Implementation of Digital Filters in Carry-Save Residue Number System

Andrea Del Re, Alberto Nannarelli and Marco Re
Department of Electrical Engineering
University of Rome "Tor Vergata" - Italy

http://dspvlsi.uniroma2.it/

Abstract

In this work, we present the implementation of a Finite Impulse Response (FIR) filter in the Residue Number System (RNS), in which we use a carry-save scheme in the binary representation of the residues to speed-up modular additions. We compare the carry-save RNS implementation with the implementations of the same filter in the traditional binary system and in plain RNS. Results show that the carry-save RNS filter is much faster and its energy dissipation per cycle comparable. Furthermore, we show that a multiple supply voltage approach for the plain RNS filter can lead to an additional reduction in power dissipation without performance degradation.

1. Introduction

The new generation of telecommunication equipment require the use of high order, high throughput Finite Impulse Response (FIR) filters. On the other hand, low power consumption is desirable for portable devices and ultra dense circuits. In this scenario, computationally intensive signal processing blocks can be effectively implemented by using the Residue Number System (RNS) arithmetic.

The use of the Residue Number System allows the decomposition of a given dynamic range in slices of smaller range on which the computation can be implemented in parallel [1, 2, 3]. The drawback presented by the RNS is the overhead due to both input and output conversions: binary to RNS, and RNS to binary. However, by using efficient conversion techniques [4, 5], the overhead can be significantly reduced.

In our work, we compare the performance, area and power of a FIR filter realized in the traditional binary system, with a RNS based one. Furthermore, we make the RNS filter faster than the traditional one by introducing a redundant carry-save representation, and on the other hand, we reduce the power dissipation of the plain RNS filter without affecting its delay by equalizing the parallel paths of the filter with a multiple supply voltage approach.

Results show that the RNS filter can sustain the same throughput as the traditional one, but dissipates less power. The carry-save RNS implementation of the filter can be clocked almost at double speed, with respect to the traditional one, without a significant increase in area and energy dissipated in a cycle.

2. RNS FIR Filter

The starting point of our design is a programmable N taps FIR filter

$$y(n) = \sum_{k=0}^{N} a_k x(n-k)$$

realized in transposed form (Figure 1).

In RNS, the FIR filter is decomposed into P filters working in parallel, as shown in Figure 2, and both binary to RNS and RNS to binary converters are required.

In each tap, a modular multiplier is needed to compute the term $\langle a_k x(n-k)\rangle_{m_i}$. Because of the complexity of modular multiplication, we used the isomorphism technique [6] to implement the product of residues By using isomorphisms, the product of the two residues is transformed into the sum of their indices which are obtained by an isomorphic transformation. According to [6], if m is prime there exists a primitive radix q such that its powers modulo m cover the set $[1, m-1]$:

$$n_i = \langle q^{w_i}\rangle_m \qquad \text{with} \quad \begin{array}{l} n_i \in [1, m-1] \\ w_i \in [0, m-2]. \end{array}$$

Both transformations $n \rightarrow w$ and $w \rightarrow n$ can be implemented with $m-1$ entries tables. Therefore, the product

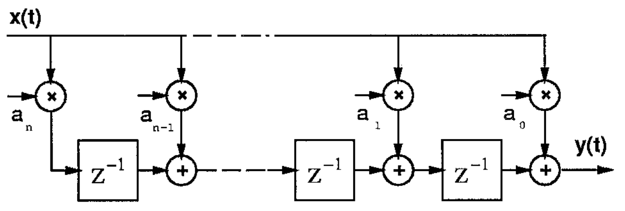

Figure 1. FIR filter in transposed form.

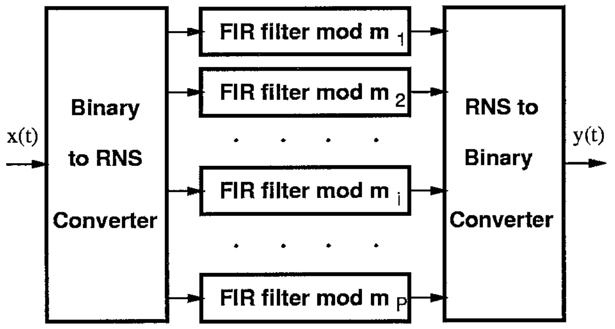

Figure 2. RNS FIR filter.

of a_1 and a_2 modulo m can be obtained as:

$$\langle a_1 \cdot a_2 \rangle_m = \langle q^w \rangle_m$$

where

$$w = \langle w_1 + w_2 \rangle_{m-1} \qquad \text{with} \quad \begin{array}{l} a_1 = \langle q^{w_1} \rangle_m \\ a_2 = \langle q^{w_2} \rangle_m \end{array}$$

Because of the transposed form of the FIR filter, the input x is the multiplicand of all the multiplications (see Figure 1). For this reason only one direct isomorphic transformation, incorporated in the binary to RNS conversion, is necessary for all the taps. On the other hand, because the coefficients of the filter (multiplicators) are constant terms loaded once at start-up, it is convenient to load directly the isomorphic representation modulo $m_i - 1$. As a result, in each tap, we reduce the modular multiplication to a modular addition, implemented as explained below, followed by an access to table (inverse isomorphism). The table is implemented as synthesized logic.

The modular addition $\langle b_1 + b_2 \rangle_m$, consists of two binary additions. If the result of $b_1 + b_2$ exceeds the modulo (it is larger than $m - 1$), we have to subtract the modulo m. In order to speed-up the operation we can execute in parallel the two operations:

$$(b_1 + b_2) \quad \text{and} \quad (b_1 + b_2 - m).$$

If the sign of the three-term addition is negative it means than the sum $(b_1 + b_2) < m$ and the modular sum is $b_1 + b_2$, otherwise the modular addition is the result of the three-term addition.

The resulting implementation of the tap for one modulo is depicted in Figure 3. Due to the transposed form of the filter (Figure 1), x must be buffered to avoid long delays. The critical path of the RNS filter, given by the slowest modulo, is:

$$t_{RNS} = t_{buffer} + t_{modMULT} + t_{modADD} + t_{REG} \ . \quad (1)$$

Because we chose as minimum clock period the delay (critical path) in the slowest tap, the converters had to be pipelined to maintain the filter throughput. This resulted in

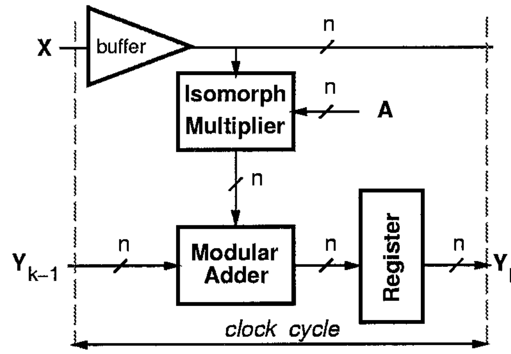

Figure 3. Structure of one modulo RNS tap.

a latency of N_{cv} required for the conversions. The expression for the area is

$$Area_{RNS} = Area_{convIN} + Area_{TAP} \cdot N + Area_{convOUT}$$

where $Area_{TAP}$ is the sum of the area of the P parallel taps: $Area_{TAP} = \sum_{i=0}^{P} Area_{TAP_i}$. As for the power dissipation, we get a similar expression

$$P_{RNS} = P_{convIN} + P_{TAP} \cdot N + P_{convIN}$$

with the limitation that the term P_{TAP} strongly depends on the switching activity and consequently on the value of the coefficients.

3. Carry-Save RNS Filter

In order to speed-up the operations by making the clock period shorter, we can resort to a carry-save representation for the binary representation of residues (ys_k, yc_k) and avoid to compute the modular addition in every tap. The operands to be added in a tap are three: the product $p_k = \langle a_k x(n-k) \rangle_{m_i}$ and the carry-save representation of y_{k-1}

$$\begin{array}{l} ys_k = sum(ys_{k-1}, yc_{k-1}, p_k), \\ yc_k = carry(ys_{k-1}, yc_{k-1}, p_k). \end{array}$$

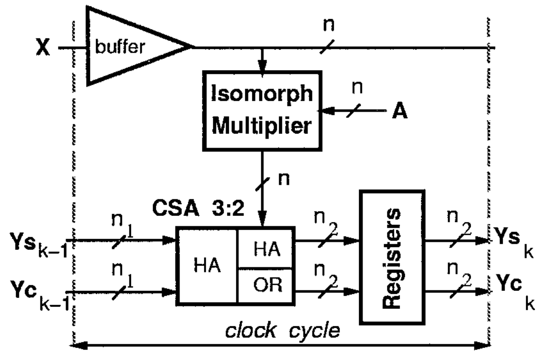

Figure 4. Tap structure for RNS carry-save.

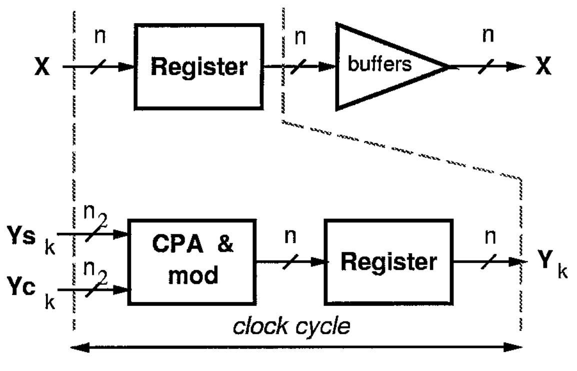

Figure 5. Relay station.

The structure of the carry-save RNS tap is shown in Figure 4. With the new representation, we reduce the term t_{modADD} of Expr. (1) to the delay of a half-adder and the critical path is now:

$$t_{CS-RNS} = t_{buffer} + t_{modMULT} + t_{XOR} + t_{REG} \quad (2)$$

The critical path is essentially determined by the multiplier latency, being t_{buffer} and t_{REG} unavoidable, and being t_{XOR} the minimum delay attainable for a half-adder.

However, the CS-representation implies the doubling of the registers, and, as the number of taps increases, a logarithmic increase in the bit-width of CSAs and registers. For this reason, it might be convenient to insert some *relay stations* (RS), which assimilate the carry-save representation of y_k and extract its modulo m_i, to prevent the bit-width from growing too much. Relay stations are depicted in Figure 5 and their delay is

$$t_{RS} = t_{cpa\&mod} + t_{REG} \quad (3)$$

They introduce an extra cycle of latency, but they can also be used to better dimension the buffering of x (i.e. reduce t_{buffer}).

The spacing of relay stations (i.e. the number n of taps between two relay stations) can be determined by combining Expr. (2), in which the term t_{buffer} increases with n, and Expr. (3), where $t_{cpa\&mod}$ depends on n. However, by rewriting $t_{buffer} = t_{int} + t_{load} \cdot n$, the sum

$$t_{int} + t_{modMULT} + t_{XOR} + t_{REG}$$

becomes dominant over Expr. (3) and, as a consequence, we can choose n, which satisfies our timing constraints, by Expr. (2) only:

$$t_{int} + t_{load} \cdot n + t_{modMULT} + t_{XOR} + t_{REG} < T_{clock}$$

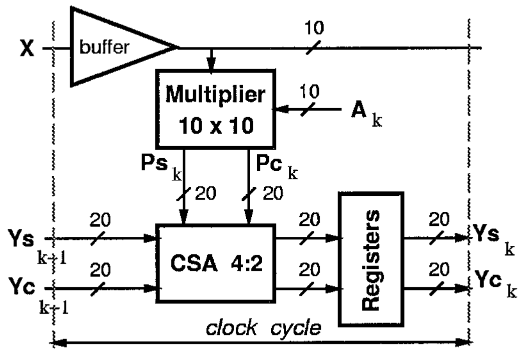

Figure 6. Tap structure for the traditional filter implementation.

4. Traditional FIR Filter

In order to evaluate the performance of these RNS filters, we have implemented a N-tap error-free FIR filter (20 bits dynamic range) in the traditional two's complement system (T2S) [7], in RNS [7] and in RNS using the carry-save scheme (CS-RNS).

For the traditional T2S filter we opted for a carry-save representation in the taps to keep the cycle time as short as possible (Figure 6).

The product $p_k = a_k x(n-k)$ is realized with a Booth multiplier [8] and the resulting partial products are accumulated in a Wallace tree structure which produces a carry-save representation of the product. Because the sum at the $(k-1)$-th tap $Y_{k-1} = \sum_{i=0}^{k-1} a_i x(n-i)$ is stored in carry-save representation, an array of 4:2 compressors [9] is required to reduce the CS representation of p_k and the CS representation of Y_{k-1} to the CS representation of $Y_k = Y_{k-1} + p_k$ in the k-th tap.

The carry-save representation is finally converted into the two's complement representation by a carry-propagate adder (realized with a carry-look-ahead scheme) in the last stage of the filter.

Filter	Cycle	Area		Power @100 MHz	
	$[ns]$	(NAND2 equiv.)	N^*	$[mW]$	N^*
T2S	5.0	200+1400N	-	6.0 + 15.5N	-
RNS	5.0	3250+920N	8	25.0 + 7.8N	4
CS-RNS	3.0	3470+1100N	12	27.4 + 10.5N	6
MV-RNS	5.0			(est.) 25.0 + 6.6N	3

Table 1. Summary of results for implemented filters.

The critical path is:

$$t_{T2S} = t_{buffer} + t_{MULT} + t_{CSA-4:2} + t_{REG}\ .$$

An extra clock cycle is needed to compute y from its CS-representation ($y = y_s + y_c$). The expression for the area as a function of the number of taps is:

$$Area_{T2S} = Area_{TAP} \cdot N + Area_{CPA}$$

where $Area_{CPA}$ is the area of the final carry-propagate adder. For the power dissipation the expression is similar to that of the area:

$$P_{T2S} = P_{TAP} \cdot N + P_{CPA}.$$

5. Results and Comparisons

For the RNS and CS-RNS filters, in order to have a dynamic range of 20 bits, as in the case of the traditional implementation, we chose the following set of moduli:

$$m_i = \{3, 5, 7, 11, 17, 64\}$$

such that $\log_2(3 \cdot 5 \cdot 7 \cdot 11 \cdot 17 \cdot 64) \geq 20$.

The multiplication was implemented with isomorphism in all the moduli but 3 (multiplication can be easily computed in tabular way) and 64 (modular product corresponds to the 6 least-significant bits of the conventional product).

Pipeline registers are added in the converters to maintain the clock cycle, determined by the critical path in the taps, to its minimum.

As for the implementation of carry-save RNS, in our design, the carry-save representation is not used for modulo 3 because its tap delay is sufficiently small. However, we have to place relay stations (just two 2-bit registers and, eventually, buffers) to synchronize $\langle X \rangle_3$ with the other $\langle X \rangle_{m_i}$. For modulo 64, the use of CS-representation does not imply an increase of bit-width in the CSAs and registers.

For all the moduli, in our carry-save RNS (CS-RNS) filter we placed relay stations spaced by 8 taps, because this configuration gives us the best delay-area tradeoff.

The filters were implemented in a $0.35\mu m$ library of Standard Cells. Delay, area and power dissipation have been determined with Synopsys tools.

The results are shown in Table 1 along with an evaluation of a possible implementation with multiple supply voltages (MV-RNS), described in the next section.

In the table, area, reported as number of NAND2 equivalent gates, and power, computed at 100 MHz, are expressed as a function of the number of taps (N). Column N^* indicates the N for which the RNS and the CS-RNS filter occupy less area and dissipate less power than the corresponding T2S filter. Although interconnections are not taken into account, local and global routing for the RNS is expected to be no worse than for the traditional filter.

6. Multi-Voltage RNS Filter

In addition to the above presented implementations, we add a possible approach to reduce the power dissipation in a RNS filter without penalizing its speed.

The power dissipated in a cell depends on the square of the supply voltage (V_{DD}) so that a significant amount of energy can be saved by reducing this voltage [10]. However, by lowering the voltage the delay increases, so that to maintain the performance this technique is applied only to cells not in the critical path.

Table 2 reports details on the implementation of the plain RNS filter for the different paths corresponding to the moduli m_i. Delay is normalized to the critical path (i.e. clock cycle), while area and power dissipation are normalized to their per tap totals.

Because the single tap delay of moduli 3, 5, 7 and 64 is less than the critical path, we can use the available time slack and reduce the supply voltage for these moduli without affecting the overall performance.

In Table 3 we report the possible supply voltage which can be used in the modulo m_i filters without increasing the critical path. The library of standard cells we used normally operates at $V_{DD} = 3.3\ V$. Table 3 also reports the power savings obtained with the listed supply voltage. These values have been computed assuming that the switching activity does not change when scaling the voltage. This assump-

Modulus	Delay	Area	Power
3	0.35	0.05	0.04
5	0.70	0.10	0.10
7	0.75	0.13	0.12
11	1.00	0.24	0.25
17	1.00	0.23	0.26
64	0.65	0.25	0.24
total	-	1.00	1.00

Table 2. Delay, area and power dissipation per tap in RNS FIR.

	single V_{DD}			multiple voltage		
m_i	Delay	V_{DD}	Power	Delay	V_{DD}	Power
3	0.35	3.3 V	0.04	1.00	1.7 V	0.01
5	0.70	3.3 V	0.10	1.00	2.7 V	0.07
7	0.75	3.3 V	0.12	1.00	3.0 V	0.10
11	1.00	3.3 V	0.25	1.00	3.3 V	0.25
17	1.00	3.3 V	0.26	1.00	3.3 V	0.26
64	0.65	3.3 V	0.24	1.00	2.7 V	0.16
tot			1.00			0.85

Table 3. Power dissipation for multiple supply voltage in tap.

tion seems to be reasonable because an increased short-circuit energy, due to longer transition times, is compensated by a suppression of some glitches, due to a longer gate delay. The table shows a reduction of about 15% in the power dissipation per tap.

The use of a multiple supply voltage requires level-shifting circuitry when going from the lower voltage to the higher one [11]. In our case, voltage level shifters are only required for a few bits before the output conversion stage.

7. Conclusions

In this work, we compared the performance, area and power of FIR filters realized with the traditional binary arithmetic, the RNS, and the RNS with a carry-save representation of residues.

Table 1 shows that the RNS filter can sustain the same throughput as the traditional one, but dissipates less power for filters with more than 4 taps. The CS-RNS implementation of the filter can be clocked almost at double speed, with respect to the traditional one, without a significant increase in area and energy dissipated in a cycle (the power increases linearly with frequency). Finally, we showed a possible implementation of the RNS filter with multiple supply voltage, which can lead to an additional reduction of the power dissipated, without affecting the performance of the filter.

Acknowledgments

This work was partially supported by MURST project: Codesign methods for low power integrated circuits.

References

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

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

[3] M.A. Soderstrand and K.Al Marayati. VLSI implementation of very high-order FIR filters. *IEEE International Symposium on Circuits and Systems (ISCAS'95)*, 2:1436–1439, 1995.

[4] S.Piestrak. A high-speed realization of a residue to binary number system converter. *IEEE Trans. Circuits Systems-II Analog and Digital Signal Processing*, 42:661–663, Oct. 1995.

[5] M. Re, A. Nannarelli, GC. Cardarilli, and R. Lojacono. FPGA Implementation of RNS to Binary Signed Conversion Architecture. *Proc. of IEEE International Symposium on Circuits and Systems*, IV:350–353, May 2001.

[6] I.M. Vinogradov. *An Introduction to the Theory of Numbers.* New York: Pergamon Press, 1955.

[7] A. Nannarelli, M. Re, and GC. Cardarilli. Tradeoffs between Residue Number System and Traditional FIR Filters. *Proc. of IEEE International Symposium on Circuits and Systems*, II:305–308, May 2001.

[8] Israel Koren. *Computer Arithmetic Algorithms.* Prentice-Hall, Inc., 1993.

[9] M.D. Ercegovac and T. Lang. *Division and Square Root: Digit-Recurrence Algorithms and Implementations.* Kluwer Academic Publisher, 1994.

[10] A. P. Chandrakasan and R. W. Brodersen. *Low Power Digital CMOS Design.* Kluwer Academic Publishers, 1995.

[11] K. Usami and M. Horowitz. Clustered voltage scaling technique for low-power design. *Proc. of International Symposium on Low Power Design*, pages 3–8, Apr. 1995.