (-1)^{c_{rj+r-1}} \times \left[u_j \times 2^{l_j} + (-1)^{e_j} \times v_j \times 2^{h_j}\right] \times 2^{rj}\,. \qquad (10)$$

Because addition is a *non-injective* function, the quintuplet (u_j, l_j, e_j, v_j, h_j) is not unique; several ones might exist for the same $|Q_j|$ value. For instance, $|Q_j| = 35$ can be expressed as $35 = 1\times2^5 + 3\times2^0$, or $35 = 5\times2^3 - 5\times2^0$, or $35 = 7\times2^2 + 7\times2^0$. Consequently, Eq. (10) is a Redundant Radix-2^r Recoding (R3) [10] of the constant C.

Corollary (1) is just one case (*r*/*s*=2) among many others. A number of Q_j partitionings are possible (*r*/*s*=3, 4, 5, ...), but higher values of *r*/*s* increase the number of sub-digits (u_j, v_j, w_j, t_j, z_j, …), which makes (10) difficult to handle.

R3 algorithm is illustrated hereafter for the particular case of 21≤*N*≤83. For this interval, optimal *Upb* in RADIX-2^r is attained with *r*=4 (see the *Upb* formula). To preserve optimality in *Upb* for R3, the trick here is to use sub-digits (P_{j0} and P_{j1}) with *s*=4, which means that for Q_j *r*=2×4=8. Hence, with (*s*, *r*)=(4, 8) optimality in *Upb* is guaranteed.

For *r*=8, 0≤ $|Q_j|$ ≤128, and (10) becomes:

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCSII.2014.2387620, IEEE Transactions on Circuits and Systems II: Express Briefs

$$C = \sum_{j=0}^{(N+1)/8-1} \left(u_j \times 2^{l_j} + (-1)^{e_j} \times v_j \times 2^{h_j}\right) \times (-1)^{c_{8j+7}} \times 2^{8j}$$
$$= \sum_{j=0}^{(N+1)/8-1} (Z_1 + Z_2)_j \times (-1)^{c_{8j+7}} \times 2^{8j}, \qquad (11)$$

where $Z_1 = u_j \times 2^{l_j}$; $Z_2 = (-1)^{e_j} \times v_j \times 2^{h_j}$; u_j and $v_j \in \{0,1,3,5,7\}$; $l_j \in \{0,1,2,...,7\}$; $h_j \in \{0,1,2,3\}$; and $e_j \in \{0,1\}$.

Note that $|Q_j|=(Z_1+Z_2)_j$. The product $C \times X$ becomes:

$$C \times X = \sum_{j=0}^{(N+1)/8-1} \left[(u_j \times X) \times 2^{p_j} + (-1)^{e_j} \times (v_j \times X) \times 2^{h_j}\right] \times (-1)^{c_{8j+7}} \times 2^{8j} \qquad (12)$$

The partitioning of the constant C according to (11) is depicted in Fig. 1.

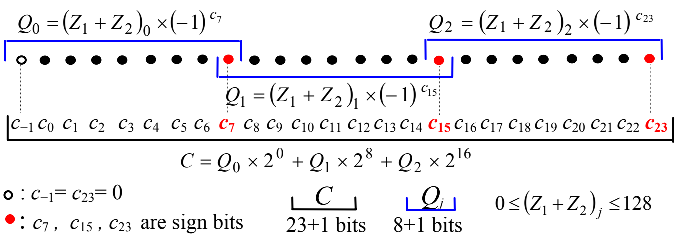

Fig. 1. Partitioning of a 23-bit constant C using R3 algorithm.

Since $|Q_j|$ may have several notations in (Z_1, Z_2), we must carefully select among a big number of cases, the recoding (R3) that yields an *Avg* not higher than RADIX-2^r *Avg*. We have shown that for RADIX-2^r, $Avg(\forall Q_j = 0) = \lceil (N+1)/r \rceil / 2^r$, and based on the same reasoning developed in Section II we can easily prove that $Avg(\forall Q_j = 1) = (2 \times r - 1) \times \lceil (N+1)/r \rceil / 2^r$. Thus, we can write: $Avg(\forall Q_j = 0,1) = r \times \lceil (N+1)/r \rceil / 2^{r-1}$. Keeping the same $Avg(\forall Q_j = 0,1)$ value in R3 ensures that the total R3 *Avg* will not be higher than RADIX-2^r *Avg*, because the number of PPs and the odd-digit set are identical in R3 and RADIX-2^r. This means also that R3 and RADIX-2^r have the same *Ath*.

One efficient R3 recoding is obtained using a C-program that exhaustively explores for each odd $|Q_j|$ varying from 1 to 127, all (u_j, l_j, e_j, v_j, h_j) possibilities and selects the least adder consumer combination according to the following priority ordering: $(u_j,v_j)=(u_j,0)$; $(u_j,v_j)=(1,1)$; $(Z_1,Z_2)=(1\times 2^7,Z_2)$; and finally $(Z_1,Z_2)=(Z_1,\pm 1\times 2^0)$. These two latter couples allow the following simplifications:

$$\cdots + (1\times 2^7 + Z_2)\times 2^{8j} + (Z_1 - 1\times 2^0)\times 2^{8j+8} \pm \cdots = \cdots - (1\times 2^7 - Z_2)\times 2^{8j} + Z_1 \times 2^{8j+8} \pm \cdots$$
$$\cdots - (1\times 2^7 + Z_2)\times 2^{8j} + (Z_1 + 1\times 2^0)\times 2^{8j+8} \pm \cdots = \cdots + (1\times 2^7 - Z_2)\times 2^{8j} + Z_1 \times 2^{8j+8} \pm \cdots$$

In case none of those cited cases is encountered, C-program pursues in the following priority ordering: $(u_j,v_j)=(1,3)$ or $(3,1)$; $(u_j,v_j)=(3,3)$; $(u_j,v_j)=(1,5)$ or $(5,1)$; $(u_j,v_j)=(5,5)$; $(u_j, v_j)=(1,7)$ or $(7,1)$; $(u_j,v_j)=(7,7)$; $(u_j,v_j)=(3,5)$ or $(5,3)$; $(u_j,v_j)=(3,7)$ or $(7,3)$; $(u_j,v_j)=(5,7)$ or $(7,5)$. This ordering maximizes the occurrences of the digit "1", then of "3", and minimizes those of "5" and "7" in $|Q_j|$ digits, which will more likely reduce the number of additions in the whole recoding of the constant C. Optimized odd $|Qj|$ combinations are grouped in Table III. Even $|Qj|$ combinations are directly derived from the odd ones using a left-shift operation.

For a given $21 \le N \le 83$, optimality in *Upb* for RADIX-2^r and R3 is guaranteed with r=4 and (s, r)=(4, 8), respectively. To RADIX-2^r corresponds $Avg(\forall Q_j = 0,1) = \lceil (N+1)/4 \rceil / 2$.

Counting the number of u_j=1, v_j=0, and v_j=1 in both the odd and even $|Q_j|$ of Table III, we can easily prove that for R3, $Avg(\forall v_j = 0) = 24 \times \lceil (N+1)/8 \rceil / 128$ and $Avg(\forall u_j = 1) + Avg(\forall v_j = 1) = 104 \times \lceil (N+1)/8 \rceil / 128$. This gives $Avg(\forall u_j = 1) + Avg(\forall v_j = 0,1) = \lceil (N+1)/8 \rceil$, which is equal to $Avg(\forall Q_j = 0,1)$. This is the formal proof that R3 *Avg* can not be higher than RADIX-2^r *Avg*.

As for *Upb*, R3 comprises $\lceil (N+1)/8 \rceil$ terms Q_j, each one groups two digits (Z_1,Z_2). Thus, the total number of PPs is $\lceil (N+1)/4 \rceil$. Since 3 odd-digits are required, $Upb = \lceil (N+1)/4 \rceil + 2$, which is equal to RADIX-2^r *Upb*. It is important to mention that $21 \le N \le 83$ was chosen just to make the demonstration simpler (Table III), but the proofs hold true for any value of N.

TABLE III
R3 ALGORITHM: ODD AND EVEN $|Q_J|$ DIGIT RECODING FOR $21 \le N \le 83$

Odd $\lvert Q_j \rvert$	$Z_1=u_j \times 2^{l_j}$	$Z_2=(-1)^{e_j} \times v_j \times 2^{h_j}$	$(Z_1+Z_2)_j$	Even $\lvert Q_j \rvert$	$(Z_1+Z_2)_j$
1	1×2^0	0×2^0	U_1	2	$2^1\times U_1$
3	3×2^0	0×2^0	U_3	4	$2^2\times U_1$
5	5×2^0	0×2^0	U_5	6	$2^1\times U_3$
7	7×2^0	0×2^0	U_7	8	$2^3\times U_1$
9	1×2^3	1×2^0	U_9	10	$2^1\times U_5$
11	3×2^2	-1×2^0	U_{11}	12	$2^2\times U_3$
13	3×2^2	1×2^0	U_{13}	14	$2^1\times U_7$
15	1×2^4	-1×2^0	U_{15}	16	$2^4\times U_1$
17	1×2^4	1×2^0	U_{17}	18	$2^1\times U_9$
19	5×2^2	-1×2^0	U_{19}	20	$2^2\times U_5$
21	5×2^2	1×2^0	U_{21}	22	$2^1\times U_{11}$
23	3×2^3	-1×2^0	U_{23}	24	$2^3\times U_3$
25	3×2^3	1×2^0	U_{25}	26	$2^1\times U_{13}$
27	7×2^2	-1×2^0	U_{27}	28	$2^2\times U_7$
29	7×2^2	1×2^0	U_{29}	30	$2^1\times U_{15}$
31	1×2^5	-1×2^0	U_{31}	32	$2^5\times U_1$
33	1×2^5	1×2^0	U_{33}	34	$2^1\times U_{17}$
35	1×2^5	3×2^0	U_{35}	36	$2^2\times U_9$
37	1×2^5	5×2^0	U_{37}	38	$2^1\times U_{19}$
39	5×2^3	-1×2^0	U_{39}	40	$2^3\times U_5$
41	5×2^3	1×2^0	U_{41}	42	$2^1\times U_{21}$
43	5×2^3	3×2^0	U_{43}	44	$2^2\times U_{11}$
45	3×2^4	-3×2^0	U_{45}	46	$2^1\times U_{23}$
47	3×2^4	-1×2^0	U_{47}	48	$2^4\times U_3$
49	3×2^4	1×2^0	U_{49}	50	$2^1\times U_{25}$
51	3×2^4	3×2^0	U_{51}	52	$2^2\times U_{13}$
53	3×2^4	5×2^0	U_{53}	54	$2^1\times U_{27}$
55	7×2^3	-1×2^0	U_{55}	56	$2^3\times U_7$
57	7×2^3	1×2^0	U_{57}	58	$2^1\times U_{29}$
59	1×2^6	-5×2^0	U_{59}	60	$2^2\times U_{15}$
61	1×2^6	-3×2^0	U_{61}	62	$2^1\times U_{31}$
63	1×2^6	-1×2^0	U_{63}	64	$2^6\times U_1$
65	1×2^6	1×2^0	U_{65}	66	$2^1\times U_{33}$
67	1×2^6	3×2^0	U_{67}	68	$2^2\times U_{17}$
69	1×2^6	5×2^0	U_{69}	70	$2^1\times U_{35}$
71	1×2^6	7×2^0	U_{71}	72	$2^3\times U_9$
73	5×2^4	-7×2^0	U_{73}	74	$2^1\times U_{37}$
75	5×2^4	-5×2^0	U_{75}	76	$2^2\times U_{19}$
77	5×2^4	-3×2^0	U_{77}	78	$2^1\times U_{39}$
79	5×2^4	-1×2^0	U_{79}	80	$2^4\times U_5$
81	5×2^4	1×2^0	U_{81}	82	$2^1\times U_{41}$
83	5×2^4	3×2^0	U_{83}	84	$2^2\times U_{21}$
85	5×2^4	5×2^0	U_{85}	86	$2^1\times U_{43}$
87	5×2^4	7×2^0	U_{87}	88	$2^3\times U_{11}$
89	3×2^5	-7×2^0	U_{89}	90	$2^1\times U_{45}$
91	3×2^5	-5×2^0	U_{91}	92	$2^2\times U_{23}$
93	3×2^5	-3×2^0	U_{93}	94	$2^1\times U_{47}$
95	3×2^5	-1×2^0	U_{95}	96	$2^5\times U_3$
97	3×2^5	1×2^0	U_{97}	98	$2^1\times U_{49}$
99	3×2^5	3×2^0	U_{99}	100	$2^2\times U_{25}$
101	3×2^5	5×2^0	U_{101}	102	$2^1\times U_{51}$
103	3×2^5	7×2^0	U_{103}	104	$2^3\times U_{13}$
105	7×2^4	-7×2^0	U_{105}	106	$2^1\times U_{53}$
107	7×2^4	-5×2^0	U_{107}	108	$2^2\times U_{27}$
109	7×2^4	-3×2^0	U_{109}	110	$2^1\times U_{55}$
111	7×2^4	-1×2^0	U_{111}	112	$2^4\times U_7$
113	7×2^4	1×2^0	U_{113}	114	$2^1\times U_{57}$
115	7×2^4	3×2^0	U_{115}	116	$2^2\times U_{29}$
117	7×2^4	5×2^0	U_{117}	118	$2^1\times U_{59}$
119	7×2^4	7×2^0	U_{119}	120	$2^3\times U_{15}$
121	1×2^7	-7×2^0	U_{121}	122	$2^1\times U_{61}$
123	1×2^7	-5×2^0	U_{123}	124	$2^2\times U_{31}$
125	1×2^7	-3×2^0	U_{125}	126	$2^1\times U_{63}$
127	1×2^7	-1×2^0	U_{127}	128	$2^7\times U_1$

Note that $9=1\times 2^3+1\times 2^0$ in R3 (1 addition) and $9=1\times 2^4-7\times 2^0$ in RADIX-2^r (2 additions), taking into account that the recoding is on 8+1=9 bits (Fig. 1). There are many cases where the number of additions is lower, as in 10, 40,…

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCSII.2014.2387620, IEEE Transactions on Circuits and Systems II: Express Briefs

CDE is performed in a linear runtime on the $\lceil (N+1)/8 \rceil$ digits U_k as an ultimate optimization step. It is illustrated by the product $P=(2631689)_{10}\times X$. We first calculate the product (P) in RADIX-2^r and then in R3.

$P_{RADIX} = X_0\times 2^{20}-X\times 2^{19}+X_0\times 2^{12}-X\times 2^{11}+X\times 2^4-X_1$
with $X_0=(X\times 2)+X$ and $X_1=(X\times 2^3)-X$.
$P_{R3}=U_{40}\times 2^{16}+U_{40}\times 2^8+U_9$ with $U_{40}= U_5\times 2^3$; $U_5=(X\times 2^2)+X$ and $U_9=(X\times 2^3)+X$. Note that P_{RADIX} requires 7 additions, while P_{R3} needs only 4. A saving of 2 additions is due to the redundancy (U_9 and U_{40}), and a saving of 1 addition is due to CDE (U_{40}).

Avg has been *exhaustively* calculated for values of *C* varying from 0 to 2^N-1, for N=8, 16, 24, and 32. But for N=64, we have computed *Avg* using 10^{10} uniformly distributed random values of C. For N=64, R3 uses 14.16% less additions than RADIX-2^r (Table IV). For $N\leq 32$, the saving is not substantial because the number of U_k digits is low (≤4). But for N=64, it is equal to 8, offering more possibilities to CDE.

We have also determined the smallest value that requires *q* additions, for *q* varying from 1 to the *Upb* of the recoding. Table V summarizes the results for a 32-bit constant. Note that starting from *q*=7, higher values are given by R3.

We have compared R3 to a number of well-known non-recoding heuristics, for which neither *Avg* nor *Upb* bounds are known. While they exhibit lower *Avg* (Fig. 2), their respective *Upb* could be higher (Bernstein's algorithm, Table VI).

TABLE IV
R3 VERSUS RADIX-2^r: AVERAGE NUMBER OF ADDITIONS (*Avg*)

N	*Avg*		Saving
	RADIX-2^r	R3	%
8	1.86	1.79	3.76
16	4.51	4.32	4.21
24	6.79	6.48	4.56
32	8.96	8.51	5.02
64	17.51	15.03*	14.16

*: Obtained from 10^{10} uniformly distributed random values of C. *N* is the bit-size of the constant C. For *N*=8, the saving is exclusively due to the redundancy (see Table III).

TABLE V
R3 VERSUS RADIX-2^r: SMALLEST VALUES UP TO A 32-BIT CONSTANT

q	RADIX-2^r	R3
1	3	3
2	11	11
3	43	43
4	139	139
5	651	651
6	2699	2699
7	33419	34971
8	526491	559259
9	8422027	17336475
10	134744219	143163547
11	2155905675	2290385547

q: number of additions.

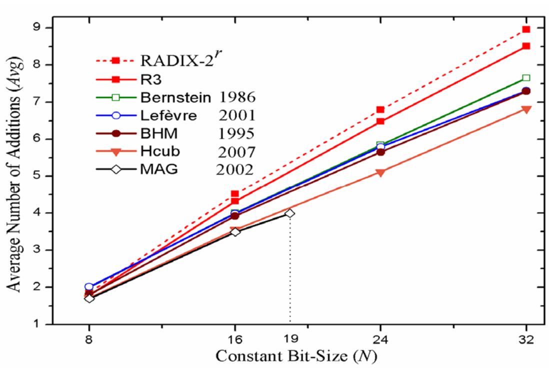

Fig. 2. *Avg* comparison for an *N*-bit constant.

VI. CONCLUSION AND FUTURE WORK

A fully-predictable and sublinear-runtime SCM heuristic has been developed (RADIX-2^r) and improved (R3). In addition to the maximum number of additions, we have also determined the exact complexities for the average and adder-depth. These three complexities are the lowest analytic bounds known so far for the multiplication by a constant. However, optimal bounds remain an open research problem.

Our current work deals with the application of radix-2^r arithmetic to the multiple-constant-multiplication problem.

TABLE VI
R3 AND RADIX-2^r VERSUS NON-RECODING ALGORITHMS: RUNTIME COMPLEXITY AND NUMBER OF ADDITIONS OF SOME SPECIAL CASES

Algorithm	$(84AB5)_H$ *N*=20	$(64AB55)_H$ *N*=23	$(5959595B)_H$ *N*=31	Runtime [7]
BIGE [14]	<u>4</u>	<u>5</u>	<u>6</u>	$O(2^N)$
Bernstein [4]	8^G	7	8	$O(2^N)$ [5]
Hcub [7]	<u>4</u>	6	8	$O(N^6)$
BHM [6]	5	7	9	$O(N^4)$
Lefèvre [5]	<u>4</u>	6	9	$O(N^3)$
RADIX-2^r [1]	5	7	10	$O(N/r)$
R3	<u>4</u>	6	8	$O(N)$

N: Constant bit-size; $r=2\cdot W\left(\sqrt{(N+1)\cdot \log(2)}\right)/\log(2)$; G: Greater than R3 Upb; R3 Upb= 7, 8, and 10 for *N*=20, 23, and 31, respectively; <u>x</u>: Optimal number of additions.

REFERENCES

[1] A.K. Oudjida and N. Chaillet, "Radix-2^r Arithmetic for Multiplication by a Constant," IEEE Trans. on Circuits and Systems II: Express Brief, vol. 61, no 5, pp. 349-353, May 2014.
[2] A. Avizienis, "Signed-digit number representation for fast parallel arithmetic," IRE Trans. on Electronic Computers, vol. EC-10, No. 3, pp. 389–400, September 1961.
[3] V.S. Dimitrov, L. Imbert, and A. Zakaluzny, "Multiplication by a Constant is Sublinear," Proceedings of the 18th IEEE Symposium on Computer Arithmetic (ARITH), pp. 261-268, Montpellier, France, June 25-27 2007.
[4] R.L. Bernstein, "Multiplication by Integer Constant," Software– Practice and Experience 16, 7, pp. 641-652, 1986.
[5] V. Lefèvre, "Multiplication by an Integer Constant," INRIA Research Report, No. 4192, Lyon, France, May 2001.
[6] A.G. Dempster and M.D. Macleod, "Use of Minimum Adder Multiplier Blocks in FIR Digital Filters," IEEE Trans. on Circuits and Systems-II: Analog and Digital Signal Processing 42, 9, pp. 569-567, 1995.
[7] Y. Voronenko and M. Püschel, "Multiplierless Multiple Constant Multiplication," ACM Trans. on Algorithms (TALG), vol. 3, No. 2, article 11, pp. 1-38, May 2007.
[8] O. Gustafsson, A.G. Dempster, and L. Wanhammar, "Extended Results for Minimum-Adder Constant Integer Multipliers," Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS), vol. 1, pp. I-73 I-76, Scottsdale Arizona, USA, May 2002.
[9] K. Johansson, O. Gustafsson, L.S. DeBrunner, and L. Wanhammar, "Minimum Adder Depth Multiple Constant Multiplication Algorithm for Low-Power FIR Filters," Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1439-1442, Rio de Janeiro, Brazil, May 2011.
[10] A.K. Oudjida, M.L. Berrandjia, and N. Chaillet, "A New Low-Power Recoding Algorithm for Multiplierless Single/Multiple Constant Multiplication," Proceedings of the 12th edition of IEEE-FTFC Low-Voltage Low-Power Conference, DOI:10.1109/FTFC.2013.6577750, Paris, France, June 19-21 2013.
[11] R.I. Hartley, "Subexpression Sharing in Filters Using Canonic Signed Digit Multipliers," IEEE Trans. on Circuits and Systems II: Analog and Digital Signal Processing, vol. 43, No. 10, pp. 677-688, October 1996.
[12] A.K. Oudjida, N. Chaillet, M.L. Berrandjia, and A. Liacha, "A New High Radix-2^r ($r \geq 8$) Multibit Recoding Algorithm for Large Operand Size ($N \geq 32$) Multipliers," Journal of Low Power Electronics (JOLPE), vol. 9, N° 1, pp. 50-62, ISSN: 1546-1998/2013/9/50/62, American Scientific Publishers (ASP), April 2013.
[13] A.K. Oudjida, N. Chaillet, A. Liacha, and M.L. Berrandjia, "A New Recursive Multibit Recoding Algorithm for High-Speed and Low-Power Multiplier, " Journal of Low Power Electronics (JOLPE), vol. 8, N° 5, pp. 579-594, ISSN: 1546-1998/2012/8/579/594, American Scientific Publishers (ASP), December 2012.
[14] J. Thong and N. Nicolici, "An optimal and practical approach to single constant multiplication," IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, vol. 30, no. 9, pp. 1373-1386, Sep. 2011.