VLSI Costs of Arithmetic Parallelism: A Residue Reverse Conversion Perspective

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

This paper reports how VLSI cost metrics (area, delay, power) of residue reverse converters scale with the cardinality and dynamic range of moduli sets. The study uses CMAC reverse converters, reported previously by the authors to be the most efficient known to date in terms of area and delay. In all, 134 reverse converters with dynamic ranges from 32 to 120 bits and set cardinalities ranging from 4 to 20 are actually constructed and analyzed. It is seen that area, delay and power costs are cardinality insensitive once the cardinality exceeds a threshold (usually between five to eight). For cardinalities beyond this threshold, conversion costs are essentially dynamic range dependent. This insensitivity is explained in detail by noting the counterbalancing effects of the various sub-units of a CMAC reverse converter. Since practical implementations of RNS usually employ cardinalities beyond the abovementioned thresholds, the significance of this study is its conclusion that increasing the set cardinality in most implementations will have a marginal, if any, effect on VLSI reverse conversion costs.

1. Introduction

After a period of intense initial enthusiasm owing to the promise of truly high-performance parallel arithmetic, Residue Number Systems (RNS) [1] remained dormant as the difficulties of conversion into and out of the number system were realized. However, three recent trends have led to renewed interest – the growing ubiquity of digital signal processing [2,3], the long-awaited dawn of mainstream Multiple Valued Logic (MVL) and the growing need for power conscious arithmetic. DSP workloads are ideal for RNS as they are abundant in multiply accumulate (MAC) operations while sparing in the "RNS-unfriendly" operations of division and comparison. The use of MVL in commercial memories [4] signals the steady emergence of this paradigm. While density was the driver for MVL in memory, the already large and still increasing disparity in available devices and I/O [5] will also be a driver in logic. In MVL with the number of levels not equal to a power of two, weighted arithmetic can be unwieldy while residue arithmetic is both elegant and economical [6]. Finally, it has been shown recently that significant power reduction can be achieved if residue arithmetic is used judiciously [7].

While residue arithmetic, by itself, is carry-free, parallel and very hardware efficient, the conversion into and out of the system is not and this has prevented mainstream use. The problem of forward conversion is now well understood and schemes to construct efficient forward converters for general moduli exist [8]. This is not the case for reverse conversion, though. Research has been carried out for both specific [9, 10] and general sets [11-18], but the costs are still significant. Thus, reverse conversion is the main reason why so few RNS based systems have been constructed.

The key to efficient residue based systems is to balance the cost and benefits of arithmetic parallelism, as has been observed in some successful demonstrations of its use [19-21]. The benefit, in terms of speed-up, is easy to deduce in RNS – to a first order it is same as the cardinality of the moduli set, provided that the latter is well balanced. The cost of forward conversion can also be calculated to a high degree of accuracy, owing to Piestrak's work [8]. However, the impact of parallelism on reverse conversion costs has not been analyzed in detail. There are studies that report the cost of conversion as asymptotic growth rates of the dynamic range [14-16], but the effect of cardinality is not well understood.

In this paper, we analyze in detail, how the VLSI costs of reverse conversion change with increasing parallelism. Specifically, we study the area, delay, power and power density costs of reverse conversion with general moduli sets. For this we use the Compressed Multiply Accumulate (CMAC) converters which we proposed in earlier work [11-13]. Apart from being the first study to explore the cost of parallelism, we believe our work is also the first in discussing extremely large moduli sets with as many as 20 parallel channels.

The paper is organized as follows. Section 2 has an overview of residue number systems, the reverse conversion problem and CMAC converters. In Section 3 we discuss the methodology used to auto-generate the more than hundred reverse converters used in our study. The area costs of parallelism are in section 4, while the delay costs are analyzed in section 5. The effect of parallelism on power is studied in section 6. In section 7, we summarize the main results of this investigation.

2. Background

In this section, we provide a brief overview of residue number systems (RNS), the reverse conversion problem and Compressed Multiply ACcumulate (CMAC) reverse converters.

2.1. Residue Number System (RNS)

Let N be a non-negative integer in the range $[0,M)$, with $2^n < M < 2^{n+1}$. Consider the binary representation of N in the conventional weighted number system –

$$N \equiv \{b_{n-1}, b_{n-2}, \ldots, b_0\} = \sum_{k=0}^{n-1} b_k \cdot 2^k$$

In a residue system based on the *moduli-set* $\{m_1, m_2, ..., m_s\}$, such that

$$\prod_{k=1}^{s} m_k \geq M$$

N can be unambiguously represented by means of s residue digits,

$$N \equiv \{r_1, r_2, \ldots, r_s\} \text{where, } r_k = |N|_{m_k}$$

The number of moduli is called the *cardinality* and henceforth denoted by s. The count of numbers that can be represented i.e. M is called the *dynamic range* of the RNS. It is usually denoted by n (in bits). A dynamic range of n bits implies that the given RNS can represent at-least 2^n and not more than 2^{n+1} numbers.

RNS is a carry-free number system i.e. residue digits can be added, subtracted and multiplied individually without communicating with other residue digits. Hence, residue arithmetic is highly parallel. To ensure maximum efficiency and utilization, the computational load must be distributed evenly over all residue channels. In arithmetic, this implies that all channels must have identical or nearly identical wordlengths. This attribute of a moduli set is known as its *balance*. Quantitatively, the balance (in bits) is just the difference between the widths of the maximum residues left by the smallest and largest moduli in the set. When the difference is zero, the set is said to be *perfectly balanced*. In such a set, as also in sets that have small imbalances, the arithmetic speedup is proportional to the number of parallel channels i.e. the cardinality. For instance, a well balanced 3-moduli set will exhibit a speedup of almost 3 times over a normal, weighted number system when performing addition while a balanced 10-moduli set will exhibit a speed-up of close to 10. Second order effects, like modulo correction which must be performed in residue arithmetic due to finite wordlength constraints, depend on the moduli themselves and affect the speedup. Nevertheless, for reasonably balanced sets, cardinality is an excellent indicator of their potential parallelism.

2.2. Compressed Multiply Accumulate (CMAC) reverse converters

Given a number $X = \{r_1, r_2, \ldots, r_{s-1}, r_s\}$ in the residue number system, the *reverse conversion problem* entails determining the binary equivalent $b_{n-1}\, b_{n-2} \ldots b_0$ of X. Our earlier work dealt with constructing area-time efficient reverse converters for arbitrary moduli sets that overcame the limitations of the then existing proposals. We proposed a Compressed Multiply Accumulate (CMAC) technique to perform efficient reverse conversion. Actually fabricated CMAC converters have been shown to be 3-4 times more area and latency efficient than existing proposals. Details of these converters can be found in [11-13] and an example of their use in an actual ASIC in [19]. Here, we only provide an overview to aid understanding of the analysis in sec. 4.

Consider the following formulation of the Chinese Remainder Theorem –

$$N = \left| \sum_{k=1}^{s} \beta_k \cdot M_k \right|_M = \left| \sum_{k=1}^{s} X_k \right|_M = |X|_M$$

$$\text{where } \beta_k = \left| \frac{r_k}{M_k} \right|_{m_k} = \left| r_k \cdot M_k^{-1} \right|_{m_k}$$

$$X_k = \beta_k M_k$$

In conventional implementations, the reverse conversion proceeds in 4 steps –

1. **β-generation**: The β_ks are generated using modular multipliers i.e. $\beta_k = |r_k . M_k^{-1}|_{mk}$.
2. **Partial sum generation**: Partial sums are generated using conventional multipliers i.e. $X_k = \beta_k M_k$.
3. **Partial sum addition**: Partial sums are added together i.e. $X = \Sigma X_k$.
4. **Final modulo correction**: X is reduced modulo M to yield a number N i.e. $N = |X|_M$.

The cornerstone of the CMAC approach is the realization that steps (2) and (3) can be performed much more efficiently than current proposals. This is because existing schemes view the generation of X from β_ks as a multiply-accumulate (MAC) operation given by –

$$X = \sum \beta_k \cdot M_k$$

Since the $\boldsymbol{M_k}$s are known beforehand, we *need not* treat the above as a general MAC operation. All we need to do is –

1. Booth-encode the $\boldsymbol{M_k}$s, and using the width of the corresponding β_k, generate ***s*** *symbolic* partial products.
2. Take the ***s*** symbolic products and compress them to generate ***h*** numbers of varying widths that must be added using a carry save adder (CSA) tree.

Since our approach removes the redundancies of a general MAC implementation, we call it a C*ompressed* M*ultiply* AC*cumulate* (*CMAC*) approach. We illustrate this technique by the following example.

Consider a 6 moduli set {17, 23, 27, 29, 31, 32}. Let us represent the β_ks corresponding to these moduli as the 5-bit numbers B[4:0], C[4:0], D[4:0], E[4:0], F[4:0] and G[4:0]. Note that the maximum width of all residues is the same – i.e. this is a perfectly balanced set. After Booth-encoding the M_ks (called the *weights*) and performing a symbolic multiplication, we derive the *symbolic* partial sums. As an example, consider

$$\beta_2 \ * M_2 =$$

(C4 C3 C2 C1 C0) * (+0+0+00-000+0+00-0-00000), which yields the following symbolic partial product –

```
 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1 c4 c3 c2 c1 c0 -- -- -- -- --
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  1 -- -- -- -- --
 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1 c4 c3 c2 c1 c0 -- -- -- -- -- --
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  1 -- -- -- -- -- --
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- C4 C3 C2 C1 C0 -- -- -- -- -- -- --
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- C4 C3 C2 C1 C0 -- -- -- -- -- -- -- -- -- --
 1  1  1  1  1  1  1  1  1  1 c4 c3 c2 c1 c0 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
-- -- -- -- -- -- -- -- -- -- -- -- -- -- --  1 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
-- -- -- -- -- -- -- C4 C3 C2 C1 C0 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
-- -- -- -- -- C4 C3 C2 C1 C0 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
```

Compression/Compaction

```
 1  1  1 C4 C3 C4 C3 C4 C3 C2 C1 C0 c2 c1 C4 C3 C4 C3 C2 C1 C0 c4 c3 c2 c1 c0 -- -- -- -- --
-- -- --  1  1 C2 C1 C2 C1 C0 c4 c3 -- -- c0  1 C2 C1 C0 c4 c3 c2 c1 c0 --  1 -- -- -- -- --
-- -- -- -- --  1  1 C0  1  1 -- -- -- -- -- --  1  1 --  1  1 -- --  1 -- -- -- -- -- -- --
-- -- -- -- -- -- --  1 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
```

While C[4:0] denotes β_2 , c[4:0] denotes the bit-wise negation of β_2. This arises because of the '-1's in the Booth-encoding of M_2. In the figure above, note that the extra '1' below the complemented β_2 is needed since we are using 2's complement arithmetic. The string of 1's that must be added to the partial sum to achieve a 2's complement representation is called the *negative correction*.

After all the symbolic partial products are obtained and individually compressed as shown above, we compress them collectively as shown on the next page. The 14 compressed numbers written one below the other comprise the *CMAC table*. Note that the individual negative corrections (not shown for brevity) are combined into a single, overall *negative correction* as a result of the compaction. Also note that *X* has been expressed directly in terms of the β_ks. In other words, all that is needed to obtain *X* is the addition of the numbers in the *CMAC* table. This operation is termed *CMAC evaluation*. In this case, the height of the CMAC table (*h*) is 14. The addition of these 14 numbers of varying width is carried out using carry save adder (CSAs) trees followed by a carry propagate adder (CPA). Depending on the power, speed and area constraints, various other implementations of *CMAC* evaluation are possible. For instance, we can use a carry look-ahead adder (CLA) instead of a CPA. In this paper, however, CMAC implies a CPA based CMAC. The CMAC approach is used in our study since it yields area and performance figures that are superior to other converters [11] and also because it can handle arbitrary moduli sets.

3. The analysis methodology

We now explain the methodology used to arrive at the results in section 4. The auto-generation of the reverse converters is also explained in some detail. Some pointers to help the reader interpret the, often dense, plots are also included.

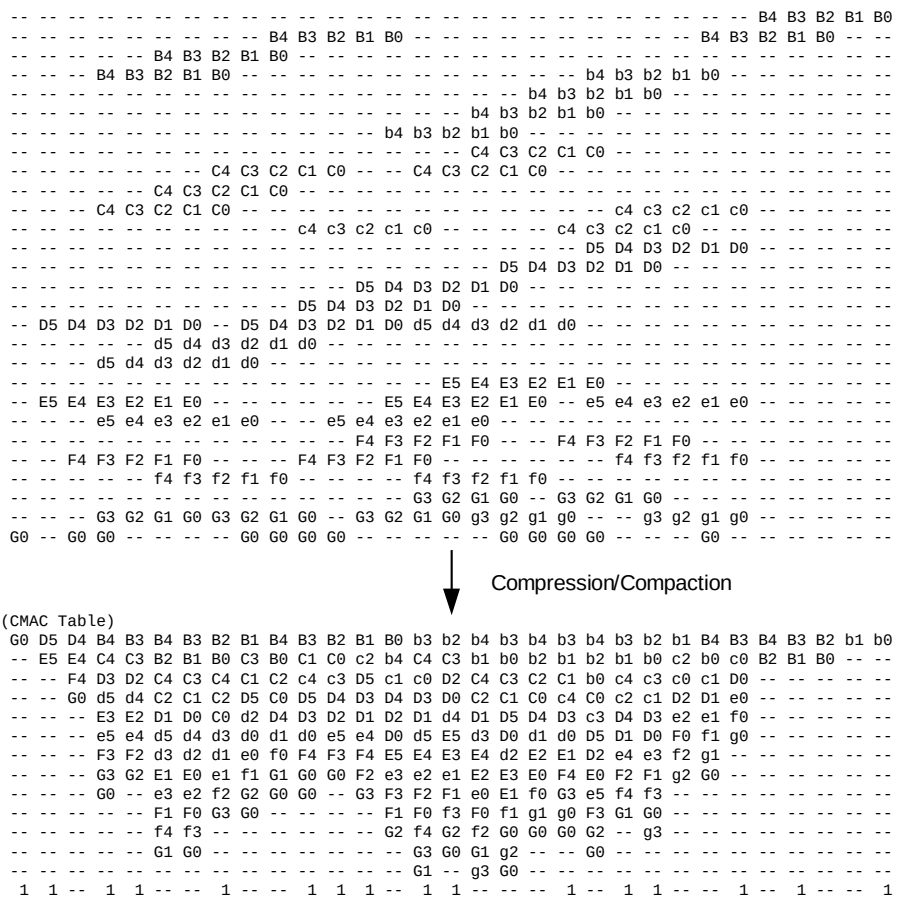
3.1. Choice of moduli set

In investigating the costs of parallelism, it would be ideal if the speedup achieved by a RNS was used as a measure of parallelism and various VLSI measures like area, delay and power used as cost metrics. This is not possible since the solution space of moduli sets that can obey a given (*speedup*, *dynamic-range*) constraint can be prohibitively large (but finite) and each element in this space represents a possibly unique cost. Hence the cost turns out to be a set, rather than a single value. As an example, consider the problem of specifying the cost of achieving a speedup of 6 for a dynamic range of 48 bits. There are many sets that would obey such a constraint. Hence, we can only speak of the cost of (6, 48) with reference to a particular set.

It is easy to see however, that given this huge space of possible moduli sets, we must choose one that delivers the lowest cost (while meeting the constraint). What is *not* easy is to exhaustively compute the cost metrics of each moduli set in this space.

We use heuristics to narrow down the search. The first heuristic is that the moduli set must be well balanced. This constraint can be formally enforced by stating that the maximum permissible imbalance is *d* bits. This narrows the solution space considerably. Once the balance constraint is imposed, the cardinality can be constrained using the (*imbalance*, *speedup*) pair. For reasonably low imbalance (say *d*=3), the speedup is proportional to cardinality. The constant of proportionality depends on the application at hand (and to a much lesser extent the choice of moduli themselves). Assume, for simplicity that we take the cardinality equal to the desired speedup. Hence, we have a 4-tuple constraint (*speedup*, *range*, *imbalance*, *cardinality*). The solution space satisfying this is significantly smaller than the original, but for large cardinalities it is still not amenable to an exhaustive search.

A final heuristic is used to narrow this space to a single solution. If we choose one modulus equal to a power of 2, say 2^v, we do not have to reverse convert the lower *v* bits – they are simply equal to the residue mod 2^v. This significantly reduces the reverse converter costs since the internal working range of the converter is reduced by *v* bits. Instead of the usual *n*+*e* bits, it is reduced to *n*+*e*-*v* bits (*e* denotes the additional bits needed to prevent internal overflows). Since the modulus is 2^v, the residue channel will have a width of *v* bits. But we want all channels to have the same width, which is approximately *range*/*cardinality*. Hence, we choose $v = \lceil n/s \rceil$.



With this choice of *v*, one modulus is fixed. We then choose (*s*-1)/2 moduli by traversing odd numbers beginning with 2^v+1 in an ascending order and ensuring that the pair-wise prime constraint is met. We then do the same for choosing the remaining moduli by starting at 2^v-1 and proceeding in a descending order. The reason we choose moduli around this 2^v modulus is because it allows the inclusion of moduli of the form 2^v+1 and 2^v-1, which make residue channels and forward conversion more efficient.

Although the above procedure is based on heuristics, it is totally deterministic. The number theoretic interactions are too complex and range and cardinality dependent to claim that this procedure yields the optimal moduli set. However, it does yield an upper bound. More importantly, comparing this solution to hand optimized moduli-sets shows that it yields reasonably tight upper bounds. For instance, we have reported extremely optimal converters with areas of less than 2 mm^2 [11]. The heuristic solution yields an area of close to 5-6 mm^2 – which is quite remarkable since both the moduli set and the VLSI implementation in the heuristic solution were auto-generated.

3.2. Use of auto-generators

To obtain conclusive results, we analyze reverse conversion for dynamic ranges from 32 to 120 bits (with steps of 8 bits) and cardinalities ranging from 4 to 20. The reason we do not consider lower cardinalities is two-fold. Firstly, most practical implementations seldom use 2 or 3 moduli sets. Secondly, reverse conversion costs for such sets are well known [9,10]. Of the 204 possible (*cardinality*, *range*) pairs, several are invalid due to the finite number of pair-wise prime numbers less than a given number. For instance, there is no possible (20, 32) set since we cannot find 20 pair-wise prime numbers that when multiplied yield a number less than 2^{33}. When we eliminate such cases, we end up with 134 valid tuples.

For each of these tuples, we arrive at a close to optimal moduli set using the procedure outlined in sec. 3.1. The moduli set thus determined is fed into an auto-generator that can build the entire CMAC converter. It is from this auto-generated reverse converter that we obtain the VLSI metrics like delay and area. The auto-generator computes the inverses, Booth encodes the weights, generates the CMAC table, compresses it, constructs the layout and finally carries out physical compaction of the CMAC converter.

3.3. VLSI implementation specific terminology

In section 4, the following terminology is used without qualification –

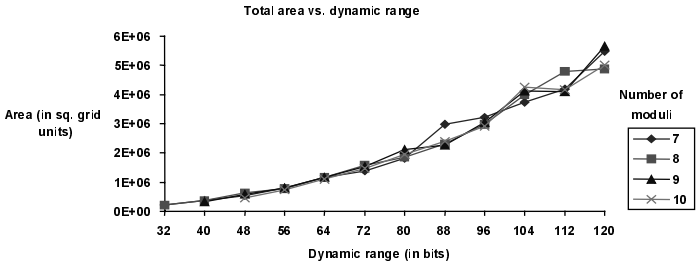
a. *Height*: Refers to the dimension along the data-flow. Units are in terms of a full-adder's height.
b. *Width*: The dimension perpendicular to the data-flow. Unit is full adder (FA) width.
c. *Delay*: Units are always full adder delay.
d. *CSA Tree/CMAC CSA*: This is a tree of CSAs which compresses the CMAC table yielding a sum and carry output.
e. *The CMAC CPA*: The CPA that accepts the sum and carry output from the CSA tree and performs the carry propagation.
f. *Modulo Correction CPA*: Refers to the CPA that performs the final addition after the ROM lookup is performed.
g. *Reflow*: The CMAC CSA tree determines the converter width, which is usually larger than the combined width of the *s* ß-generators. Hence, the latter are "reflowed" to occupy the remaining width. This can significantly decrease the height of the ß-generators and hence their area contribution.
h. *Power*: Is calculated for worst case conditions i.e. converter is operating at maximum operating clock frequency (MOCF) which is simply the inverse of converter delay. No glitching analysis is done and switching activity is taken to be a constant – independent of input statistics. The units are abstract.
i. *Power Density*: Is simply the power (as defined above) divided by total converter area.
j. *CMAC Height/CSA-Tree Height*: This refers to the height of the CMAC table (*h*).
k. *Cardinality*: As defined earlier. We often use *s* instead of cardinality at places.
l. *Dynamic Range*: As defined earlier. We often use *n* instead of dynamic range for brevity.
m. *ß-height*: Instead of the usual modulo multiplication, we use an optimized technique called bit-unfolding [11] to construct the ß-generators. The maximum of all the *s* ß-generator heights (*after* they have been *reflown*) is termed the *ß-height*.

Some useful pointers about the curves presented in the following section –

a. When plotting a cost metric (like area, delay etc.) against dynamic range, a set of 17 curves results corresponding to the possible cardinalities (4 to 20). In several cases the curves are too close to be distinguished and hence the legend (which correlates the curve with the cardinality) is omitted.
b. When data points clutter graphs and make it difficult to observe the major trend, they are omitted.
c. Due to number theoretic interactions, some curves display outlier points. This happens because once in a while a certain combination of moduli results in weights that have especially efficient Booth encoding.
d. A common phenomenon underlies many of the cost vs. cardinality curves in the next section. As we increase the cardinality, the value of *v* decreases. This in turn causes the internal working range of the converter to *increase* (recall that the internal working range is $n+e-v$ bits). This leads to an *increase* in the cost of CMAC evaluation.

4. Area analysis of CMAC reverse converters

The increase in area with the dynamic range is shown in the following graph –



Graph 1 - Area vs. dynamic range

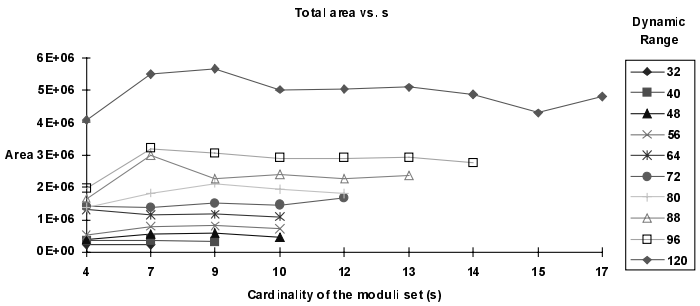
The area for each dynamic range has been calculated for different cardinalities. From these curves, note that –

1. The area grows at close to quadratic rate. In earlier work, we have shown the area complexity to be

$O(n^2 \log n)$ [12,13]. There is a remarkable agreement in the curve here and the predicted rate, although the latter was calculated using fastest possible VLSI elements like CLAs and not the slower CPAs used here.

2. The differences for varying cardinalities are *not* pronounced. To a first order it seems that the area is dependent solely on the dynamic range.

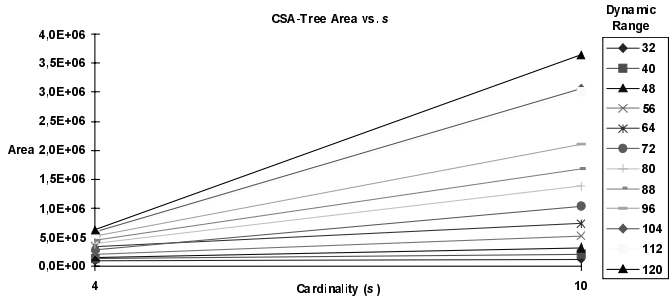
To see this last point in greater detail, we look at the next graph –



Graph 2 Area vs. cardinality

Note that there is generally an increase in the area from 4 to 7 moduli. But after that the area stays remarkably unchanged. Analysis reveals that this is due to two reasons –

1. Between s = 4 and s = 6-8, there are large changes in the β-generator height and CSA tree height. After about 7-8 moduli, the bits/moduli have decreased to a point below which they are relatively insensitive to changes. Of course, large dynamic ranges become truly area insensitive at higher cardinalities. This can be seen by comparing the 120-bit curve with the other low cardinality curves.
2. Increases in the CSA tree area are counterbalanced by decreases in the β-generator area. To see this, we look at the plots of tree area and ß-generator area.

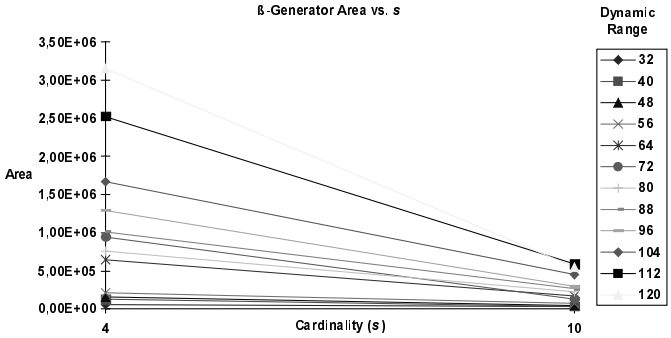


Graph 3 - CSA Area vs. Cardinality

As the cardinality increases, the internal working range of the converter also increases. This in turn causes an increase in the CSA tree width, which is proportional to the internal wordlength. This explains the increase in area. But observe the area of the β-generator (graph 4).

The area *decreases* with cardinality because –

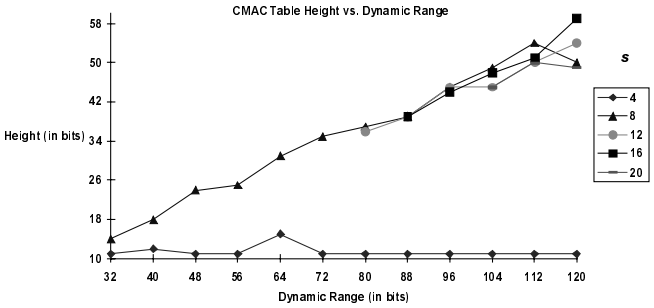
1. As mentioned earlier, the bits/moduli decrease and hence small β-generators result.
2. The increased width of the CSA trees allows for more space to *re-flow* the β-generators.



Graph 4 - β-generator area vs. set-cardinality of a CMAC reverse converter

Another important statistic is the height of the CMAC table (h) since it determines the height and delay of the CSA trees in turn. We have charted CMAC height versus the dynamic range below. The graph shows that the CMAC height is proportional to the dynamic range (interestingly, for sufficiently large cardinality, it is very close to half the dynamic range). It is helpful to recall the factors that affect the CMAC height –

a. The number of operands that are compressed (this number is proportional to s).
b. The width of the operands.
c. The number and location of non-zero bits in the Booth-encoded weights (M_ks).

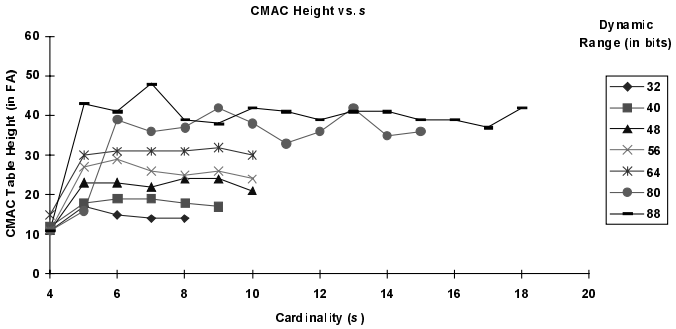


Graph 5 – CMAC height vs. dynamic range

While the second factor clearly suggests that the height should increase for increasing dynamic ranges with a fixed cardinality, it is not easy to predict how the third factor affects the height. A proof of this is marked difference between the curves for s=4 and general s. In the latter case, the selection of uncorrelated moduli sets led to a linear increase in height with dynamic range. For the case of cardinality four, the sets generated are of the form $\{2^m-3, 2^m-1, 2^m, 2^m+1\}$ where m is roughly $n/4$. For this selection, it turns out that the distance between the non-zero bits is greater than the width of the generated ß's, leading to excellent compression and a constant, range-independent height.

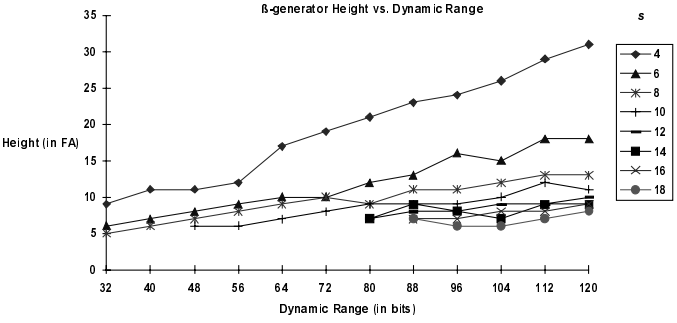
In graph 6 below, which explores how the CMAC height (h) changes with cardinality, we notice that after an initial increase (from 4 to 6), it is fairly fixed and

dependent on the choice of the moduli. This clarifies an important point about the interaction of the three factors that determine this height. From factor (a), we should expect a linear increase in height as *s* increases. This linear increase will only result if the symbolic products get stacked vertically on one another in the CMAC table. The curves above show that this does not happen – the decrease in widths due to increasing cardinality combined with uniformly dispersed non-zero bits in the Booth encoded weights negate the effect of factor (a), resulting in no significant height increase. In fact, we cannot even claim that the height is strictly monotonic in *s* – several sets with high cardinality show heights that are clearly *lower* than those exhibited by sets with smaller cardinalities.



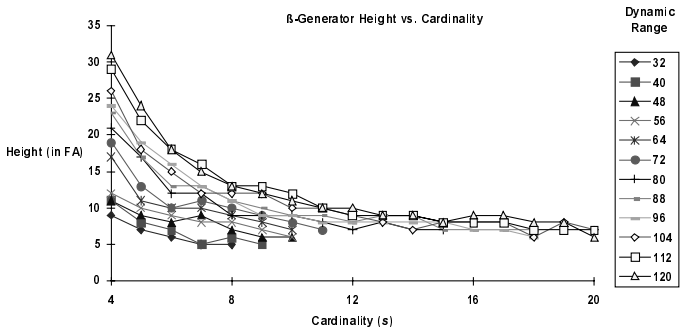
Graph 6 - CMAC height vs. cardinality

Having analyzed the CMAC height in detail, the effect of cardinality is studied on the ß-generators. We begin with how ß-generator height (ß-height) varies with dynamic range –



Graph 7 – β-generator height vs. dynamic range

Note that for constant *s*, the height generally grows proportionally to *n*. Recall that the core of the ß-generation is a bit unfolded modulo multiplier. As the range increases for a given *s*, the width of the input operands to this multiplier increases proportionally. This explains why the height of the ß-generator, which is proportional to the operand width, is proportional to *n* too. Again, the situation is not that simple – certain moduli can have extremely compact ß-generators owing to special number theoretic properties. However, the analysis remains valid because the height of the ß-generator section is determined by the height of the worst case ß-generator and it is provably impossible to construct high cardinality sets where *all* moduli have compact ß-generators [13].



Graph 8 - β-generator height vs. set-cardinality

Observe the unmistakable *reduction* of the β-generator height with increasing cardinality. On closer examination, it turns out that the curves for a given dynamic range are proportional to the reciprocal of *s*. The reason for this is the same as we presented earlier – the height of the ß-generator is proportional to the bits/modulus, which in turn varies inversely as *s* for fixed dynamic ranges.

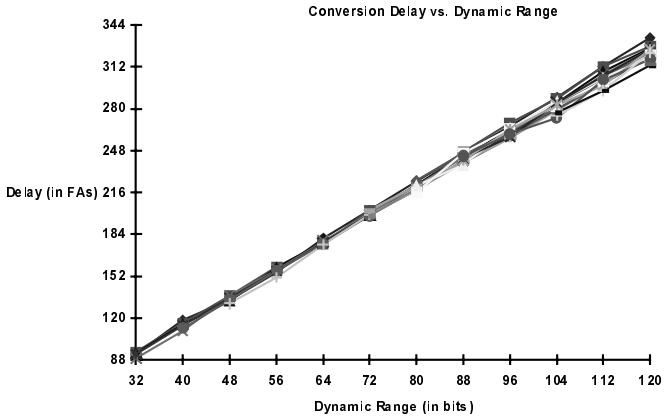
With this we conclude the section on the effect of cardinality on area –

1. Converter area rises with *n* at a close to quadratic rate (graph 1).
2. Converter area is relatively insensitive to the number of moduli for high cardinality sets (graph 2). This is because the increase in CMAC evaluation area with growing cardinality is countered by decreases in ß-generation area (graphs 3 and 4).
3. CMAC height grows linearly with dynamic range (graph 5) but is quite insensitive to cardinality (graph 6). CMAC height becomes *range insensitive* if moduli sets for the different ranges are instantiated from a fixed class, as happened to be the case for *s*=4 (graph 6).
4. β-generator height grows proportionally to the dynamic range for a given cardinality (graph 7). The height falls at a rate inversely proportional to the cardinality of the set for fixed dynamic range (graph 8).

5. Delay analysis of CMAC reverse converters

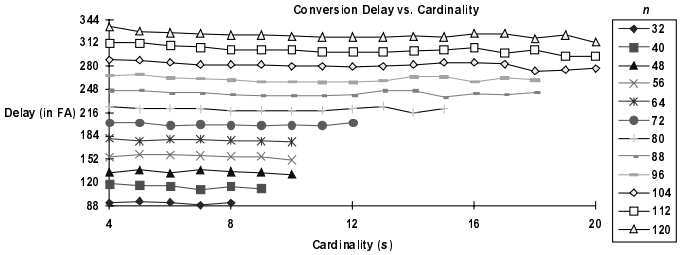
We now study how reverse conversion delay varies with the dynamic range and with increasing number of moduli. In graph 9 below, it is seen that the total delay is proportional to the dynamic range. This is expected since this is the CPA CMAC version and the CPA delay is proportional to the dynamic range.

The more interesting and unexpected result is that conversion delay is literally independent of the cardinality, as is clear from the graph 10 below. Recall the four main contributors to conversion delay – ß-



Graph 9 – Reverse converter delay vs. dynamic range

generation, CMAC CSA addition, CPA following CMAC evaluation and finally mod-correction. The last three

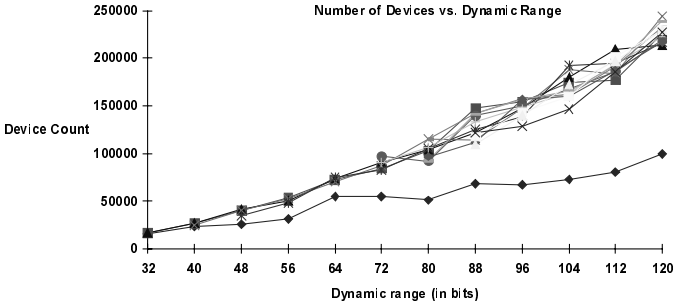


Graph 10 – Reverse converter delay vs. cardinality

delays increase slowly with cardinality, mainly due to increasing internal dynamic range. The ß-generators however, show a steep decrease in delay as cardinality increases. This explains the extraordinary invariance of the delay curves.

6. Power and power density analysis of CMAC reverse converters

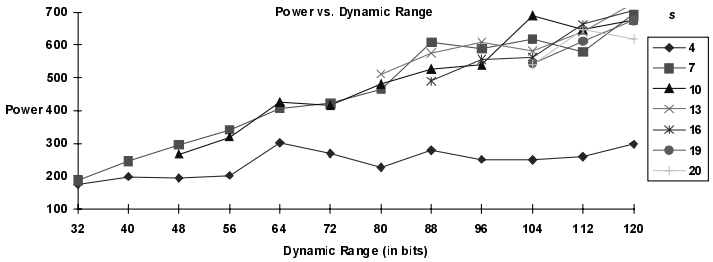
We shall now discuss the dependence of power and power-density on the dynamic range and the cardinality. Before looking at the plot of power against dynamic range, we first study the plot of devices vs. dynamic range–



Graph 11 – Number of devices vs. dynamic range

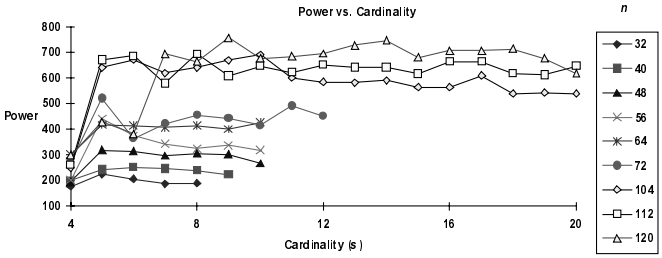
The CMAC converters exhibit high locality. Hence, gate and not interconnect capacitance dominates. Ignoring the routing capacitance, a good way to visualize the increase in converter capacitance with dynamic range is to plot the number of devices versus dynamic range, as we have done above. Note that the number of transistors rises at a rate slightly higher than linear.

When power is plotted against the dynamic range (graph 12), we see that power rises at a sluggish linear rate. This happens because both the capacitance and delay increase with the dynamic range at more or less the same rate. Since the maximum operating clock frequency (MOCF) varies inversely as the delay, the two factors act in opposing directions. It turns out that the rate of increase of gate capacitance is higher than the rate of decrease of the MOCF. Hence, the overall power *increases* with *n*.



Graph 12 – Conversion power vs. dynamic range

Next, we graph the effect of cardinality on power. Again, the curves are quite flat, suggesting that there is no significant power penalty in increasing the cardinality beyond 5 or 6 moduli. The reason for this is two-fold. The first is that the increase in capacitance as we increase cardinality is minimal. Secondly, the MOCF also remains more or less unchanged with cardinality since the delay variation with cardinality is minimal (as we have seen in graph 10).



Graph 13 – Conversion power vs. cardinality

Power density is an important VLSI metric because it determines the power distribution and the heat flux. High power density is bad news for the designer since electromigration may set in. We have not included the graphs due to lack of space. These graphs show that power density falls with increase in dynamic range. This is because the conversion area increases at a higher than quadratic rate while conversion power only increases at a linear rate. Hence, the power per unit area varies inversely as the dynamic range. Like power, power-density is also seen to be extremely insensitive to cardinality.

We conclude that increasing cardinality for a given dynamic range has no significant power or power density penalties.

7. Conclusion

In this paper, we analyzed the VLSI costs of increasing arithmetic parallelism in the context of reverse conversion in residue number systems (RNS). To carry out this investigation, more than hundred reverse converters that spanned dynamic ranges from 32 to 120 bits and cardinalities from 4 to 20 were auto-generated. The effect of increased set cardinalities on area, delay, power and power density of these converters was observed. The most important conclusion of this study was that the VLSI costs of greater parallelism in residue reverse converters are minimal – once the set cardinality crosses 5-6. This behavior was explained by analyzing the interaction between the various units in a reverse converter. It was shown that as the cardinality increases, the complexity of the mod-multipliers (ß-generators) decreases, while the complexity of partial sum generation, addition (CMAC evaluation) and modulo correction increases. It turns out that these opposing trends cancel each other out, leading to cardinality insensitive costs beyond 5-6 moduli. We also showed that there is no significant degradation in the conversion delay as cardinality is increased. Finally, while power rises with increasing dynamic ranges, power density actually falls as area increases at a higher rate than power. Also, both power and power density are insensitive to cardinality beyond thresholds similar to those reported for area and delay.

We hope that this comprehensive analysis of VLSI reverse conversion costs, especially with regard to cardinality, will be useful to researchers who are investigating highly parallel arithmetic and to system designers who are evaluating the use of RNS in practical systems.

References

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

[2] 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.

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

[4] Intel Inc., *MultiLevel Cell Technology – An Evolutionary Memory Technology with Revolutionary Benefits*. http://developer.intel.com/design/flcomp/isfbgrnd.htm.

[5] A. Osseiran, *Challenges in Testing Next Generation ASICs*. In Advanced CMOS and BiCMOS IC Design. Ecole Polytechnique Federale de Lausanne, 1998.

[6] M. Soderstrand, *Personal Communication*. Intnl. Conf. on Acoustics, Speech and Signal Processing, ICASSP'98, Seattle, May 1998.

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.

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

[14] F. Barsi and M. C. Pinotti, *A Fully Parallel Algorithm for Residue to Binary Conversion*. Information Processing Letters, vol. 50, pp. 1-8, 1994.

[15] G. Alia and E. Martinelli, *A VLSI Algorithm for Direct and Reverse Conversion from Weighted Binary Number System to Residue Number System*. IEEE Trans. on CAS, vol. 31, pp. 1033-1039, Dec. 1984.

[16] R. M. Capocelli and R. G. Carlo, *Efficient VLSI Networks for Converting an Integer from Binary System to Residue Number System and Vice-Versa*. IEEE Trans. on CAS, vol. 35, pp. 1425-1430, Nov. 1988.

[17] B. Chakraborti, J. S. Soundararajan and A. L. N. Reddy, *An Implementation of Mixed-Radix Conversion for Residue Number Applications*. IEEE Transactions on Computers, vol. 35, no. 8, pp. 762-764, Aug. 1986.

[18] K. Koc, *A Fast Algorithm for Mixed Radix Conversion in Residue Arithmetic*. Proc. 1989 IEEE Intnl. Conf. on Computer Design (ICCD), pp. 18-21, 1989.

[19] A. Deodhar, M. Bhardwaj, T. Srikanthan and C. T. Clarke, *Designing Efficient Residue Arithmetic Based VLSI Correlators*. Proc. Intl. Conf. Acoust., Speech and Signal Proc., ICASSP'98, pp. 3021-3024, 1998.

[20] M. Bhardwaj and B. Ljusanin, *The Renaissance: A RNS-based, Application Specific Vector Co-Processor for Embedded DSP Applications.* Presented at the 32nd Asilomar Conf. on Signals, Systems and Computers, Nov. 1998.

[21] R. Shanbhag and R. E. Siferd, *A Single Chip Pipelined 2-D FIR Filter Using Residue Arithmetic*. IEEE Journal of Solid State Circuits, vol. 26, no. 5, pp. 796-805, May 1991.