A new Low-Power recoding algorithm for multiplierless single/multiple constant multiplication.
Abdelkrim K. Oudjida, Mohamed L. Berrandjia, Nicolas Chaillet

To cite this version:

HAL Id: hal-00869542
https://hal.archives-ouvertes.fr/hal-00869542
Submitted on 7 Oct 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
A New Low-Power Recoding Algorithm for Multiplierless Single/Multiple Constant Multiplication

A.K. Oudjida, M.L. Berrandjia
Microelectronics and Nanotechnology Division
Centre de Développement des Technologies Avancées
Algiers, Algeria
a_oudjidia@cdta.dz

N. Chailliet
AS2M Department
FEMTO-ST Institute
Besançon, France
nicolas.chaillet@femto-st.fr

Abstract—Optimizing the number of additions in constant coefficient multiplication is conjectured to be a NP-hard problem. In this paper, we report a new heuristic requiring an average of 29.10% and 10.61% less additions than the standard canonical signed digit representation (CSD) and the double base number system (DBNS), respectively, for 64-bit coefficients. The maximum number of additions per coefficient is bounded by \((N/4)+2\), and the time-complexity of the recoding is linearly proportional to \(N\), where \(N\) is the bit-size of the constant. These performances are achieved using a new redundant version of radix-2\(^8\) recoding.

Keywords—Double Base Number System (DBNS); High-Speed and Low-Power Design; Multiplierless Single/Multiple Constant Multiplication (SCM/MCM); Radix-2\(^r\) Booth recoding.

I. BACKGROUND AND MOTIVATION

Many applications in DSP and control, such as linear time invariant (LTI) filters/controllers, involve the computation of a large number of multiplications of one variable by a set of constants. To be efficiently handled, the implementation must be multiplierless, that is, using exclusively additions, subtractions and shifts. This problem is known as single/multiple constant multiplication (SCM/MCM) and is conjectured to be NP-hard [1]. A big number of heuristics have been proposed. They are classified into four categories:

- Digit-recoding heuristics such as CSD [2] and DBNS [3];
- Common subexpression elimination (CSE) using pattern matching. Examples are Lefèvre [4] and Boullis [5];
- Directed acyclic graph (DAG) based algorithms such as Hcub [6], H(k) [7], and MAG [8];
- Mixed algorithms combining CSE and DAG such as the recent optimal algorithm BIGE [1].

A good survey and a detailed comparative study showing pros and cons of various algorithms is given in [1][6][9]. Despite the big number of proposed heuristics, the vast majority of LTI system optimizations use the CSD representation for constant encoding [10]. The rational is that:

- CSD recoding is easy to implement;
- The adder complexity in CSD is known, which is not the case for the other heuristics [1][11]. In CSD the number of adders is bounded by \((N+1)2-1\) and tends asymptotically to an average value of \((N/3)-8/9\), which yields to 33% saving over the naïve add-and-shift approach.

The central point of this work is the minimization of the total number of additions. Based on radix-2\(^r\) signed-digit number system [14][15], a new Redundant Radix-2\(^r\) Recoding (R3) is proposed as an alternative to existing heuristics. Applied to the particular case of radix-2\(^8\) with \(N=64\), a saving of 29.10% is achieved over CSD, which yields to much less power consumption and more speed. In addition, the new recoding shows high aptitude for common subexpression elimination, which makes it a good candidate for MCM.

The paper is organized as follows. Section I outlines the necessity of a linear runtime heuristic with a high compression ratio to handle large bit-size constants. Section II introduces the new R3 algorithm, while Section III compares the results to CSD and DBNS recodings. Finally, Section IV provides some concluding remarks and suggestions for future work.

II. NEW REDUNDANT RADIX-2\(^r\) ALGORITHM (R3) FOR MULTIPLICATION BY A N-BIT CONSTANT

A N-bit C constant is expressed in radix-2\(^r\) as follows:

\[
C = \sum_{j=0}^{(N/r)-1} c_j 2^j + 2^j c_{j+1} 2^{j+1} + 2^j c_{j+2} 2^{j+2} + \ldots + 2^j c_{j+r-2} 2^{j+r-2} - 2^{-j} c_{j+r-1} \]

\[
= \sum_{j=0}^{(N/r)-1} Q_j 2^j \quad ;
\]

where \(c_{-1} = 0\) and \(r \in \mathbb{N}^+\). For simplicity purposes and without loss of generality, we assume that \(r\) is a divider of \(N\).

This work is supported by “Centre de Développement des Technologies Avancées (CDTA), Algiers, Algeria, in collaboration with FEMTO-ST Institute, Besançon, France.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Name</th>
<th>Author</th>
<th>Year</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>BITE</td>
<td>Thong</td>
<td>[1]</td>
<td>2011</td>
<td>(O(2^N))</td>
</tr>
<tr>
<td>H(k)</td>
<td>Dempster</td>
<td>[7]</td>
<td>2004</td>
<td>(O(2^N))</td>
</tr>
<tr>
<td>MAG</td>
<td>Gustafsson</td>
<td>[8]</td>
<td>2002</td>
<td>(\Omega(2^N))</td>
</tr>
<tr>
<td>BIGE</td>
<td>Bernstein</td>
<td>[12]</td>
<td>1986</td>
<td>(O(2^N))</td>
</tr>
<tr>
<td>Hcub</td>
<td>Voronenko</td>
<td>[6]</td>
<td>2007</td>
<td>(O(N^3))</td>
</tr>
<tr>
<td>BHM</td>
<td>Dempster</td>
<td>[13]</td>
<td>1995</td>
<td>(O(N^4))</td>
</tr>
<tr>
<td>- Lefèvre</td>
<td>[4]</td>
<td>2001</td>
<td>(O(N^3))</td>
<td></td>
</tr>
<tr>
<td>DBNS</td>
<td>Dimitrov</td>
<td>[5]</td>
<td>2007</td>
<td>(O(N))</td>
</tr>
<tr>
<td>CSD</td>
<td>Avizienis</td>
<td>[2]</td>
<td>1961</td>
<td>(O(N))</td>
</tr>
</tbody>
</table>
In eq. (1), the two’s complement representation of \( C \) constant is split into \( N/r \) two’s complement slices (\( Q_j \)), each of \( r+1 \) bit length. Each pair of contiguous slices has one overlapping bit. To eq. (1), corresponds a digit-set \( D[j^r] \) such as:

\[
Q_j \in D[j^r] = \left\{ \text{-2}^{r-1}, \text{-2}^{r-2} + l, \ldots, \text{-2}^0, 0, \text{1}, \text{2}, \cdots, \text{2}^{r-1}-1, \text{2}^{r-1} \right\}
\]

The sign of \( Q \) term is given by \( c_{j^r} \), bit, and \( Q_j = \sum_{k=0}^{2^r-1} m_j \times 2^k \), with \( m_j \in \{0,1,2,\ldots,r-1\} \) and \( m_j \in OM[j^r] = \left\{ 1, 3, 5, \ldots, 2^{r-1}-1 \right\} \cup \{0\} \). \( OM[j^r] \) represents the required set of odd-multiples in radix-2\(^r\) recoding, with \( |OM[j^r]| = 2^r-2 \). Finally, \( C \) can be expressed as follows:

\[
C = \sum_{j=0}^{N/r-1} (1)^{j\%r+1} \times m_j \times 2^k \times 2^j .
\]

Equation (2) is not redundant since for each \( C \) constant corresponds a unique representation (\( m_j \)). To make the solution space larger in order to select a less adder-consuming representation of \( C \), the recoding must be redundant. To achieve such a goal, we announce the following theorem:

**Theorem 1**. In radix-2\(^r\), \( |Q|=|A_j \times 2^p+(-1)^e \times B_j \times 2^h| \), where: \( A_j, B_j \in \{0,1,3,5,7\} \); \( p \in \{0,1,2,\ldots,r-1\} \); and \( e \in \{0,1\} \).

The proof of the above theorem is based on our Theorem (1) described in [16][17]. Note that different notations for \( Q \) are possible. For instance: \( 37=1 \times 2^5+5 \times 2^1 \) or \( 37=5 \times 2^3-3 \times 2^1 \). We illustrate the idea for \( r=8 \), where \( 0 \leq |Q| \leq 128 \). Equation (2) becomes:

\[
C = \sum_{j=0}^{N/r-1} (1)^{j\%r+1} \times \left( A_j \times 2^p+(-1)^e \times B_j \times 2^h \right) \times (-1)^{j\%r+1} \times 2^j
\]

where \( Z_1=A_j \times 2^p \); \( Z_2=(-1)^e \times B_j \times 2^h \); \( A_j, B_j \in \{0,1,3,5,7\} \); \( p \in \{0,1,2,\ldots,r-1\} \); and \( e \in \{0,1\} \).

Note that \( |Q|=(Z_1+Z_2) \). The partitioning of \( C \) constant according to eq. (3) is depicted in Fig. 1, while the recodings of odd and even \( Q \) digits are separately denoted in Table II.

The product \( C \times X \) becomes:

\[
C \times X = \sum_{j=0}^{N/r-1} \left( A_j \times 2^p+(-1)^e \times B_j \times 2^h \right) \times \left( X \times 2^j \times 2^j \right)
\]

Note that when \( A_j, B_j \in \{3,5,7\} \), one extra adder is needed since for instance: \( 3 \times X=2^0 \times X \times X \). The diagram below illustrates the recoding rules using R3 algorithm.

### Table II: Odd and even \( Q \) digit recoding using R3 algorithm

<table>
<thead>
<tr>
<th>Odd ( Q )</th>
<th>Even ( Q )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( Z=1 \times 2^0+2 \times 2^1 )</td>
<td>( Z=1 \times 2^0+2 \times 2^1 )</td>
</tr>
<tr>
<td>( Z=1 \times 2^0+2 \times 2^1 )</td>
<td>( Z=1 \times 2^0+2 \times 2^1 )</td>
</tr>
<tr>
<td>( Z=1 \times 2^0+2 \times 2^1 )</td>
<td>( Z=1 \times 2^0+2 \times 2^1 )</td>
</tr>
<tr>
<td>( Z=1 \times 2^0+2 \times 2^1 )</td>
<td>( Z=1 \times 2^0+2 \times 2^1 )</td>
</tr>
</tbody>
</table>

**Figure 1.** Partitioning of a 24-bit \( C \) constant using R3 algorithm.
Our recoding is highly redundant, i.e., each \([Q]\) may have several notations in \(Z_1\) and \(Z_2\) digits. We fully exploited this property to minimize the number of adders using a C-program which exhaustively explores for each odd \([Q]\), all possible notations and selects the least adder consumer combination according to the following priority order: \((A_1, B_1)=(A, 0); (A_1, B_1)=(1, 1); (Z_1, Z_2)=(1 \times 2^1, Z_2)\); and finally \((Z_1, Z_2)=(Z_1, 1 \times 2^0)\). These two latter couples allow the following simplification:

\[
\ldots \left(\left\lfloor \frac{x \cdot 2^1 + Z_1 \cdot 2^1}{x \cdot 2^1 + 2^1 + Z_1 \cdot 2^1 + 2^1}\right\rfloor \cdot 2^1 \right)
\ldots \left(\left\lfloor \frac{x \cdot 2^1 - Z_1 \cdot 2^1}{x \cdot 2^1 - 2^1 - Z_1 \cdot 2^1 - 2^1}\right\rfloor \cdot 2^1 \right)
\ldots
\]

In case none of those cases is encountered, C-program pursues in the following priority order: \((A_1, B_1)=(1,3)\) or \((3,1)\); \((A_1, B_1)=(3,3); (A_1, B_1)=(1,5)\) or \((5,1)\); \((A_1, B_1)=(5,5)\); \((A_1, B_1)=(1,7)\) or \((7,1)\); \((A_1, B_1)=(7,7); (A_1, B_1)=(3,5)\) or \((5,3)\); \((A_1, B_1)=(3,7)\) or \((7,3)\); \((A_1, B_1)=(5,7)\) or \((7,5)\). This order maximizes the occurrences of 1, then of 3, and minimizes those of 5 and 7 in \([Q]\) digits, which will more likely reduce the number of adders in the whole C recoding. Furthermore, we perform common \(U_k\) digit elimination as an ultimate of 5 and 7 in | | maximizes the occurrences of 1, then of 3, and minimizes those number of adders in the whole | |

Concerning DBNS, Dimitrov [3] calculated average and upper-bound values from \(10^5\) uniformly distributed random constants, for 32 and 64 bits only (Table IV). Note that DBNS upper-bounds will be higher if the worst cases are not attained by the pattern of \(10^5\) constants.

<table>
<thead>
<tr>
<th>Constant Bit-width (N)</th>
<th>CSD ((\text{Avg}))</th>
<th>CSD ((\text{Upb}))</th>
<th>R3 ((\text{Avg}))</th>
<th>R3 ((\text{Upb}))</th>
<th>Saving ((\text{Avg,}%))</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1.7882</td>
<td>4</td>
<td>1.7254</td>
<td>3</td>
<td>3.5119</td>
</tr>
<tr>
<td>16</td>
<td>4.4445</td>
<td>8</td>
<td>4.1050</td>
<td>6</td>
<td>7.6386</td>
</tr>
<tr>
<td>24</td>
<td>7.1111</td>
<td>12</td>
<td>6.2846</td>
<td>8</td>
<td>11.6226</td>
</tr>
<tr>
<td>32</td>
<td>9.7777</td>
<td>16</td>
<td>8.3194</td>
<td>10</td>
<td>14.9145</td>
</tr>
<tr>
<td>64</td>
<td>20.4444</td>
<td>32</td>
<td>14.4932</td>
<td>18</td>
<td>29.1091</td>
</tr>
</tbody>
</table>

*. Obtained from \(10^6\) uniformly distributed random \(C\) values.

Another performance indicator of the recoding is the smallest value that requires \(q\) additions, for \(q\) varying from 1 to the upper-bound of the recoding. Table V summarizes this information for 32-bit constant. Note that starting from \(q=7\), higher values are provided by R3 algorithm.

<table>
<thead>
<tr>
<th>Constant Bit-width (N)</th>
<th>DBNS ((\text{Avg}))</th>
<th>R3 ((\text{Avg}))</th>
<th>R3 ((\text{Upb}))</th>
<th>Saving ((\text{Avg,}%))</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>9.05</td>
<td>13</td>
<td>8.3194</td>
<td>10</td>
</tr>
<tr>
<td>64</td>
<td>16.2151</td>
<td>21</td>
<td>14.4932</td>
<td>18</td>
</tr>
</tbody>
</table>

*: Taken from Fig.1 in [3]; *: Obtained from \(10^6\) uniformly distributed random constants.

Predictability in addition-number (Upb and Avg) and runtime/storage requirements informs on the heuristic capabilities and limitations. Upb denotes exactly the length of the critical-path formed by successive additions, while Avg gives an idea on the compression performance of the heuristic. On the other hand, runtime/storage complexity helps to decide whether the use of the heuristic is appropriate with regard to a constant bit-width (N). While this latter is known for all heuristics (Table I), addition complexity is unknown for most of them [1] [11]. Pinch was the first to set an asymptotic
complexity $O(N \log(N))$ for $Upb$ [18]. Better, based on DBNS arithmetic [19], Dimitrov [20] gave a rough evaluation of the hidden constant ($a$) in the big $O$-notation as being $1 \leq a \leq 2$. Only CSD and R3 do have exact analytic expressions for addition complexity (only $Upb$ for R3). For the all remaining heuristics, no addition complexity does exist. This is a real handicap as there is no visibility on how the heuristic evolves with respect to $N$, unless to exhaustively calculate $Avg$ (Fig. 2) and $Upb$, but this is still limited to low values of ($N \leq 32$) as an excessive compute power is required. Though heuristics of Fig. 2 exhibits higher compression ratios than R3 for $N > 16$, some values of Table VI are not only greater than the ones provided by R3, but also equal or even greater than $Upb$ of R3. For $N > 128$, only Lefèvre algorithm remains practical $O(N^3)$, because even when neglecting the hidden constant $a$ in $O(N^3)$, Hcub requires more than 4398 billions of iterations. Another serious drawback of non-recoding heuristics is the overflow risk because of uncontrolled shift spans [3]. Such a problem never occurs in digit-recoding heuristics: CSD, DBNS and R3.

It becomes now clear why despite the large number of existing heuristics, CSD is not only used in designing the vast majority of LTI systems [10], but incorporated in most of advanced synthesis tool as well, such as in Synopsys Design Compiler Ultra [10][21].

IV. CONCLUSION AND FUTURE WORK

An efficient alternative (R3) to the most commonly used heuristic (CSD) has been proposed. Instead CSD, the use of R3 in designing LTI systems leads to much less power consumption and more speed. A pending issue is to determine the analytic expression of the average number of additions ($Avg$) needed by R3 with regard to constant bit-width $N$.

![Figure 2. Comparison of R3 with non-recoding heuristics](image)

**TABLE VI: NUMBER OF ADDERS: SOME PECULIARITIES**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Hexadecimal Values</th>
<th>$N=20$</th>
<th>$N=24$</th>
<th>$N=28$</th>
<th>$N=32$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bernstein [12]</td>
<td>$8^6$</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Hcub* [6]</td>
<td>$6$</td>
<td>8$^2$</td>
<td>9$^4$</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>BHM* [13]</td>
<td>5</td>
<td>7</td>
<td>7</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Lefèvre [4]</td>
<td>4</td>
<td>8$^2$</td>
<td>6</td>
<td>11$^2$</td>
<td></td>
</tr>
<tr>
<td>R3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

* Limited to 26 bits; $a$: Lowest number of additions; $N$: Constant bit-size; E: Equal to $Upb$ of R3; G: Greater than $Upb$ of R3; $Upb$ of R3 $= (N/4)^2$.

**REFERENCES**


