Proceedings of 2009 APSIPA Annual Summit and Conference, Sapporo, Japan, October 4-7, 2009

An Algorithm for Bit-Serial Addition of SPT Numbers for Multiplierless Realization of Adaptive Equalizers

Sunav Choudhary, Pritam Mukherjee, Mrityunjoy Chakraborty
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur, INDIA
email:mrityun@ece.iitkgp.ernet.in

***Abstract*— The "sum of power of two (SPT)" is an effective format to represent multipliers in a digital filter which reduces the complexity of multiplication to a few shift and add operations. The canonic SPT is a special sparse SPT representation that guarantees occurrence of at least one zero between every two nonzero SPT bits. This paper presents a novel algorithm for bit serial addition of two numbers, each given in canonic SPT form, to produce a result also in canonic SPT. The proposed algorithm uses the properties of canonic SPT numbers effectively, resulting in considerable reduction of the hardware complexity of the bit serial adder. The algorithm is particularly useful for multiplier free realization of adaptive filters and equalizers, where the current weight vector and the update term, both presumed to be given in canonic SPT, are required to be added in a way that retains the canonic SPT format for the updated weight vector.**

I. Introduction

In a digital filter, the complexity of realization both in terms of silicon area and time is determined primarily by the multipliers. Consequently, efforts have been made to design filters that are free of multipliers. Ideally, if a multiplier is replaced by a single signed power of two term, the complexity reduces enormously since multiplication by a power of two amounts to a simple shift operation. However, the coefficient quantization error in such cases can be substantial affecting the filter performance considerably. A more effective approach for this is to approximate each multiplier by a sum of (signed) power of two (SPT) while keeping the number of power of two terms as few as possible. A well known sparse SPT representation in this context is the so-called canonic SPT [1]. Under this, a coefficient, say w, is represented as,

$$w \;=\; \sum_{r=1}^{R} s(r)\; 2^{g(r)}, \tag{1}$$

where $s(r) \in \{1, -1, 0\}$ is the r-th SPT coefficient, $g(r)$ is an increasing sequence of integers and R is the number of terms specified a priori. In canonic SPT, no two consecutive terms are nonzero (i.e., ± 1) simultaneously, i.e., if for any r, $s(r) = \pm 1$, then both $s(r+1$ and $s(r-1)$ must be zero (for example, $11 = 2^4 - 2^2 - 2^0$, $19 = 2^4 + 2^2 - 2^0$ etc.). In other words, the canonic SPT guarantees that at least $\lfloor \frac{R}{2} \rfloor$ SPT coefficients in (1) are zero. A circuit to convert 2's complement numbers into canonic SPT, both in bit serial and parallel form, has been presented in [1].

The SPT format has been used widely by researchers over years for efficient realization of fixed coefficient digital filters ([3]-[9]). The proposed algorithms are, however, offline techniques which can not be used for realizing adaptive filters whose coefficients change with time and thus can not be represented by a fixed SPT expression. In an adaptive filter (e.g., the LMS algorithm), the filter weights are updated as,

$$Future\_Weight \;=\; Current\_Weight \;+\; Update. \tag{2}$$

Assuming that both the $Current\_Weight$ and the $Update$ terms are available in canonic SPT form (the latter can be converted to canonic SPT by the circuit of [1]), it is then important to ensure that the summation in (2) generating $Future\_Weight$ produces the result in canonic SPT as well. Towards this objective, we present a technique in this paper for bit serial addition of two canonic SPT numbers producing the result also in canonic SPT.

II. Proposed Algorithm for SPT Addition

Let the two numbers which are to be added be $a = a_N a_{N-1} \cdots a_1 a_0$ $(\equiv \sum_{i=0}^{N} a_i\, 2^i)$ and $b = b_N b_{N-1} \cdots b_1 b_0$ $(\equiv \sum_{j=0}^{N} b_j\, 2^j)$ represented in canonic SPT forms, i.e., $a_i,\ b_j \in \{1, -1, 0\}$, with no two successive a_i's and b_j's taking nonzero values. In the proposed scheme, in the i-th cycle, we add a_i, b_i and the incoming carry c_i generated in the $(i-1)$-th cycle, and produce the new carry c_{i+1} and an intermediate result sp_i, which is to be adjusted to the final value s_i in the $(i+1)$ clock cycle. In other words, in the proposed scheme, there is a latency of one cycle between the i-th cycle input and the corresponding output. The proposed algorithm is given below where we use the notation 1^* to denote ± 1.

Algorithm : *Given a_i, b_i, c_i and sp_{i-1}, carry out the following steps at the i-th cycle :*
Step 1 (Addition) : Add a_i, b_i and c_i to produce c_{i+1} and sp_i.
Step 2 (Adjustment) : For adjustment, we utilize the following identities : $2^i + 2^{i-1} = 2^{i+1} - 2^{i-1}$ and $2^i - 2^{i-1} = 2^{i-1}$.
- *If $sp_i = 1^*$ and $sp_{i-1} = -1^*$, then adjust sp_i to 0 and take $s_{i-1} = 1^*$ as the output (of the previous cycle).*
- *If $sp_i = sp_{i-1} = 1^*$, then take $s_{i-1} = -1^*$, adjust sp_i*

to 0 *and propagate* 1^* *to the* $(i+1)$*-th step as* c_{i+1} [Note that for this case, c_{i+1} from Step 1 can not be 1^*, since this would imply that all the three bits, a_i, b_i and c_i are 1^* each simultaneously, which is, however, not possible, as shown in Lemma 1 below].

• No adjustment needed otherwise, meaning $sp_{i-1} \rightarrow s_{i-1}$.

As seen above, the four bits, a_i, b_i, c_i and sp_{i-1} are used to generate s_{i-1} and c_{i+1}. Theoretically, these four bits can have a total of $3^4 = 81$ combinations. However, as shown by Lemmas 1-3 below, only a fraction of these combinations are feasible while the remaining ones can not come up. This results in considerable savings in hardware as one can use the so-called "don't care" states for the invalid combinations.

Lemma 1 : *The three bits,* a_i*,* b_i *and* c_i *can not be non-zero simultaneously.*

Proof : Suppose that the three bits, a_i, b_i and c_i are non-zero simultaneously. From the characteristics of the canonic SPT format, this implies that $a_{i-1} = 0$ and $b_{i-1} = 0$. The only possible way to maintain c_i non-zero in this case is to have $c_{i-1} = 1^*$ and $sp_{i-2} = 1^*$ (in the $(i-1)$-th cycle), which would lead to the following adjustments/assignments, as per the algorithm above : $sp_{i-1} \rightarrow 0$, $sp_{i-2} \rightarrow s_{i-2} = -1^*$ and $c_i = 1^*$. The combination, $c_{i-1} = 1^*$ and $sp_{i-2} = 1^*$ can, however, occur only when $a_{i-2} = 1^*$, $b_{i-2} = 1^*$ and $c_{i-2} = 1^*$, i.e., all the three bits, a_{i-2}, b_{i-2} and c_{i-2} are nonzero. Proceeding recursively, for i even, this would then mean that the bits, a_0, b_0 and c_0 are nonzero simultaneously, which is, however, not possible, since, in the proposed scheme, we always have $c_0 = 0$. Again, for i odd, the above means a_1, b_1 and c_1 are nonzero simultaneously. However, c_1 can not be non-zero, since, from the canonic SPT property, we have, in this case, $a_0 = 0$, $b_0 = 0$ and separately, $c_0 = 0$. Hence proved.

Lemma 2 : *If exactly one of the four bits,* a_i*,* b_i*,* c_i *and* sp_{i-1} *is zero, then it has to be* c_i*.*

Proof : Suppose, $a_i = 0$ and $b_i \neq 0$, $c_i \neq 0$ and $sp_{i-1} \neq 0$. From the canonic SPT property, it then follows that $b_i = 0$. To have $sp_{i-1} \neq 0$, one of the two bits, a_{i-1} and c_{i-1} must be nonzero, which, however, implies $c_i = 0$ and thus a contradiction. Same logic applies to the case where $b_i = 0$ and the remaining three bits are nonzero. Again, if $sp_{i-1} = 0$, we have, a_i, b_i and c_i nonzero simultaneously, which is not permitted as per Lemma 1. Hence, the only possibility is $c_i = 0$.

Lemma 3 : *If exactly two of the four bits,* a_i*,* b_i*,* c_i *and* sp_{i-1} *are zero, then at least one of them has to be* c_i *or* sp_{i-1}*.*

Proof : Suppose $a_i = b_i = 0$ and $c_i \neq 0$, $sp_{i-1} \neq 0$. In this case, to maintain $c_i \neq 0$, $sp_{i-1} \neq 0$, all the three bits, a_{i-1}, b_{i-1} and c_{i-1} have to be nonzero simultaneously which is, however, not permissible as per Lemma 1. Hence proved.

III. Circuit Implementation

As seen above, the algorithm is simply a rule to transform the pair $(c_i + a_i + b_i,\ sp_{i-1})$ to the triplet (c_{i+1}, sp_i, s_{i-1}). Define three functions, $f_c(a_i, b_i, c_i, sp_{i-1})$, $f_{sp}(a_i, b_i, c_i, sp_{i-1})$ and $f_s(a_i, b_i, c_i, sp_{i-1})$ which generate the quantities c_{i+1}, sp_i and s_{i-1} respectively at the i-th cycle following the above algorithm. The truth table for each function is displayed in Table I, where we have used Lemmas 1-3 to reduce the number of combinations to just 37 from $3^4 = 81$. The corresponding block diagram for hardware realization

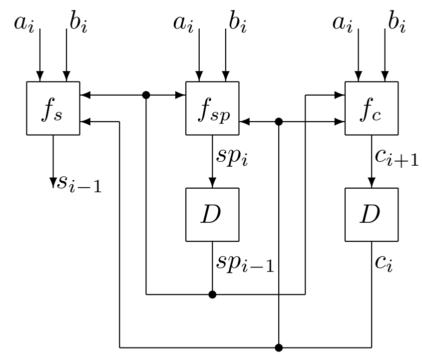

Fig. 1. Block Diagram of Bit-Serial Adder for numbers in Cannonical SPT form

of the bit serial adder is shown in Fig. 1. Here f_c, f_{sp} and f_s are combinatorial blocks which implement the respective truth tables given in Table 1. Note that since the SPT representation uses altogether three bits, namely, $1, -1, 0$, in a digital implementation, each SPT bit is represented by two binary bits, as per the following : $(1)_{SPT} = (01)_2$, $(0)_{SPT} = (00)_2$ and $(\bar{1})_{SPT} = (11)_2$, which is consistent with 2's complement representation of signed binary numbers.

References

[1] Y.C. Lim, J.B. Evans and B. Liu, "Decomposition of Binary Integers into Signed Power-of-Two Terms", *IEEE Trans. Circuits and Systems*, vol.38, no.6, pp. 667-672, June, 1991.

[2] Y.C. Lim, R. Yang, D. Li and J. Song, "Signed power-of-two term allocation scheme for the design of digital filters", *IEEE Trans. Circuits and Systems, part II*, vol.46, no.5, pp. 577-584, May, 1999.

[3] H.H. Dam, A. Cantoni, K.L. Teo and S. Nordholm, "FIR Variable Digital Filter With Signed Power-of-Two Coefficients", *IEEE Trans. Circuits and Systems, Part I*, vol.54, no.6, pp. 1348-1357, June 2007.

[4] S.Y. Park, N.I. Cho, "Design of Multiplierless Lattice QMF: Structure and Algorithm Development", IEEE Trans. Circuits and Systems, part II, vol.55, no.2, pp. 173-177, Feb. 2008.

[5] Z.G. Feng, K.L. Teo, "A Discrete Filled Function Method for the Design of FIR Filters With Signed-Powers-of-Two Coefficients", *IEEE Trans. Signal Processing*, vol.56, no.1, pp. 134-139, Jan. 2008.

[6] M. Aktan, A. Yurdakul and G. Dundar, "An Algorithm for the Design of Low-Power Hardware-Efficient FIR Filters", *IEEE Trans. Circuits and Systems part I*, vol.55, no.6, pp. 1536-1545, July 2008.

[7] Y.J. Yu and Y.C. Lim, "Design of Linear Phase FIR Filters in Subexpression Space Using Mixed Integer Linear Programming", *IEEE Trans. Circuits and Systems part I*, vol.54, no.10, pp. 2330-2338, Oct. 2007.

[8] F. Xu, C.H. Chang and C.C. Jong, "Design of Low-Complexity FIR Filters Based on Signed-Powers-of-Two Coefficients With Reusable Common Subexpressions", *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol.26, no.10, pp. 1898-1907, Oct. 2007.

[9] S.Y. Park and N.I. Cho, "Design of signed powers-of-two coefficient perfect reconstruction QMF Bank using CORDIC algorithms", *IEEE Trans. Circuits and Systems part I*, vol.53, no.6, pp. 1254-1264, June, 2007.

TABLE I

TRUTH TABLE FOR BIT-SERIAL ADDED OUTPUT

External Inputs		Internal Inputs		Internal Outputs		External Output
a_i	b_i	c_i	sp_{i-1}	$c_{i+1} = f_c(a_i, b_i, c_i, sp_{i-1})$	$sp_i = f_{sp}(a_i, b_i, c_i, sp_{i-1})$	$s_{i-1} = f_s(a_i, b_i, c_i, sp_{i-1})$
0	0	0	0	0	0	0
0	0	1	0	0	1	0
0	1	0	0	0	1	0
1	0	0	0	0	1	0
0	0	$\bar{1}$	0	0	$\bar{1}$	0
0	$\bar{1}$	0	0	0	$\bar{1}$	0
$\bar{1}$	0	0	0	0	$\bar{1}$	0
0	1	1	0	1	0	0
1	0	1	0	1	0	0
1	1	0	0	1	0	0
0	$\bar{1}$	$\bar{1}$	0	$\bar{1}$	0	0
$\bar{1}$	0	$\bar{1}$	0	$\bar{1}$	0	0
$\bar{1}$	$\bar{1}$	0	0	$\bar{1}$	0	0
$\bar{1}$	1	0	0	0	0	0
1	$\bar{1}$	0	0	0	0	0
0	$\bar{1}$	1	0	0	0	0
$\bar{1}$	0	1	0	0	0	0
0	1	$\bar{1}$	0	0	0	0
1	0	$\bar{1}$	0	0	0	0
0	0	0	1	0	0	1
0	1	0	1	1	0	$\bar{1}$
1	0	0	1	1	0	$\bar{1}$
0	$\bar{1}$	0	1	0	0	$\bar{1}$
$\bar{1}$	0	0	1	0	0	$\bar{1}$
1	1	0	1	1	0	1
$\bar{1}$	$\bar{1}$	0	1	$\bar{1}$	0	1
$\bar{1}$	1	0	1	0	0	1
1	$\bar{1}$	0	1	0	0	1
0	0	0	$\bar{1}$	0	0	$\bar{1}$
0	1	0	$\bar{1}$	0	0	1
1	0	0	$\bar{1}$	0	0	1
0	$\bar{1}$	0	$\bar{1}$	$\bar{1}$	0	1
$\bar{1}$	0	0	$\bar{1}$	$\bar{1}$	0	1
1	1	0	$\bar{1}$	1	0	$\bar{1}$
$\bar{1}$	$\bar{1}$	0	$\bar{1}$	$\bar{1}$	0	$\bar{1}$
$\bar{1}$	1	0	$\bar{1}$	0	0	$\bar{1}$
1	$\bar{1}$	0	$\bar{1}$	0	0	$\bar{1}$