ELSEVIER

Signal Processing 51 (1996) 65–67

SIGNAL PROCESSING

A note on digital filter implementation using hybrid RNS-binary arithmetic

Behrooz Parhami*

Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106-9560, USA

Received 23 June 1995; revised 6 February 1996

Abstract

Binary logic implementation of residue arithmetic using longer intermediate pseudo-residues has been shown to offer advantages over standard binary and lookup-table RNS realizations. In this note, we show that a small modification to one such proposed scheme leads to even greater advantages.

Zusammenfassung

Man hat gezeigt, daß die Binärlogik-Realisierung von Restklassenarithmetik mit der Verwendung längerer Pseudoresidual-Zwischenwerte gegenüber den üblichen binären, tabellenorientierten RNS-Realisierungen Vorteile zu bieten hat. In diesem Kurzbericht zeigen wir, daß eine kleine Modifikation eines solchen Vorschlags zu noch größeren Vorteilen führt.

Résumé

L'implantation en logique binaire de l'arithmétique des résidus utilisant les pseudo-résidus intermédiaires plus longs a montré ses avantages par rapport aux réalisations en binaire standard et en RNS de tables de correspondances. Dans cette note, nous montrons qu'une légère modification de l'un de ces schémas conduit à des avantages encore plus importants.

Keywords: Digital filters; Inner product; RNS arithmetic; VLSI architectures

1. Introduction

In [1], Ibrahim suggests a scheme for the efficient implementation of digital filters. His scheme is based on residue number system (RNS) arithmetic whereby an integer is represented by its residues with respect to a set of moduli. Multiplication in standard RNS arithmetic is based either on table-lookup (requiring large, expensive tables in some cases) or on conventional binary multiplication followed by a time-consuming modular reduction. To simplify the required modular reduction after each multiply-accumulate step, Ibrahim suggests that the accumulated sum be kept in double-length

*Tel.: + 1 805 893 3211; fax + 1 805 893 3262: e-mail: parhami@ece.ucsb.edu.

"pseudo-residue" form, delaying the complex reduction to proper residues to the very end, where it is done only once.

Even though Ibrahim's presentation is in terms of FIR filters, the same technique is applicable to many other signal processing problems involving multiply-accumulate or inner product computations. Thus, the significance of the method extends beyond the limited application area discussed. In this note, we show that a small modification to Ibrahim's proposed scheme leads to even greater advantages for all such applications.

2. The original implementation

Briefly, Ibrahim's proposed scheme is as follows. To compute an inner product with respect to one of the moduli, say m, two n-bit modulo-m residues are multiplied to obtain a $2n$-bit product. This $2n$-bit product is added to a $2n$-bit running total, to give a $(2n + 1)$-bit result; viewed as a $2n$-bit value and an overflow bit. Note that standard RNS arithmetic dictates that any intermediate result exceeding $m - 1$ be reduced to a proper n-bit residue before further processing. Since $x \bmod m = (x \pm km) \bmod m$ for any integer k, Ibrahim's use of $2n$-bit pseudo-residues, that may be off from the proper modulo-m residues by a multiple of m, does not cause any problem in the intermediate computations.

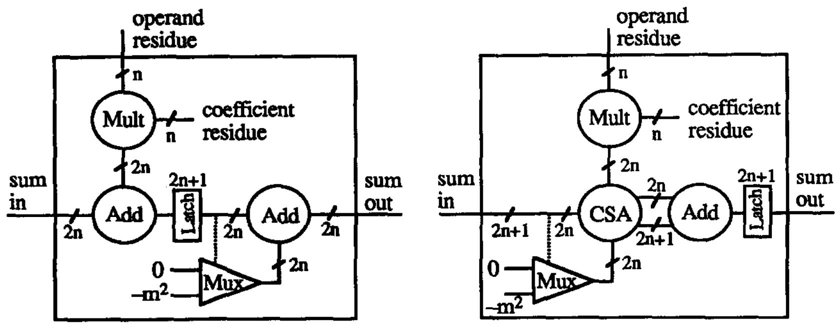
In Ibrahim's first implementation, the overflow bit is used to select 0 or $-m^2$ to be added to the $2n$-bit value. Since overflow, if any, results from adding a magnitude not exceeding $(m - 1)^2$ to a $2n$-bit value, subtracting m^2 (which is a multiple of m and therefore does not change the modulo-m residue of the result) is adequate to bring the result back to the $2n$-bit range. Besides the final modulo-m reduction hardware for the $2n$-bit result, this implementation requires an $n \times n$ multiplier and two $2n$-bit adders for the inner product step, as shown in Fig. 1(a).

Ibrahim also provides an improved implementation which replaces the first of the two adders with a carry-save adder (CSA). This is depicted in Fig. 1(b). Rather than subtracting m^2 from the $(2n + 1)$-bit sum immediately following each addition, this is done within the following cell, allowing the use of a CSA to combine three values: the $2n$-bit sum, the new product term, and $-s_{2n}m^2$, the last term being selected by the overflow bit s_{2n}.

Ibrahim concludes that the above cells, when used to implement a FIR filter, lead to improved cost and speed with respect to both previously proposed RNS schemes (using logic-circuit or tabular realizations) and standard binary designs.

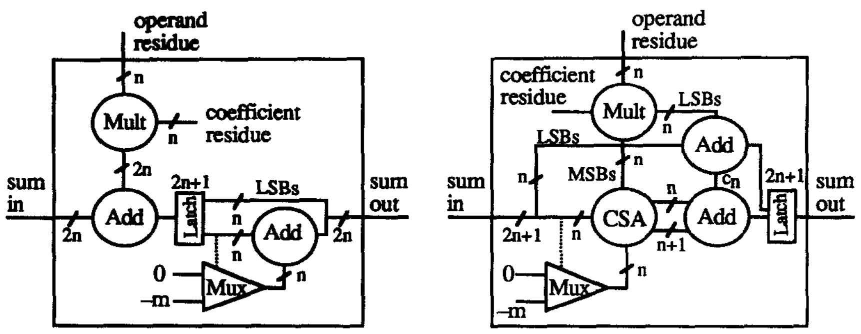
3. The proposed enhancement

When the overflow bit s_{2n} is 1, the $(2n + 1)$-bit sum is in the range $2^{2n} \leqslant s < 2^{2n+1}$. Any multiple of m can be subtracted from this sum which will be eventually reduced to n bits by a final modulo-m operation. We suggest subtracting $2^n m$, which is no



(a) Hybrid RNS-Binary Basic Cell (b) Improved Basic Cell Architecture

Fig. 1. The original implementations of the multiply–add cell for each RNS modulus.



(a) Hybrid RNS-Binary Basic Cell (b) Improved Basic Cell Architecture

Fig. 2. The proposed enhanced implementations of the multiply–add cell.

greater than $2^n(2^n - 1)$, instead of m^2. This will not change the validity of the design or the associated proofs and analyses. Neither will this have an adverse affect of the final reduction step to convert the result to an n-bit residue, since subtracting a larger value tends to make the intermediate results smaller and thus easier (certainly no harder) to reduce. Since $2^n m$ has 0's for the n least significant bits, the cell designs are simplified, as shown in Fig. 2.

Comparing Figs. 1(a) and 2(a), we see that in our proposed design, the widths of the multiplexer and of the second (right-hand) adder have been reduced from $2n$ to n. This affects not only the cost of the cell but also its speed. As for Figs. 1(b) and 2(b), a similar reduction is seen in the widths of the multiplexer and carry-save adder. The two n-bit and $(n + 1)$-bit adders in Fig. 2(b) are equivalent to the $(2n + 1)$-bit adder in Fig. 1(b). However, since in our proposed design of Fig. 2(b), the high-order $n + 1$ bits of this $(2n + 1)$-bit addition are applied with a few gate levels of delay, an optimized design can lead to lower complexity and delay for this $(2n + 1)$-bit addition as well.

Reference

[1] M.K. Ibrahim, "Novel digital filter implementations using hybrid RNS-binary arithmetic", *Signal Processing*, Vol. 40, Nos. 2-3, November 1994, pp. 287–294.