A Reverse Converter for the 4-Moduli Superset $\{2^n-1, 2^n, 2^n+1, 2^{n+1}+1\}$

M. Bhardwaj
Signal Processing and Control ICs
Microelectronics Design Center
Siemens Microelectronics (Asia Pacific)
168 Kallang Way
Singapore 349253.
manish@ezmsgp.hl.siemens.de

T. Srikanthan
Center for High Performance
Embedded Systems (CHiPES)
Nanyang Technological University
Nanyang Avenue
Singapore 639798.
astsrikan@ntu.edu.sg

C. T. Clarke
Submetrics Limited
Lower Bristol Road
Bath
United Kingdom
BA29ER.

Abstract

In this paper the authors propose an extension to the popular $\{2^n-1, 2^n, 2^n+1\}$ moduli set by adding a fourth modulus – $2^{n+1}+1$. This extension leads to higher parallelism while keeping the forward conversion and modular arithmetic units simple. The main challenge of efficient reverse conversion is met by three techniques described herein for the first time. Firstly, we reverse convert linear combinations of moduli hence reducing the number of non-zero bits in the Booth encoded multiplicands from n to merely 2. Secondly, it is shown that division by 3, if introduced at the right stage, can be implemented very efficiently and can, in turn, reduce the cost of the converter. To implement VLSI efficient modulo reduction, we propose two techniques – Multiple Split Tables (MST) and a Modified Division Algorithm (MDA). It is shown that the MST can reduce exponential ROM requirements to quadratic ROM requirements while the MDA can reduce these further to linear requirements. As a result of these innovations, the proposed reverse converter uses simple shift and add operations and needs a lookup with only 6 entries. The delay of the converter is approximately 10n+13 full adder delays and the area cost is quadratic in n.

1. Introduction

In recent years, parallel arithmetic has grown in importance due to the increasing demands by digital signal processors for GOPS-like performance with µW-like power constraints [1]. This has led to renewal of research interest in the highly parallel arithmetic offered by residue number systems (RNS) [2-4]. A significant portion of this research is focussed on reducing the penalty of conversions into and out of this number system – the so called forward and reverse conversions. Such reductions are achieved by proposing new "conversion-friendly" moduli-sets or innovative converter structures for a given set, or a class of sets. As a result of conversion research, it is now well established that forward conversion and residue arithmetic for moduli of the *2^n-type* (i.e. expressible in the form 2^n or $2^n\pm1$) can be implemented quite efficiently [5]. However, there are no similar guidelines for reverse conversion. Individual properties of moduli do not guarantee efficient reverse conversion due to the number theoretic interactions involved. This has been a major hurdle in moving beyond the popular $\{2^n-1, 2^n, 2^n+1\}$ set, which, while offering extremely efficient forward and reverse conversion, cannot be employed in practical systems due to its limited parallelism.

In this paper we propose a new moduli set that increases parallelism while retaining efficient forward conversion and residue arithmetic. This set – $\{2^n-1, 2^n, 2^n+1, 2^{n+1}+1\}$ – is a super-set of the $\{2^n-1, 2^n, 2^n+1\}$ set and hence termed a 4-moduli "superset". With low-cost forward conversion and arithmetic taken care by the choice of the moduli, we focus exclusively on the more difficult task of reverse conversion for this set.

The paper is organized as follows. In section 2, we provide a short overview of residue number systems (RNS) and the various factors that are of importance in choosing a "good" moduli set. We also justify the choice of our superset over other possible 2^n-type 4-moduli supersets. In section 3, we develop a new algorithm to implement the reverse conversion. The VLSI implementation details along with area and delay estimates are presented in section 4. The last section summarizes the cost and performance factors associated with this 4-moduli set and highlights the salient points of this research.

2. Background

The following section introduces the terminology and notation employed in the remainder of the paper.

2.1. Residue Number Systems

Residue Number Systems (RNS) are "carry-free", non-weighted number systems that expose parallelism by representing weighted numbers as tuples of remainders or *residues* [2]. The basis of RNS is a set of *moduli* $\{m_1, m_2, \ldots, m_p\}$, which are usually constrained to be pair-wise relatively prime. The residue representation of an integer X in such a RNS is a p-tuple,

$$X \equiv (x_1, x_2, \ldots, x_p),$$

where $\{x_i\}$ are the least positive residues of X modulo m_i, and are denoted thus –

$$x_i = |X|_{m(i)}, \quad i = 1, 2, \ldots, p.$$

Beginning with zero, all numbers up to and excluding M given by,

$$M = \prod_{i=1}^{p} m_i$$

may be unambiguously encoded in a RNS. This number, M, is called the *dynamic range* of the given RNS. The dynamic range is usually specified in bits, rather than as a number. A RNS is said to have a dynamic range of j bits if it can represent at-least 2^j and not more than 2^{j+1} unique numbers.

The advantage of residue arithmetic is that addition, subtraction and multiplication can be performed for each residue independent of all other residues. Arithmetic is thus parallel and the resulting speedup is close to the number of moduli. This last statement holds only if all moduli share the computation workload equally. In other words, it is desirable to have identical computation costs for all residue channels, which can be guaranteed by having moduli that do not differ greatly in magnitude. This attribute can be quantified by stating the difference (in bits) between the bit-widths of the highest residues of the smallest and largest moduli in the set. We refer to this difference as the *imbalance*. A perfectly balanced set has an imbalance of zero and achieves the maximum attainable speedup. Our discussion here simplifies the actual situation somewhat, since it does not take the nature of moduli into account. To a first order, however, good balance implies a *potential* speedup proportional to the cardinality of the moduli set.

There are two main penalties that negate this significant potential speedup –

1. *Conversion penalties*
   Significant area and delay costs are incurred in forward and reverse conversion. While forward conversion is the classical modulo operation, the theoretical basis for reverse conversion is the Chinese Remainder Theorem (CRT) –

$$X = \left| \sum r_k \cdot \left| \frac{1}{\hat{M}_k} \right|_{m_k} \cdot \hat{M}_k \right|_M$$

$$\hat{M}_k = \frac{M}{m_k}$$

   Another form of the CRT that is more useful for the closed form derivation we shall carry out in sec. 3. is [2] –

$$X + M.A(X) = \sum r_k \cdot \left| \frac{1}{\hat{M}_k} \right|_{m_k} \cdot \hat{M}_k$$

   The summation term on the RHS will be referred to as a partial sum. The second factor in this summation term will be referred to as the inverse corresponding to m_k or, for sake of brevity, simply as "the inverse of m_k" (m_k^{-1}).
2. *Correction penalties*.
   Although the arithmetic itself is highly parallel, the residues resulting from an arithmetic operation must be corrected. Specifically, a residue r_i that results from an arithmetic operation like addition or multiplication of residues is seldom guaranteed to be in the $[0, m_i)$ range and extra circuitry is needed to reduce it to this range. The need for correction is an artifice of the finite wordlength constraint and not an essential mathematical requirement. In fact, by allowing for some increased internal range, the frequency of these corrections can be reduced [6-8].

Note that correction penalties are strongly related to forward conversion penalties, since both are essentially modulo operations. It is instructive at this point to analyze how the choice of moduli affects these penalties. It has been shown conclusively that moduli of the form 2^n or $2^n \pm 1$ (which we refer to as *2^n-type* moduli) have significantly reduced forward conversion and, consequently, correction penalties [5]. This is because the operand to be forward converted (or equivalently modulo corrected) can be split into independent fields and added or subtracted to get the modulo reduced number [5]. It is due to this reason that the $\{2^n-1, 2^n, 2^n+1\}$ set receives such great attention – its number theoretic properties allow efficient circuit realizations [9-16]. Henceforth, we refer to this set as the 3-moduli set (although there are other 3-moduli sets too [17]).

As stated in the introduction, reverse conversion efficiency is a complex function of all moduli rather than a single modulus. Even so, the 3-moduli set has enjoyed extremely efficient reverse conversion too [9, 10]. Previous research and a close analysis of the CRT has shown that efficient reverse converter design is possible when –

1. Closed forms exist for inverses [11].
2. Inverses have efficient Booth encoding (i.e. mostly zeros in the encoding).
3. Products of moduli are of the 2^n-type. Such products can significantly ease the final mod correction because end-around-carry (EAC) schemes may be employed [5, 9, 10].
4. Partial sum generation is possible by simple hardwired shifts [11] or simple shift-and-add operations.

Although reverse conversion costs can be notoriously unrelated to the nature of the moduli [8], having 2^n-type moduli definitely helps in achieving objective 4 and to a limited extent objective 3. As for the inverses, we will see in later sections that 2^n-type moduli do not necessarily imply 2^n-type inverses or inverses with efficient Booth encodings.

The preceding discussion shows that while increasing the *number* of moduli (and maintaining good balance), for a given dynamic range, results in greater speedup, the *nature* of the constituent moduli determine the conversion and correction costs. 2^n-type moduli sets, whatever their cardinality, guarantee extremely efficient forward conversion and modulo correction but they do not guarantee efficient reverse conversion by default. Combined with the fact that reverse conversion is undoubtedly the costliest of all conversion and correction penalties, in terms of both area and delay, we focus exclusively on this aspect while discussing our proposed moduli set.

2.2. Motivation for investigating the $\{2^n-1, 2^n, 2^n+1, 2^{n+1}+1\}$ set

For signal processing applications, the parallelism offered by the 3-moduli set is insufficient. The problem is *not* that this set delivers a speedup of "merely" 3 times. Indeed, from the VLSI point of view, reducing the delay of an arithmetic unit by 66% is a remarkable achievement! The real problem is that this set delivers a negligible *actual* speedup due to the multiply accumulate (MAC) nature of DSP arithmetic. Consider a MAC unit that performs a 16 by 16 bit multiply, followed by a 36-bit add. The 4 extra bits in the accumulation are usually needed for compensating rounding and other quantization effects. Clearly, such a MAC implemented in RNS needs a 36 bit dynamic range. With the 3-moduli set, this calls for moduli that leave 12 and 13-bit residues. If we now compare the overall speed-up, we note that instead of a 16 by 16 bit multiply, we do a 13 by 13 bit multiply. This isn't even a speed-up of 2! The problem is an essential one, arising from the fact that the dynamic range for n by n bit multiplication is $2n$. This is unlike addition, where the result's range stays at n (ignoring the possibility of overflow). Besides this real drawback, it is definitely important to investigate sets with higher cardinalities since there are always performance hungry applications desiring speed-ups larger than 3.

Set investigated	*Suitability and Remarks*
$\{2^n-1, 2^n, 2^n+1, 2^{n+2}-1\}$	Poor. Inelegant closed forms of inverses
$\{2^n-1, 2^n, 2^{n+1}+1, 2^{n+2}-1\}$	Poor. Same reason as above
$\{2^n-1, 2^n+1, 2^{n+1}+1, 2^{n+2}-1\}$	Poor. Same reason as above
$\{2^n, 2^n+1, 2^{n+1}+1, 2^{n+2}-1\}$	Fair. One of the inverses is 2^n-1; rest are not as good
$\{2^n-1, 2^n, 2^n+1, 2^{n+1}+1\}$, $n = 2m+1$	Good. Two of the inverses work out to be 2^n-1 and 2^{n-1} while the other two have reasonable closed forms
$\{2^n-1, 2^n, 2^{n+1}-1, 2^{n+1}+1\}$, $n = 2m+1$	Good. One inverse turns out to be 1; another 2^n-3; the other two are both equal to $(2^{n+1}-1)/3$.
$\{2^n-1, 2^n, 2^n+1, 2^{n+1}-1\}$, $n = 2m$	Good. Identical to properties of set $\{2^n-1, 2^n, 2^n+1, 2^{n+1}+1\}$

Table 1 - Comparison of various 4-moduli supersets

Hence, the key motivation behind expanding the 3-moduli set was to increase parallelism, while retaining reasonable balance and ease of forward conversion and modulo correction. This set was chosen from a list of potential candidates tabulated above. The first 4 sets had very cumbersome inverses and in some cases no tractable closed forms. The last three sets were found to be quite similar with respect to the previously discussed heuristics determining reverse conversion efficiency. Hence, we chose to investigate the set $\{2^n-1, 2^n, 2^n+1, 2^{n+1}+1\}$, which is a valid moduli set for odd n (for even n it violates the pair-wise prime constraint). With a suitable 4-moduli, 2^n-type-set chosen, the challenge was to construct an efficient reverse converter for it – preferably one which involved only simple shift-and-add operations.

3. A reverse converter for the $\{2^n-1, 2^n, 2^n+1, 2^{n+1}+1\}$ superset – the algorithm

We now describe the development of a VLSI efficient reverse conversion algorithm. In developing this algorithm, we have benefited from and employed the various techniques developed by researchers working on the 3-moduli set [9-16]. Most often, these techniques exploit the 2^n-type nature of the moduli set and allow simple and efficient hardware to realize modular operations. However, the additional modulus creates new and unforeseen problems. To combat these, we have developed three new techniques that allow efficient reverse conversion. In the derivation of a suitable arithmetic procedure, note how we exploit the knowledge and nature of the moduli and their inverses.

3.1. Closed form of the result using the Chinese Remainder Theorem (CRT)

Let $m_1 = 2^n - 1$, $m_2 = 2^n$, $m_3 = 2^n+1$, $m_4 = 2^{n+1} + 1$, with $n = 2m+1$ (m is positive and integral).

We denote the residue number to be reverse converted by X. Thus, we have –

$$X \equiv (r_1, r_2, r_3, r_4)$$

The inverses for the moduli work out to be -

$$m_1^{-1} = 2^{n-1} + (2/3)(2^{n-1} - 1),$$

$$m_2^{-1} = 2^n - 1,$$

$$m_3^{-1} = 2^{n-1},$$

$$m_4^{-1} = (2^n + 3) + (1/3)(2^n+1).$$

Invoking the Chinese Remainder Theorem (CRT),

$X + M.A(X)$

$= m_1^{-1}.\hat{M}_1.r_1 + m_2^{-1}.\ \hat{M}_2.r_2$

$+ m_3^{-1}.\ \hat{M}_3.r_3 + m_4^{-1}.\ \hat{M}_4.r_4$

$= m_1^{-1}.(2^n)(2^n+1)(2^{n+1}+1).r_1$

$+ m_2^{-1}.(2^n-1)(2^n+1)(2^{n+1}+1).r_2$

$+m_3^{-1}.(2^n)(2^n-1)(2^{n+1}+1).r_3 + m_4^{-1}.(2^n)(2^n-1)(2^n+1).r_4$

Since the lower n bits of X are identical to the residue mod 2^n i.e. r_2, we need not compute these bits. The remaining higher order bits are given by,

$$\left\lfloor \frac{X}{2^n} \right\rfloor = Y, \text{ say.}$$

Working this out from the closed form, we get,

$\left\lfloor \frac{X}{2^n} \right\rfloor + M.A(X)/2^n$

$= m_1^{-1}.(2^n+1)(2^{n+1}+1).r_1 + \left\lfloor \frac{m_2^{-1}\hat{m}_2 r_2}{2^n} \right\rfloor$

$+ m_3^{-1}.(2^n-1)(2^{n+1}+1).r_3 + m_4^{-1}.(2^n-1)(2^n+1).r_4$

$= m_1^{-1}.(2^n+1)(2^{n+1}+1).r_1$

$+ \{(2^n-1)(2^n+1)(2^{n+1}+1)-(2.(2^{2m}-1)+2^m)\}r_2$

$+ m_3^{-1}.(2^n-1)(2^{n+1}+1).r_3$

$+ m_4^{-1}.(2^n-1)(2^n+1).r_4$, or,

$= E$, for brevity.

Note that M is a multiple of 2^n and hence, we have omitted the floor function from the second term in the LHS. To get rid of the *M.A(X)* term, we re-write the equation above, taking mod $m_1m_3m_4$ of both sides,

$$\left| \left\lfloor \frac{X}{2^n} \right\rfloor + \frac{MA(X)}{2^n} \right|_{m_1m_3m_4} = \left\lfloor \frac{X}{2^n} \right\rfloor = Y = |E|_{m_1m_3m_4}$$

3.2. Calculating $|E|_{m_1m_3m_4}$

Two problems arise if we try evaluating $|E|_{m_1m_3m_4}$ (i.e. Y) directly. Firstly, division by 3 occurs in m_1^{-1} and m_4^{-1} and E cannot be evaluated by simple shift-and-add operations. Secondly, existing approaches like, Piestrak's Multiple-Operand-Modulo-Adders (*MOMA*s) [5] cannot be used to perform the modulo operation on E. This is because the width of the fields into which the input operands must be split in Piestrak's approach turns out to be $(2n)(n+1)$ – inapplicable for E which has only about $4n$ bits in all!

Hence we find Y by splitting the moduli product and applying the CRT i.e. we first find E_1 and E_2 where,

$$E_1 \equiv |E|_{m1m3} \text{ and } E_2 = |E|_{m4}$$

and then apply the CRT to retrieve E from E_1 and E_2.

There are two reasons for adopting this approach -

1. Although $|E|_{m1m3m4}$ is difficult to evaluate, E_1 and E_2 can be evaluated *very* efficiently.
2. Efficient evaluation of E_1 and E_2 would be useless without a fast algorithm to derive E from E_1 and E_2. Fortunately, this operation is fast because closed forms exist for the inverses of m_1m_3 and m_4.

In the following discussion, we shall use $M_1 = m_1m_3$ and $M_2 = m_4$.

3.3. Calculating E_1 and E_2

The expression for E_1 works out to be -

$E_1 = |E|_{m1m3} = |E|_{2^{2n}-1}$

$= |\ (2^{n-1})(2^n+1)r_1 - 2^n r_2 - (2^{n-1})(2^n-1)r_3\ |_{2^{2n}-1}$

This expression is evaluated very efficiently using (5, 2^{2n}-1) MOMAs that use an end-around-carry and CSA style architecture. For details on the techniques used for reduction modulo 2^{2n}-1, the reader is referred to [9,10].

The expression for E_2 turns out to be even simpler and is given by -

$$E_2 = |E|_{m4} = |E|_{2^{n+1}+1} = |2r_2 - 2r_4|_{2^{n+1}+1}.$$

If $r_2 > r_4$ i.e. if the expression is positive, it is guaranteed to be correct mod m_4. If it is negative, we simply add $2m_4$ to the expression, even though m_4 will suffice in some cases. This is to keep this unit fast and simple. The cost incurred is a 1-bit increase in the internal wordlength.

3.4. Combining E_1 and E_2 to get E

From the CRT,

$$E = \left| \hat{M}_1 \left| \frac{1}{\hat{M}_1} \right|_{M_1} E_1 + \hat{M}_2 \left| \frac{1}{\hat{M}_2} \right|_{M_2} E_2 \right|_{M1 \cdot M2}$$

$$= \left| M_2 \left| \frac{1}{M_2} \right|_{M_1} E_1 + M_1 \left| \frac{1}{M_1} \right|_{M_2} E_2 \right|_{M1 \cdot M2}$$

$$= \left| w_1 E_1 + w_2 E_2 \right|_{M1 \cdot M2} \qquad (3.1)$$

The inverses are calculated thus -

$$\left| \frac{1}{M_2} \right|_{M_1} = \frac{1}{3} \cdot (2^{2n} + 2^{n+1} - 2), \text{ and,}$$

$$\left| \frac{1}{M_1} \right|_{M_2} = 2^n + \frac{1}{3} \cdot (2^n - 2).$$

Hence,

$$w_1 = \frac{1}{3} \cdot (2^{2n} + 2^{n+1} - 2)(2^{n+1} + 1), \text{ and,}$$

$$w_2 = (2^n + \frac{1}{3} \cdot (2^n - 2))(2^{2n} - 1).$$

If we evaluate ***E*** using equation (3.1) above, we find that in spite of Booth recoding the weights w_1 and w_2, the evaluation is very costly since both w_1 and w_2 have approximately *n* non-zero bits after Booth recoding. Thus the multiplication cost is not only high, it also scales linearly with *n*.

To overcome this, we used an innovative technique. We note that although w_1 and w_2 have 'bad' Booth recoding, $(2w_1 - w_2)$ and (w_1+w_2) have very efficient, *n*-independent coding. This is evident in the expressions below -

$2w_1 - w_2 = 2^{3n+1} + 2^{2n} - 2^{n+1}$, (3 bits in Booth Recoding)

$w_1 + w_2 = 2^{2n+2} - 2$. (2 bits in Booth Recoding)

Hence equation (4.1) can be re-written thus -

$$E = \left| \frac{1}{3} \cdot [(2w_1 - w_2)(E_1 - E_2) + (w_1 + w_2)(E_1 + 2E_2)] \right|_{M1 \cdot M2}$$

$$= | K |_{M1M2}$$

Note that earlier on, we rejected direct evaluation of *E* because of division by 3 in the expressions for two modulo inverses. The reason was that the number of partial products in the subsequent multiplication would scale linearly with *n*. Here, the division by 3 incurs no such cost. This is because we exploit the fact that we can very efficiently divide a number by 3 when it is known, a-priori, to be a perfect multiple. The reader may wish to verify that a simple chain of subtractor like cells can perform such a-priori division. Furthermore, concepts similar to carry-selection or carry-lookahead can be used to modify this ripple-like structure to make it faster.

3.5. Performing the final mod operation

Thus far, we have been successful in finding fast and lean implementations for the various operations. But the final modulo operation poses a problem. This is because existing methods fail for unconventional moduli sets like the one under investigation. As we described in sec. 3.2, a widely used strategy is to break up the number to be reduced into fields [5]. Similar to the earlier case with *E*, this strategy does not succeed here since the field width for mod $(2^{2n}-1)(2^{n+1}+1)$ is $(2n)(n+1)$ which exceeds the width of *K* itself.

The other technique is to use ROMs [18]. This is also not applicable here since the "overflow" is of the same order as the valid range (overflow in this context is defined as the number of bits in the input operand minus the number of bits in the desired mod-corrected output). Worse still, the overflow is not fixed and varies *linearly* with *n* (~3*n*). Since the overflow bits are used to address the ROM, the latter's size increases *exponentially* with *n*. This makes the design highly non-scaleable.

We present two techniques that overcome this problem – one is an adaptation of the ROM-technique while the other is algorithmic. Before presenting the details, we derive the number of overflow bits. The dynamic range for the conversion is –

$L = M_1M_2 = (2^{2n}-1)(2^{n+1}+1) = (2^{3n-1} - 1 + 2^{2n} - 2^{n+1})$. Clearly, $2^{3n} - 1 > L > 2^{3n-1}$. Hence the first *3n-1* bits are guaranteed to be correct mod *L*. The maximum value of *K* can be derived in terms of *L*, as follows –

$$K = w_1E_1 + w_2E_2 = M_1^{-1}.M_2.E_1 + M_2^{-1}.M_1.E_2$$

$$\left\lceil \frac{K_{\max}}{L} \right\rceil = \left\lceil \frac{M_1^{-1}M_2E_{1\max} + M_2^{-1}M_1E_{2\max}}{M_1M_2} \right\rceil$$

$$= \left\lceil \frac{M_1^{-1}M_2(M_1-1) + M_2^{-1}M_1(2M_2-1)}{M_1M_2} \right\rceil$$

$$= \left\lceil \frac{M_1^{-1}M_2M_1 + 2M_2^{-1}M_1M_2 - M_1^{-1}M_2 - M_2^{-1}M_1}{M_1M_2} \right\rceil$$

$$= M_1^{-1} + 2M_2^{-1} + \left\lceil \frac{-M_1^{-1}M_2 - M_2^{-1}M_1}{M_1M_2} \right\rceil$$

$$= M_1^{-1} + 2M_2^{-1} + \left\lceil -\left(\frac{M_1^{-1}}{M_1} + \frac{M_2^{-1}}{M_2} \right) \right\rceil$$

$$\leq M_1^{-1} + 2M_2^{-1}$$

Note that we have accounted for not correcting E_2 completely. Now, $M_1^{-1} + 2M_2^{-1}$ is of the form $2^{3n-1} + k$ where $k \in [0, 2^{3n-1})$. Thus, we can expect a worst case overflow of 3*n* bits. This analysis can be tightened up a bit by deriving the exact number of bits in $K_{\max}$ and one might find that the overflow would be (3*n*-1) bits and *not* 3n bits. This difference is quite insignificant in both the schemes to be discussed below. This is the first sign of the robustness of the following methods – a one bit difference would be critical in a pure ROM implementation, as it would double the ROM size. In the following sections, we often omit the floor or ceil function while stating the bits needed to address the ROM. We do this to keep the estimates tractable and intuitive. It will be clear from the context that this does not affect the results significantly.

3.5.1. Reducing ROM requirements by the Multiple Split Tables (MST) approach. A pure ROM scheme would require approximately 2^{3n} x 3*n* x 2 bit ROM. For *n* = 8, this would be about 1024 Mbit i.e. 128 Mbyte of ROM for a dynamic range of 64 bits. This is clearly impractical.

A useful technique is to try and convert 2^{3n} into $n.2^3$. What this means is that we split the 3*n* bit overflow field into *n* fields of 3 bits each. ROM lookup is used for each of these 3-bit fields and the outputs from all the ROMs are added together. In such a scenario we would have to add *n* numbers, each of which is correct mod *L*, to get the result. Doing so causes an overflow again, but this time the number of overflow bits is approximately $\log_2(n+1)$. The final correction can again be performed using a ROM. The total ROM requirement is thus -

$n.2^3$ x $3n$ + $2^{\log(n+1)}$ x $3n$ x 2 bits, or,

$n.2^3$ x $3n$ + (n+1) x $3n$ x 2 bits

which is *quadratic* in *n* as compared to the pure ROM scheme which is *exponential* in *n*. Thus about 1 Kbyte of

ROM would suffice in place of the 128 Mbyte needed earlier. The cost incurred is the need for *n-1* large wordlength adders.

3.5.2. The Modified Division Algorithm (MDA). We now present what we believe is a new technique for mod calculation when the number of overflow bits is large and scales linearly with the dynamic range. In the MST approach outlined above, we adapted the lookup approach to reduce the size and growth rate of the ROM requirements. In the modified division algorithm, we adapt the classical, division based mod-extraction approach. For sake of efficiency, we do use some lookup, but the size of the ROM used is *linear* in *n* and the number of locations is a small *constant*.

The basic idea is straight-forward. The classical definition of the mod operation is –

$$|K|_L = K - L \cdot \left\lfloor \frac{K}{L} \right\rfloor.$$

The usual reason for not computing $|K|_L$ in this manner is the costly division operation. But we can adopt a hybrid approach in which we *approximate* the quotient ($\lfloor K/L \rfloor$) by a number that can be easily computed. Thus, in effect, we perform an operation given by -

$|K|_L = K - L \cdot P - \varepsilon(K)$,

where $P \in \left[\left\lfloor \frac{K}{L} \right\rfloor - c, \left\lfloor \frac{K}{L} \right\rfloor \right]$, c being a fixed positive constant and $\varepsilon(K)$ corresponds to a final ROM correction.

The expressions above simply mean that since the exact calculation of the quotient is costly, we produce a *pseudo-quotient*, *P*, which is guaranteed to be close to the quotient with a small tolerance, *c*. It is clear that $c = 0$ corresponds to perfect division and needs no ROM correction. As *c* grows larger, it becomes easier to calculate *P*, but the final ROM correction also grows. The worst case, where $c = \lfloor K/L \rfloor$, signifies a pure ROM approach. Our task is to find an optimum point between these two extrema. In other words, we must find a good balance between the cost of predicting and subtracting *L.P* from *K* as against the cost of the final ROM correction.

Before we explain how this goal is achieved, we point out the dependency between the ROM size and the tolerance *c*. Let the result of the operation *K - L.P* be denoted by $\hat{E}$. Then,

$$\hat{E}_{\max} = (K - L.P)_{\max} = (K - L.P_{\min})$$
$$= \left(K - L.\left(\left\lfloor \frac{K}{L} \right\rfloor - c\right)\right) = |K|_L + L.c$$

$$\Rightarrow \hat{E}_{\max} \le (c+1)L.$$

Hence the resultant number can have at most $\log_2(c+1)$ overflow bits. The ROM size is thus

$2^{\log(c+1)}$ x *3n* x *2* bits i.e. (c+1) x *3n* x *2* bits.

It is thus important that we not only keep *c* to a minimum, but also independent of *n*.

Let us study the quotient $\lfloor K/L \rfloor$, in order to obtain a suitable *P* and *c* from the quotient's closed form expression. The *division* by *L* is expressed as a multiplication by L^{-1}. The closed form expression for L^{-1} is derived as follows -

$$L^{-1} = \frac{1}{L} = \frac{1}{(2^{2n}-1)(2^{n+1}+1)} = \left(\frac{2^{-2n}}{1-2^{-2n}}\right).\left(\frac{2^{-(n+1)}}{1+2^{-(n+1)}}\right)$$

Since $n > 0$, both the products are infinite series and hence,

$L^{-1} = (2^{-2n} + 2^{-4n} + 2^{-6n} + 2^{-8n} + \dots)(2^{-(n+1)} - 2^{-2(n+1)} + 2^{-3(n+1)} - 2^{-4(n+1)} + \dots)$

From our previous analysis, we have determined that *6n-1* bits would be needed for *K*. Thus $K < 2^{6n}$, and we group the infinite series terms that are $< 2^{-6n}$ separately. With this consideration, we re-write the series expansion of L^{-1},

$L^{-1} = (2^{-2n} + 2^{-4n} + (2^{-6n} / (1- 2^{-2n}))\ )(2^{-(n+1)} - 2^{-2(n+1)} + 2^{-3(n+1)} - (2^{-4(n+1)} / (1+2^{-(n+1)}))\ )$

$= (2^{-(3n+1)} - 2^{-(4n+2)} + 2^{-(5n+3)} + 2^{-(5n+1)}\ ) - (2^{-(6n+2)} + 2^{-(7n+3)} + \frac{2^{-(6n+4)} - 2^{-(8n+4)}}{1-2^{-(n+1)}} + \frac{2^{-(7n+1)} + 2^{-(8n+2)} + 2^{-(9n+3)}}{1-2^{-2n}} + \frac{2^{-(10n+4)}}{(1-2^{-(n+1)})(1-2^{-2n})})$

The group of terms which when multiplied by *K* yield a number < 1 will be denoted by δ.

$$\left\lfloor \frac{K}{L} \right\rfloor = \lfloor KL^{-1} \rfloor$$
$$= \lfloor K.(2^{-(3n+1)} - 2^{-(4n+2)} + 2^{-(5n+3)} + 2^{-(5n+1)} + \delta) \rfloor$$
$$= \lfloor K.(2^{-(3n+1)} - 2^{-(4n+2)} + 2^{-(5n+3)} + 2^{-(5n+1)}) + K.\delta \rfloor$$

Thus,

$\lfloor K.2^{-(3n+1)} \rfloor + \lfloor -K.2^{-(4n+2)} \rfloor + \lfloor K.2^{-(5n+3)} \rfloor + \lfloor K.2^{-(5n+1)} \rfloor + \lfloor K.\delta \rfloor \le \left\lfloor \frac{K}{L} \right\rfloor$ and,

$\lfloor K.2^{-(3n+1)} \rfloor - \lfloor K.2^{-(4n+2)} \rfloor - 1 + \lfloor K.2^{-(5n+3)} \rfloor + \lfloor K.2^{-(5n+1)} \rfloor + \lfloor K.\delta \rfloor \le \left\lfloor \frac{K}{L} \right\rfloor$

$$\left\lfloor \frac{K}{L} \right\rfloor \le \lfloor K.2^{-(3n+1)} \rfloor + 1 + \lfloor -K.2^{-(4n+2)} \rfloor + 1 + \lfloor K.2^{-(5n+3)} \rfloor + 1 + \lfloor K.2^{-(5n+1)} \rfloor + 1 + \lfloor K.\delta \rfloor + 1$$

$= \lfloor K.2^{-(3n+1)} \rfloor + 1 - \lfloor K.2^{-(4n+2)} \rfloor - 1 + 1 +$
$\lfloor K.2^{-(5n+3)} \rfloor + 1 + \lfloor K.2^{-(5n+1)} \rfloor + 1 + \lfloor K.\delta \rfloor + 1$

We choose

$P = \lfloor K.2^{-(3n+1)} \rfloor - \lfloor K.2^{-(4n+2)} \rfloor + \lfloor K.2^{-(5n+3)} \rfloor + \lfloor K.2^{-(5n+1)} \rfloor - 1.$

Then,

$$P \le \left\lfloor \frac{K}{L} \right\rfloor \text{ and } \left\lfloor \frac{K}{L} \right\rfloor \le P + 5 \text{ (since } \lfloor K.\delta \rfloor = 0 \text{),}$$

i.e.

$$P \in \left[\left\lfloor \frac{K}{L} \right\rfloor - 5, \left\lfloor \frac{K}{L} \right\rfloor \right]$$

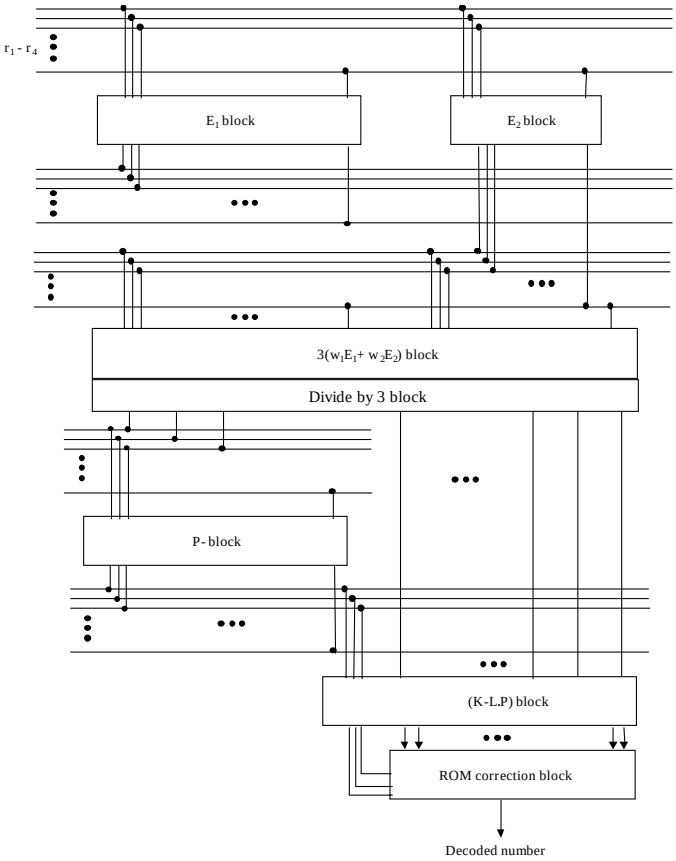
We have managed to find a suitable P and c. Note that P is obtained quite economically by shifting and adding 4 numbers. The value of $\lfloor K.2^{-x} \rfloor$ is simply what remains after we drop off the x least significant bits - a trivially performed operation.

Aside from the simplicity of calculating P, we have obtained a low value for c i.e. 5. Thus we only need to have a ROM with 6 locations, each $6n$ bits wide i.e. about $36n$ bits. This works out to be about 36 bytes for a dynamic range of approximately 64. The lookup requirement is now *linearly* dependent on n as opposed to quadratic dependency for MST and exponential for the pure ROM. The significance of this can be seen by comparing the 36 bytes for MDA to 128 Mbyte needed for a pure ROM approach and 1 Kbyte needed for the MST approach. Moreover, the number of locations stays at 6 regardless of n. The price paid is the computation of $L.P$ and its subtraction from K. But these costs are low since L is already known beforehand *and* it has an excellent Booth encoding – having only 4 bits. Hence, four shift-and-add operations allow the evaluation of $L.P$. With this, we have the complete algorithm for calculating the number given the 4 residues. At each step we have reduced the operations to simple shift-and-add type. It is now time to look at the overall architecture and evaluate the cost-performance of the various sub-units.

4. The reverse conversion architecture – area and delay

The algorithm developed above is now mapped to silicon. The figure in the adjacent column illustrates how this is done.

Note that we have adopted the modified division scheme to keep the ROM requirement low and also to illustrate a new approach to final modulo correction. The overall delay of the reverse converter is worked out next. This is done by dividing the circuit into sub-sections which are completely flow-through. From the figure, it is seen that till the K - $L.P$ generation, we have complete flow-through (we shall rectify this slight over-simplification in a while). This flow-through is broken by the need to wait for the MSBs to index the ROM and find the two correction factors. Note that the ROM correction block is composed of ROM and correction adders [5]



The time taken for the K-$L.P$ generation works out to

$\tau_{K\text{-}L.P} = \tau_K + \tau_{P(CSA)} + \tau_{K\text{-}LP(CSA)}$

$\tau_K = \tau_{K(CPA)} + \tau_{K(CSA)} + \max(\tau_{E1(CSA)}, \tau_{E2(CSA)})$

$= (5n+3) + 4 + \max(2, 1)$

$= 5n+9$ FA delays. (FA = Full Adder)

$\tau_{K\text{-}L.P} = (5n+9) + 3 + 1 = 5n + 13$ FA delays.

This delay derivation is simplistic since it does not consider the fact that the MSBs of K needs to be shifted $2n$ bits to the right when P is multiplied by L. This disturbs flow-through by $2n$ FA delays. Hence the real delay is

$\tau_{K\text{-}L.P(actual)} = (5n + 13) + 2n = 7n + 13$ FA delays.

Finally, we find the overall time as

$\tau_{total} = \tau_{K\text{-}LP(actual)} + \tau_{correction}$

$= (7n+13) + 3n = 10n+13$ FA delays.

Note that we have ignored multiplexing, ROM lookup and XOR delays. The objective of the delay analysis is to account for the most significant delays and also to see how delay scales with n. Also, the converter can be pipelined as deeply as needed, since it is built almost exclusively out of full adders and hardwired shifts.

To compute the area, we calculate the height and width in terms of a virtual grid. The grid spacing is an integral multiple of the minimum feature size λ, usually 8λ. Hence, a grid unit in 0.25µ technology would be 2µm.

The height of the converter is given by -

$h_{total} = h_{bus1} + \max(h_{E1}, h_{E2}) + h_{bus2} + h_K$

$+ h_{bus3} + h_P + h_{bus3} + h_{K\text{-}LP} + h_{ROM} + h_{correct}$
$= 4n + 2h_{FA} + (3n+5) + 4h_{FA} + (2n+2) + h_{FA}$
$+ (2n+3) + 3h_{FA} + h_{FA} + h_{FA}$
$= 11n + 12h_{FA} + 10$ grid units
(h_{FA} = height of a FA).
The width can be approximated as -
$w_{total} = w_K = 2.(5n+4)w_{FA} = (10n+8)w_{FA}$ grid units.
(w_{FA} = width of a FA)

5. Conclusion

In this paper, we proposed a new, and to the best of our knowledge the first, 4-moduli set $\{2^n\text{-}1, 2^n, 2^n\text{+}1, 2^{n+1}\text{+}1\}$, valid for odd values of *n*. This set was designed as an extension to the popular 3-moduli set $\{2^n\text{-}1, 2^n, 2^n\text{+}1\}$ in order to increase the parallelism of the ensuing arithmetic. While efficient forward conversion and residue arithmetic is guaranteed due to the 2^n-type nature of the moduli, we illustrate how reverse conversion can be performed mostly through shift-and-add operations. In developing the reverse converter, we have proposed three new techniques. Firstly, we reverse convert a linear combination of the moduli, rather than the moduli themselves. This results in achieving multiplicands with just 4 bits in Booth recoded form compared to roughly *n* bits originally. Secondly, we exploit the fact that division by 3, when the number is known to be a perfect multiple a-priori, is highly efficient provided that it is introduced at a suitable stage. Finally, we propose a Modified Division Algorithm (MDA) – an adaptation of the classical division based definition of the modulo operation. The MDA results in decreasing exponential ROM requirements to linear ROM requirements. We have also included a list of 4-moduli sets, aside from the one investigated, which show good promise. It is our sincere hope that this work will stimulate investigations of these and other candidate sets and will further enhance reverse conversion techniques suitable for 2^n-type supersets.

References

[1] M. Bhardwaj and A. Balaram, *Low Power Signal Processing Architectures using Residue Arithmetic*. Proc. Intl. Conf. Acoustics, Speech and Signal Processing, ICASSP'98, pp. 3017-3020, 1998.

[2] S. Szabo and R. I. Tanaka, *Residue Arithmetic and its Applications to Computer Technology*. New York: McGraw Hill, 1967.

[3] M. A. Soderstrand, W. K. Jenkins, G. A. Jullien and F. J. Taylor, Eds., *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing*. New York: IEEE Press, 1986.

[4] F. J. Taylor, *The Impact of Residue Arithmetic on Digital Signal Processing*. Proc. IASTED Int. Symp., 1985.

[5] J. Piestrak, *Design of Residue Generators and Multioperand Modular Adders Using Carry Save Adders*. IEEE Transactions on Computers, vol. 43, no. 1, pp. 68-77, Jan. 1994.

[6] T. Srikanthan, M. Bhardwaj and C. T. Clarke, *Area-Time Efficient VLSI Residue to Binary Converters*, IEE Proceedings – Computers and Digital Techniques, vol. 145, no. 3, pp. 229-235.

[7] T. Srikanthan, M. Bhardwaj and C. T. Clarke, *Implementing Area-Time Efficient VLSI Residue to Binary Converters*. Proc. IEEE Workshop on Signal Proc. Systems: Design and Implementation, pp. 163-172, 1997.

[8] M. Bhardwaj, *Residue Reverse Converters in VLSI*. Honours Year Thesis, School of Applied Science, Nanyang Technological University, Singapore, 1996.

[9] M. Bhardwaj, A. B. Premkumar and T. Srikanthan, *Breaking the 2n-bit Carry Propagation Barrier in Reverse Conversion for the $\{2^n\text{-}1, 2^n, 2^n\text{+}1\}$ Moduli Set*. IEEE Trans. on Cct. Syst.-I (TCAS-I), Vol. 45, No. 9, pp. 998-1002, 1998.

[10] S. J. Piestrak, *A High Speed Realization of Residue to Binary Number System Conversion*. IEEE Transactions on CAS II, vol. 41, no. 12, Dec. 1996, pp. 661-663.

[11] S. Andraos and H. Ahmad, *A New Efficient Memoryless Residue to Binary Converter*. IEEE Transactions on CAS, vol. 35, no. 11, Nov. 1988, pp. 1441-1444.

[12] G. Bi and E. V. Jones, *Fast Conversion Between Binary and Residue Numbers*. Electronic Letters, vol. 24, no. 19, Sept. 1988, pp. 1195-1197.

[13] P. Bernardson, *Fast Memoryless, Over 64 Bits, Residue to Binary Converter*. IEEE Transactions on CAS, vol. 32, no. 3, Mar. 1985, pp. 298 – 300.

[14] K. M. Ibrahim and S. N. Saloum, *An efficient Residue to Binary Converter Design*. IEEE Transactions on CAS, vol. 35, no. 9, Sept. 1988, pp. 1156-1158.

[15] A. Dhurkhadas, comments on *An Efficient Residue to Binary Converter Design*. IEEE Transactions on CAS II, vol. 37, no. 6, June 1990, pp. 849-850.

[16] F. J. Taylor and A. S. Ramanarayanan, *Efficient Residue to Decimal Converter*. IEEE Transactions on CAS, vol. 28, no. 12, Dec. 1981, pp. 1164-1169.

[17] A. B. Premkumar, M. Bhardwaj and T. Srikanthan, *High Speed and Low Cost Reverse Converters for the {2n-1, 2n, 2n+1} Moduli Set*. IEEE Trans. On Cct. Syst.- II (TCAS-II), Vol. 45, No. 7, pp. 903 – 908, 1998.

[18] B. Parhami and C. Y. Hung, *Optimal Table Lookup Schemes for VLSI Implementation of Input/Output Conversions and other Residue Number Operations*. In VLSI Signal Processing VII, J. Rabaey, P. M. Chau and J. Eldon, eds, New York: IEEE Press, 1994.