importance measurement and a method to embed the importance measurement into computation datapath in order to realize unequal error tolerance. Under this unequal error tolerance framework, we further developed approaches to use voltage overscaling in memory systems of trellis decoders. Effectiveness of such an unequal error tolerance framework and the developed techniques have been successfully demonstrated using computer simulations.

REFERENCES

[1] A. P. Chandrakasan and R. W. Brodersen, "Minimizing power consumption in digital CMOS circuits," *Proc. IEEE*, vol. 83, no. 4, pp. 498–523, Apr. 1995.

[2] R. Gonzalez, B. M. Gordon, and M. A. Horowitz, "Supply and threshold voltage scaling for low power CMOS," *IEEE J. Solid-State Circuits*, vol. 32, no. 8, pp. 1210–1216, Aug. 1997.

[3] G. Karakonstantis, N. Banerjee, K. Roy, and C. Chakrabarti, "Design methodology to trade off power, output quality and error resiliency: Application to color interpolation filtering," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, Nov. 2007, pp. 199–204.

[4] N. R. Shanbhag, "Reliable and energy-efficient digital signal processing," in *Proc. Des. Autom. Conf.*, Jun. 2002, pp. 830–835.

[5] S. Das, D. Roberts, S. Lee, S. Pant, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "A self-tuning DVS processor using delay-error detection and correction," *IEEE J. Solid-State Circuits*, vol. 41, no. 4, pp. 792–804, Apr. 2006.

[6] J. P. Fishburn, "Clock skew optimization," *IEEE Trans. Computers*, vol. 39, no. 7, pp. 945–951, Jul. 1990.

[7] R. B. Deokar and S. S. Sapatnekar, "A graph-theoretic approach to clock skew optimization," in *Proc. IEEE Int. Symp. Circuits Syst.*, Jun. 1994, pp. 407–410.

[8] J. L. Neves and E. G. Friedman, "Optimal clock skew scheduling tolerant to process variations," in *Proc. Des. Autom. Conf. (DAC)*, Jun. 1996, pp. 623–628.

[9] Y. Liu, T. Zhang, and J. Hu, "Low power trellis decoder with overscaled supply voltage," in *Proc. IEEE Workshop Signal Process. Syst. (SiPS): Des. Implementation*, 2006, pp. 205–208.

[10] A. Agarwal, V. Zolotov, and D. T. Blaauw, "Statistical timing analysis using bounds and selective enumeration," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 22, no. 9, pp. 1243–1260, Sep. 2003.

[11] H. Chang and S. S. Sapatnekar, "Statistical timing analysis under spatial correlations," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 24, no. 9, pp. 1467–1482, Sep. 2005.

[12] Y. Lu, C. N. Sze, X. Hong, Q. Zhou, Y. Cai, L. Huang, and J. Hu, "Register placement for low power clock network," in *Proc. Asia South Pacific Des. Autom. Conf. (ASP-DAC)*, Jan. 2005, pp. 588–593.

[13] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. Inform. Theory*, vol. 42, no. 3, pp. 429–445, Mar. 1996.

[14] J. H. Han, A. T. Erdogan, and T. Arslan, "High speed Max-Log-Map turbo SISO decoder implementation using branch metric normalization," in *Proc. IEEE Comput. Soc. Annu.Symp. VLSI*, May 2005, pp. 173–178.

[15] R. Hegde and N. R. Shanbhag, "Soft digital signal processing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 9, no. 6, pp. 813–823, Dec. 2001.

[16] Y.-N. Chang, H. Suzuki, and K. K. Parhi, "A 2-Mb/s 256-state 10-mW rate-1/3 Viterbi decoder," *IEEE J. Solid-State Circuits*, vol. 35, no. 6, pp. 826–834, Jun. 2000.

[17] G. Feygin and P. Gulak, "Architectural tradeoffs for survivor sequence memory management in Viterbi decoders," *IEEE Trans. Commun.*, vol. 41, no. 3, pp. 425–429, Mar. 1993.

[18] Li H.-L and C. Chakrabarti, "A new architecture for the Viterbi decoder for code rate k/n," *IEEE Trans. Commun.*, vol. 44, no. 2, pp. 158–164, Feb. 1996.

[19] C.-C. Lin, Y.-H. Shih, H.-C. Chang, and C.-Y. Lee, "Design of a power-reduction Viterbi decoder for WLAN applications," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 52, no. 6, pp. 1148–1156, Jun. 2005.

[20] F. J. Kurdahi, A. M. Eltawil, Y.-H. Park, R. N. Kanj, and S. R. Nassif, "System-level sram yield enhancement," in *Proc. Int. Symp. Quality Electron. Des.*, Mar. 2006, p. 6.

Fast Scaling in the Residue Number System

Yinan Kong and Braden Phillips

***Abstract*—A new scheme for precisely scaling numbers in the residue number system (RNS) is presented. The scale factor K can be any number coprime to the RNS moduli. Lookup table implementations are used as a basis for comparisons between the new scheme and scaling schemes from the literature. It is shown that new scheme decreases hardware complexity compared to previous schemes without affecting time complexity.**

***Index Terms*—Computational complexity, digital arithmetic, table lookup, residue arithmetic.**

I. INTRODUCTION

A. Residue Number System and Scaling

The residue number system (RNS) provides a means for efficient multiplication and addition of integers; however, scaling within RNS is less efficient and this problem has long prevented wider adoption of RNS. In this context, scaling an integer X means reducing its word length by dividing by a constant K

$$Y = \left\lfloor \frac{X}{K} \right\rfloor. \tag{1}$$

In binary arithmetic, K is usually chosen to be a power of 2 such that word length reduction is achieved by simply truncating a number's binary representation. There is no equivalent operation in RNS with the consequence that a result accumulated through a sequence of multiplications [as is often the case in digital filters or multiple-point fast Fourier transfers (FFTs)] can grow in word length until it overflows the dynamic range of the RNS.

An RNS [1] is characterized by a set of N coprime moduli $\{m_1, m_2, \ldots, m_N\}$. In the RNS, a number X is represented in N channels: $X = \{x_1, x_2, \ldots, x_N\}$, where x_i is the residue of X with respect to m_i, i.e., $x_i = \langle X \rangle_{m_i} = X \bmod m_i$. Within the RNS there is a unique representation of all integers in the range $0 \leq X < M$, where $M = m_1 m_2, \ldots, m_N$. M is therefore known as the dynamic range of the RNS. Two other values, M_i and $\left\langle M_i^{-1} \right\rangle_{m_i}$ are commonly used in RNS computations and are worth defining here. $M_i = (M/m_i)$ and $\left\langle M_i^{-1} \right\rangle_{m_i}$ is its multiplicative inverse with respect to m_i such that $\left\langle M_i \times M_i^{-1} \right\rangle_{m_i} = 1$.

If X, Y, and Z have RNS representations given by $X = \{x_1, x_2, \ldots, x_N\}$, $Y = \{y_1, y_2, \ldots, y_N\}$, and $Z = \{z_1, z_2, \ldots, z_N\}$, then denoting * to represent the operations +, -, or ×, the RNS version of $Z = X * Y$ satisfies

$$Z = \{\langle x_1 * y_1 \rangle_{m_1}, \langle x_2 * y_2 \rangle_{m_2}, \ldots, \langle x_N * y_N \rangle_{m_N}\}.$$

Thus addition, subtraction, and multiplication can be concurrently performed on the N residues within N parallel channels, and it is this high

Manuscript received June 21, 2007; revised January 15, 2008. First published January 13, 2009; current version published February 19, 2009.. This work was supported by the Australian Research Council's Discovery Project Scheme (DP0559582).

The authors are with the Centre for High Performance Integrated Technologies and Systems (CHiPTec), the School of Electrical and Electronic Engineering, the University of Adelaide, Adelaide, SA 5005, Australia (e-mail: ykong@eleceng.adelaide.edu.au; phillips@eleceng.adelaide.edu.au).

Digital Object Identifier 10.1109/TVLSI.2008.2004550

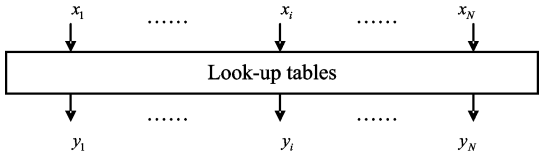

Fig. 1. Scaling using LUTs.

speed parallel operation that makes the RNS attractive. There is, however, no such parallel form of scaling or division.

B. Scale Factor K

From (1), we have

$$Y = \left\lfloor \frac{X}{K} \right\rfloor = \frac{X - \langle X \rangle_K}{K}$$

such that in an RNS

$$y_i = \langle Y \rangle_{m_i} = \left\langle \frac{X - \langle X \rangle_K}{K} \right\rangle_{m_i}. \tag{2}$$

Many scaling algorithms [2]–[4] take the scale factor K to be a product of a subset of the moduli, $K = \prod_{i=1}^{S} m_i$, because this permits the rapid evaluation of $\langle X \rangle_K$ from

$$\langle X \rangle_K = \left\langle \sum_{i=1}^{S} k_i \left\langle k_i^{-1} x_i \right\rangle_{m_i} \right\rangle_K.$$

If K is relatively prime to m_i, the multiplicative inverse of K modulo m_i exists. If we denote this as $\langle K^{-1} \rangle_{m_i}$, then (2) becomes

$$y_i = \left\langle \langle X - \langle X \rangle_K \rangle_{m_i} \times \langle K^{-1} \rangle_{m_i} \right\rangle_{m_i} \tag{3}$$

which is an equation to quickly generate scaled residues y_i. However, the evaluation of y_i for $1 \leq i \leq S$ is much more difficult. In these channels $\langle K^{-1} \rangle_{m_i}$ does not exist as $K = \prod_{i=1}^{S} m_i$ is not relatively prime to m_i. This step always consumes more time and hardware than the evaluation of y_i for $S+1 \leq i \leq N$ [5]. In [6], K is fixed to 2. The current paper extends this idea, allowing K to be any number coprime with the RNS moduli.

II. Space and Time Complexity

A. Lookup Table Implementation

It has been common for RNS scaling schemes to operate using lookup tables (LUTs) [2], [3], [5], [7]–[10]. Scaled results are precomputed and stored in a network of LUTs as shown in Fig. 1. In practice these LUTs may be implemented using devices such as ROMs, RAMs, PLAs, or combinatorial logic according to whichever is most appropriate for the target hardware platform. The various scaling schemes lead to different structures in the LUT network and, in general, trade reduced latency (achieved through exploiting parallelism within the network) against hardware cost.

In this paper, we will use LUT implementations to provide a fair basis for comparisons between scaling schemes. We assume that all LUTs in an implementation have the same size and then compare time complexity counted in lookup cycles (LUCs) and space complexity measured in the total number of LUTs.

Note that both the time and space complexity are heavily dependent on the width of each modulus and the size of the LUTs selected. We use r to denote the number of residue inputs addressing each LUT and assume that r remains the same for all of the LUTs within an implementation. For example, if we use the 5-bit moduli $\{19, 23, 29, 31\}$ and use ROMs with an address space of 32 K $=$ 4 K $\times$ 8 bits, then $r = \lfloor (\log_2 4K/5) \rfloor = 2$ because each memory can accommodate two residue inputs at most.

LUT implementations are appropriate for field-programmable gate array (FPGA) implementations which are typically rich in memory resources. For other platforms, alternatives to LUT implementations do exist. Instead of precomputing values and storing them in tables, they can be evaluated dynamically as the operation proceeds. This is the case for the RNS systems of [11]–[13].

B. Scaling Complexity

Early attempts at scaling were performed by converting from RNS to a positional (binary) representation where scaling can be trivially performed before the result is converted back to the RNS [1]. Such schemes incurred a time complexity of $O(N)$ LUCs. An improved form used in [2] and [4] decreased the number of LUCs to $O(\log_r N)$ by expressing the scaled integer Y as a sum of terms that can be evaluated in parallel

$$Y = \left\lfloor \frac{X}{K} \right\rfloor \approx \sum_{j=1}^{N} f(x_j). \tag{4}$$

The exact time complexity of residue arithmetic structures following this form is derived in the Appendix to be $\lceil \log_r N \rceil$. The exact space complexity is also shown to be $\lceil (N - 1/r - 1) \rceil$. Subsequent scaling schemes (e.g., [3], [8], [9], [14]) have not reduced time complexity below $O(\log_r N)$. The space complexity of scaling has remained at $O(N^2)$ LUTs [7] with little improvement over the development of RNS scaling algorithms [2], [3], [8], [9], [14].

The scaling scheme in this paper decreases the space complexity to $O(N)$ while maintaining $O(\log_r N)$ time complexity.

III. New Scaling Scheme

A. Base Extension Step

The new scaling scheme assumes the scaling factor K is a positive integer coprime to any of the moduli m_i. For comparison with other scaling schemes using a LUT implementation, we require that K is a constant with word length at most $(r-1)$ times the word length of the channel moduli. The first step in the new scaling scheme is to evaluate $\langle X \rangle_K$ from the RNS representation of X, i.e., $\{x_1, x_2, \ldots, x_N\}$. This is a typical base extension problem.

Efficient algorithms for base extension are presented in [1], [8], [15] and [16]. The scheme in [1] uses mixed radix conversion (MRC) which is relatively slow and costly; [15] employs an extra RNS channel with modulus greater than N; [16] performs an approximate extension; and [8] achieves exact scaling without an extra RNS channel. Any exact base extension is appropriate for our purposes. The algorithms [15] and [8] are the most time and space efficient, generating $\langle X \rangle_K$ in $O(\log_r N)$ LUCs using $O(N)$ LUTs. This efficiency does come at a cost: [15] requires extra hardware to maintain the extra channel; and [8] can be as slow as the MRC in some rare cases.

B. New Scaling Step

From (3), we can write

$$y_i = \left\langle \langle x_i - \langle X \rangle_K \rangle_{m_i} \times \langle K^{-1} \rangle_{m_i} \right\rangle_{m_i}. \tag{5}$$

Because K is coprime with all m_i, $\langle K^{-1} \rangle_{m_i}$ always exists and (5) can be used to evaluate y_i in every channel. For a constant K, $\langle K^{-1} \rangle_{m_i}$ can be precomputed and stored in a LUT.

Given $\langle X \rangle_K$ (5) can be implemented directly in each channel using subtraction and multiplication modulo m_i; however, to compare this scheme with those surveyed in the previous section, we will consider

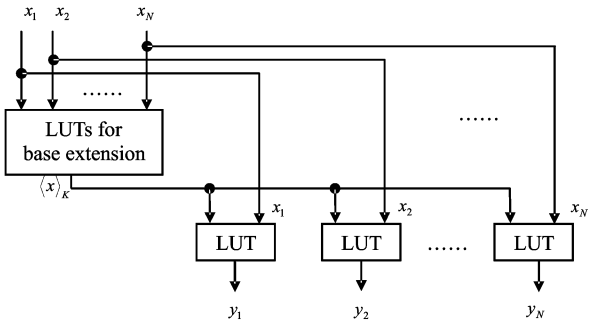

Fig. 2. Architecture to perform RNS scaling by K.

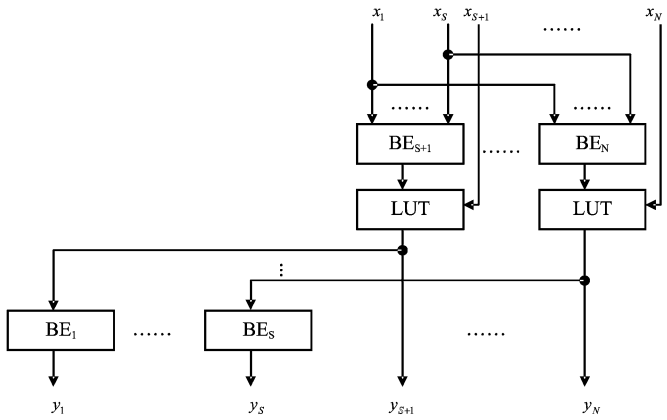

Fig. 3. Conventional scaling using BE blocks [8].

an implementation using LUTs. As there are only two inputs to (5), x_i and $\langle X \rangle_K$, (5) can be implemented using a single LUT for each output residue y_i provided the word length of K is at most $(r-1)$ times the word length of the moduli. In this case the scaling step only uses one LUC and N LUTs.

For example, if $r = 3$ and the channel width is 5 bits, the addressing capacity of each LUT is $2^{3\times 5} \times 5 = 32\text{ K} \times 5$ bits. In this case K can be as large as $(r-1) \times 5 = 10$ bits. K can be made larger if we use a larger LUT or concatenate LUTs to allow more addressing capacity. In the example above, if the largest available LUT is 512 K $= 64\text{ K} \times 8$, i.e., $2^{K+5} \leq 64$ K, then the scale factor K can be as large as $\log_2 64 + 10 - 5 = 11$ bits.

C. Whole Scaling Process

The scaling scheme is illustrated in Fig. 2. The base extension block in Fig. 2 costs $O(\log_r N)$ LUCs and $O(N)$ LUTs, and the scaling step consumes one LUC and N LUTs. Thus, the time complexity of this new scaling process is $O(\log_r N) + 1 = O(\log_r N)$ and the space complexity is $O(N) + N = O(N)$. The latter is an improvement for the scaling problem in RNS since all other methods known to the authors incur $O(N^2)$ hardware cost. The main reason is they need $O(N)$ LUTs to scale in one channel. When scaling over N channels, their space complexities become $O(N^2)$.

More specifically, suppose the base extension block in [8] is used, which has an exact time complexity of $T_r(N) + 2$ and an exact space complexity of $2S_r(N) + 4$, where $T_r(N) = \lceil \log_r N \rceil$ and $S_r(N) = \lceil (N - 1/r - 1) \rceil$. The exact time complexity of this new scaling process is $T_r(N) + 3$ and the exact space complexity is $2S_r(N) + N + 4$, where $T_r(N) = \lceil \log_r N \rceil$ and $S_r(N) = \lceil (N - 1/r - 1) \rceil$.

D. Example

As an example, consider the RNS moduli $m_1 = 23$, $m_2 = 25$, $m_3 = 27$, $m_4 = 29$, and $m_5 = 31$, and suppose the integer $X = 578321 = \{9, 21, 8, 3, 16\}$ is to be scaled by $K = 1039$. $\langle K^{-1} \rangle_{m_i}$ has been precomputed as $\{6, 9, 25, 23, 2\}$ for $1 \leq i \leq 5$. We base extend X to K to compute $\langle X \rangle_K = 637$. Then, according to (5), the scaled residues are computed as $y_1 = \langle \langle 9 - 637 \rangle_{23} \times 6 \rangle_{23} = 4$, $y_2 = \langle \langle 21 - 637 \rangle_{25} \times 9 \rangle_{25} = 6$, $y_3 = \langle \langle 8 - 637 \rangle_{27} \times 25 \rangle_{27} = 16$, $y_4 = \langle \langle 3 - 637 \rangle_{29} \times 23 \rangle_{29} = 5$, and $y_5 = \langle \langle 16 - 637 \rangle_{31} \times 2 \rangle_{31} = 29$. Thus, $Y = \{4, 6, 16, 5, 29\} = 556 = \lfloor (578321/1039) \rfloor$. Note that in this example all operations can be performed using 64 K $\times$ 8 bit LUTs.

E. Evaluation

Though base extension has long been used in RNS scaling, the way that it has been applied has remained the same since it was proposed in [5] in 1978. Since this time, scaling algorithms using base extension have always chosen K to be a product of a subset of the moduli: $K = \prod_{i=1}^{S} m_i$. As the scaled integer

$$\begin{aligned} Y &= \frac{X - \langle X \rangle_K}{K} \\ &= \left\langle \frac{X - \langle X \rangle_K}{K} \right\rangle_{(M/K)} \\ &= \left\langle (\langle X \rangle_{(M/K)} - \langle \langle X \rangle_K \rangle_{(M/K)}) \langle K^{-1} \rangle_{(M/K)} \right\rangle_{(M/K)} \end{aligned}$$

can be represented in the range $[0, (M/K) - 1]$, a base extension of $\langle X \rangle_K = \{x_1, x_2, \ldots, x_S\}$ to the moduli $m_{S+1}, m_{S+2}, \ldots, m_N$ is first performed. Then, the resulting residues $\{x_{S+1}, x_{S+2}, \ldots, x_N\}$ are involved in a simple table lookup step to compute the representation of Y in the range $[0, (M/K) - 1]$, i.e., $\{y_{S+1}, y_{S+2}, \ldots, y_N\}$. Finally, these residues are base extended back to moduli $m_1, m_2, \ldots, m_S$ to obtain the representation of Y over the whole RNS range, i.e., $\{y_1, y_2, \ldots, y_N\}$ [5], [8], [9]. The schemes [8] and [9] are similar but the latter replaces the base extension blocks with large LUTs with up to $S + 1$ inputs. For an RNS with more than about 3 5-bit channels, such large LUTs are not available and hence the scheme is only viable in some specific cases as stated in [9].

As shown in Fig. 3, this process involves exactly N base extension (BE) blocks no matter what kind of base extension algorithms are used. Since the space complexity of each base extension is already no less than $O(N)$, the whole scaling space complexity amounts to $O(N) \times N + (N - S) = O(N^2)$. This can be compared with the new scaling architecture in Fig. 2 which has only 1 base extension block and space complexity $O(N)$. In Fig. 3, the time complexity is $O(\log_r S + \log_r (N - S)) = O(\log_r N)$. This is the same as the new scheme.

Table I provides results comparing the new scaling process with those described in [2], [8], [9], and [14]. Assume the new scaling uses the base extension technique given in [8] and EPROMs of 256 K $=$ 32 K $\times$ 8 are used as LUTs. Therefore, the number of residue inputs addressing each memory is $r = \lfloor (\log_2 32 + 10/\text{Channel Width}) \rfloor = \lfloor (15/\text{Channel Width}) \rfloor$. The scale factor K should be no larger than $\log_2 32 + 10 - \text{Channel Width} = 15 - \text{Channel Width}$ bits long. Because the scaling schemes in [8] and [9] only support even values of N, N is always chosen to be even here, although there is no such restriction in the new scaling algorithm.

As can be seen from the Table I, the larger the dynamic range, the more obvious the advantage of the new scaling algorithm in terms of hardware.

TABLE I
COMPARISON BETWEEN NEW SCALING AND CONVENTIONAL SCALING SCHEMES USING 32 K $\times$ 8 LUTS

RNS Moduli			{17,19,23, 27,29,31}	{37,41,43,47, 53,59,61,63}	{67,71,73,79,83,89,97, 101,103,107,109,113}
N			6	8	12
Channel Width			5 bits	6 bits	7 bits
r (= $\lfloor \frac{15}{\text{Channel Width}} \rfloor$)			3	2	2
Largest K (= 15 – Channel Width)			10 bits	9 bits	8 bits
Exact Time Complexity (LUCs)	New Scaling	$T_r(N)+3$	5	6	7
	Scaling in [2]	$T_r(N)+3$	5	6	7
	Scaling in [8]	$2T_r(\frac{N}{2})+5$	7	9	11
	Scaling in [9]	$T_r(\frac{N}{2}+1)+T_r(\frac{N}{2})+1$	4	6	7
	Scaling in [14]	$T_r(N)+2$	4	5	6
Exact Space Complexity (LUTs)	New Scaling	$2S_r(N)+N+4$	16	26	38
	Scaling in [2]	$(N+2)S_r(N)+N$	30	78	166
	Scaling in [8]	$2NS_r(\frac{N}{2})+\frac{9}{2}N$	39	84	174
	Scaling in [9]	$\frac{N}{2}(S_r(\frac{N}{2}+1)+S_r(\frac{N}{2}))$	9	28	66
	Scaling in [14]	$NS_r(\frac{N}{2}+1)+2S_r(N)+S_r(\frac{N}{2})$	19	49	99

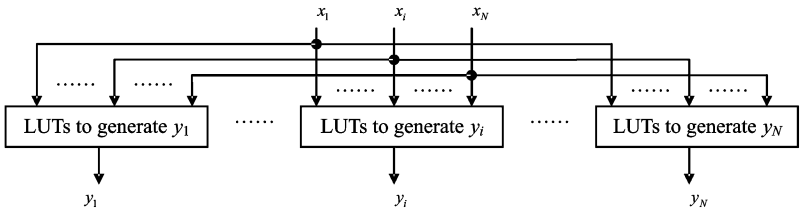

Fig. 4. Parallel architecture to perform (4).

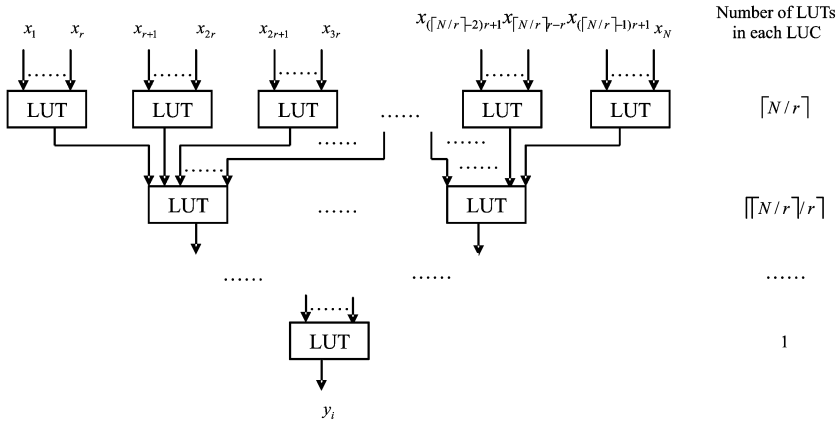

Fig. 5. One channel of parallel residue arithmetic process using memories with addressing capacity $= r$.

IV. CONCLUSION

A low latency scaling scheme is illustrated in Fig. 2 that reduces hardware cost to $O(N)$ down from $O(N^2)$ required for previous solutions. The scheme imposes no restrictions on the scale factor K other than it must not be too large and be coprime with the RNS moduli. Base extension algorithms are applied in a simple way to achieve scaling with only 1 base extension step. Most of the time and hardware resources consumed in the scaling are required by the base extension step. This means that there is a tight connection between base extension and scaling in that any improvement in base extension algorithms will immediately lead to more efficient scaling.

APPENDIX

In this appendix, the time and space complexities of residue arithmetic structures following (4) are derived. This equation is typical of residue arithmetic processes that achieve $O(\log_r N)$ time complexity. A ROM network to perform (4) was shown in Fig. 1. A more detailed diagram appears in Fig. 4. The two base extension structures ([8] and [15]) used in this paper can be implemented with this structure.

Suppose each available LUT can accept only r inputs at most while generating only one output as discussed in Section II above. Then, each channel of the parallel scaling structure in Fig. 4 can be drawn as a tree as in Fig. 5, where it is assumed that there are N input residues and n LUCs are consumed to accomplish the scaling in channel i.

In the first cycle, the number of LUTs is $\lceil N/r \rceil$. Thus, there are $\lceil N/r \rceil$ input residues to the LUTs in the second cycle, where the number of LUTs will be $\lceil \lceil N/r \rceil / r \rceil$. This proceeds recursively until only one LUT is needed, i.e., $\lceil \ldots \lceil \lceil N/r \rceil / r \rceil \ldots / r \rceil = 1$ as illustrated in Fig. 5. Using the result from Number Theory, $\lceil N/r^2 \rceil = \lceil \lceil N/r \rceil / r \rceil$, gives the number of LUTs as $\lceil N/r \rceil$ in the first cycle, $\lceil N/r^2 \rceil$ in the second and so on, until the last

cycle, where $\lceil \ldots \lceil \lceil N/r \rceil / r \rceil \ldots / r \rceil = \lceil N/r^n \rceil = 1$. Then from $N/r^n \leq \lceil N/r^n \rceil = 1$, we have $N/r^n \leq 1 \Rightarrow n \geq \log_r N$.

If $0 < N/r^{n-1} \leq 1$, then $\lceil N/r^{n-1} \rceil = 1$. This means only $n-1$ cycles are needed and this contradicts our original assumption that n cycles are required. Therefore, $N/r^{n-1} > 1 \Rightarrow n < \log_r N + 1$ and $\log_r N \leq n < \log_r N + 1$, so that

$$n = \lceil \log_r N \rceil. \tag{6}$$

This represents the exact time complexity of the ith channel of the residue arithmetic process shown in Fig. 5. Because all the N channels run in parallel, $\lceil \log_r N \rceil$ is also the exact time complexity of the scaling scheme constructed on r-input LUTs.

It can also be proven that the exact space complexity of each channel is $\lceil (N - 1/r - 1) \rceil$ such that the exact space complexity of the whole arithmetic process is $N \lceil (N - 1/r - 1) \rceil$, which is at the level of $O(N^2)$.

REFERENCES

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

[2] A. Shenoy and R. Kumaseran, "A fast and accurate rns scaling technique for high speed signal processing," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 37, no. 6, pp. 929–937, Jun. 1989.

[3] M. Griffin, M. Sousa, and F. Taylor, "Efficient scaling in the residue number system," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 1989, pp. 1075–1078.

[4] O. Aichholzer and H. Hassler, "A fast method for modulus reduction in residue number system," in *Economical Parallel Process.*, Vienna, Austria, 1993, pp. 41–54.

[5] G. A. Jullien, "Residue number scaling and other operations using rom arrays," *IEEE Trans. Comput.*, vol. 27, no. 2, pp. 325–336, Apr. 1978.

[6] U. Meyer-Bäse and T. Stouraitis, "New power-of-2 RNS scaling scheme for cell-based ic design," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 11, 4, no. , pp. 280–283, Apr. 2003.

[7] Y. Kong and B. Phillips, "Residue number system scaling schemes," in *Proc. SPIE, Smart Structures, Devices, and Systems II*, S. F. Al-Sarawi, Ed., Feb. 2005, vol. 5649, pp. 525–536.

[8] F. Barsi and M. C. Pinotti, "Fast base extension and precise scaling in RNS for look-up table implementations," *IEEE Trans. Signal Process.*, vol. 43, no. 10, pp. 2427–2430, Oct. 1995.

[9] A. Garcia and A. Lloris, "A look-up scheme for scaling in the RNS," *IEEE Trans. Comput.*, vol. 48, no. 7, pp. 748–751, Jul. 1999.

[10] J. Ramirez, U. Meyer-Bäse, A. García, and A. Lloris, "Design and implementation of rns-based adaptive filters," in *Proc. 13th Int. Conf. (FPL)*, 2003, vol. 2778, pp. 1135–1138.

[11] E. D. D. Claudio, F. Piazza, and G. Orlandi, "Fast combinatorial RNS processors for DSP applications," *IEEE Trans. Comput.*, vol. 44, no. 5, pp. 624–633, May 1995.

[12] J. Vaccaro, B. Johnson, and C. Nowacki, "A systolic discrete Fourier transform using residue number systems over the ring of Gaussian integers," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, Apr. 1986, vol. 11, pp. 1157–1160.

[13] S. J. Meehan, S. D. O'Neil, and J. J. Vaccaro, "A universal input and output RNS converter," *IEEE Trans. Circuits Syst.*, vol. 37, no. 6, pp. 799–803, Jun. 1990.

[14] N. Burgess, "Scaling an RNS number using the core function," in *Proc. 16th IEEE Symp. Comput. Arithmetic*, 2003, pp. 262–269.

[15] A. Shenoy and R. Kumaseran, "Fast base extension using a redundant modulus in rns," *IEEE Trans. Comput.*, vol. 38, no. 2, pp. 292–297, Feb. 1989.

[16] K. C. Posch and R. Posch, "Modulo reduction in residue number systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 6, no. 5, pp. 449–454, May 1995.

Fully Monolithic Cellular Buck Converter Design for 3-D Power Delivery

Jian Sun, David Giuliano, Siddharth Devarajan, Jian-Qiang Lu, T. Paul Chow, and Ronald J. Gutmann

***Abstract*—A fully monolithic interleaved buck dc-dc point-of-load (PoL) converter has been designed and fabricated in a 0.18-mm SiGe BiCMOS process. Target application of the design is 3-D power delivery for future microprocessors, in which the PoL converter will be vertically integrated with the processor using wafer-level 3-D interconnect technologies. Advantages of 3-D power delivery over conventional discrete voltage regulator modules (VRMs) are discussed. The prototype design, using two interleaved buck converter cells each operating at 200 MHz switching frequency and delivering 500 mA output current, is discussed with a focus on the converter power stage and control loop to highlight the tradeoffs unique to such high-frequency, monolithic designs. Measured steady-state and dynamic responses of the fabricated prototype are presented to demonstrate the ability of such monolithic converters to meet the power delivery requirements of future processors.**

***Index Terms*—3-D integration, dc-to-dc converters, monolithic power conversion, power delivery, power management, voltage regulator.**

I. INTRODUCTION

Future microprocessors and high-performance integrated circuits (ICs) will require multiple, dynamically scalable, sub-1-V supply voltages with total current exceeding 100 A/chip [1]. Conventional power delivery methods employing a voltage regulator module (VRM) mounted on the motherboard have several limitations in meeting future IC technology needs. One critical problem of this 2-D power delivery architecture is the long interconnect between the VRM and the processor, which creates an impedance bottleneck for dynamic power delivery and forces the use of decoupling capacitors at various locations along the power delivery path. Another problem of 2-D power delivery is the large number of power and ground pins required by the processor, which consumes expensive board area around the processor and/or increases packaging complexity. Meeting the power delivery requirements of future microprocessors and high-performance ICs requires a paradigm shift in power delivery system design and integration.

3-D power delivery [2]–[5], in which the power supply is vertically integrated with the processor in a 3-D stack, offers a possible solution to the problems of 2-D power delivery by dramatically reducing the interconnect parasitics. In addition, this ultimate point-of-load (PoL) converter configuration reduces the number of power pins and facilitates the delivery of multiple supply voltages. Of the different 3-D architectures discussed in the literature, the wafer-level 3-D approach proposed

Manuscript received September 01, 2007; revised February 19, 2008. First published February 03, 2009; current version published February 19, 2009. This work was supported in part by the Interconnect Focus Center sponsored by MARCO, DARPA, and NYSTAR, by the NSF under ERC Award EEC-9731677 (for the Center for Power Electronics Systems), and by the IBM-sponsored RPI Broadband Center.

The authors are with Rensselaer Polytechnic Institute, Troy, NY 12180 USA (e-mail: jsun@rpi.edu).

S. Devarajan was with Rensselaer Polytechnic Institute, Troy, NY 12180 USA. He is now with Linear Technology, Inc.

D. Giuliano was with Rensselaer Polytechnic Institute, Troy, NY 12180 USA. He is now with Massachusetts Institute of Technology (MIT), Boston, MA 02139 USA.

Digital Object Identifier 10.1109/TVLSI.2008.2005312