A SPT Treatment to the Realization of the Sign-LMS Based Adaptive Filters

Sunav Choudhary, Pritam Mukherjee, Mrityunjoy Chakraborty, *Senior Member, IEEE*, and Shakti Shankar Rath

***Abstract*—The "sum of power of two (SPT)" is an effective format to represent filter coefficients in a digital filter which reduces the complexity of multiplications in the filtering process to just a few shift and add operations. The canonic SPT is a special sparse SPT representation that guarantees presence of at least one zero between every two non-zero SPT digits. In the case of adaptive filters, as the coefficients are updated with time continuously, conversion to such canonic SPT forms is, however, required at each time index, which is quite impractical and requires additional circuitry. Also, as the position of the non-zero SPT terms in each coefficient word changes with time, it is not possible to carry out multiplications involving the coefficients via a few *fixed* "shift and add" operations. This paper addresses these problems, in the context of a SPT based realization of adaptive filters belonging to the sign-LMS family. Firstly, it proposes a bit serial adder that takes as input two numbers in canonic SPT and produces an output also in canonic SPT, which is then extended to the case where one of the inputs is given in 2's complement form. This allows weight updating purely in the canonic SPT domain. It is also shown how the canonic SPT property of the input can be used to reduce the complexity of the proposed adder. For multiplication, the canonic SPT word for each coefficient is partitioned into non-overlapping digit pairs and the data word is multiplied by each pair separately. The fact that each pair can have at the most one non-zero digit is exploited further to reduce the complexity of the multiplication.**

***Index Terms*—Bit serial realization, canonic SPT, sign-LMS algorithm, sum of power of two (SPT).**

I. Introduction

A POPULAR way of reducing multiplicative complexities in a digital filter is to look for a representation of the multiplier coefficient that is sufficiently sparse, as this results in evaluation of the partial products only for the non-zero bits. One way to achieve this is to approximate each multiplier coefficient 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] form, also popularly known as the canonic signed digit (CSD) format [2]. 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 (also called "SPT digit" in this paper), $g(r)$ is an increasing sequence of integers and R is the number of terms specified a priori. In canonic SPT, no two consecutive terms are non-zero (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 $\lfloor R/2 \rfloor$ SPT coefficients in (1) are zero ($\lfloor \cdot \rfloor$ denotes floor operation).

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 result in fixed SPT format for each filter coefficient. Noting the position of the non-zero SPT terms, circuitry can then be developed to implement the corresponding "shift and add" operations. Such approaches, working very well for fixed coefficient filters, however, face certain difficulties when used to realize adaptive filters. Firstly, the coefficients in an adaptive filter are updated continuously, meaning the coefficients need to be converted into the chosen sparse format at *each index* by engaging additional hardware, before using them in the multiplication. Secondly, as the positions of the non-zero terms in the converted coefficient word are not fixed but change with time, hardware needed to implement the corresponding shift and add operations becomes more complex. Presumably, due to the above difficulties, not much work seems to have been done so far in the area of canonic SPT based adaptive filter realization. In [10], a group SPT based approach has been presented which divides the coefficient word into non-overlapping blocks, with each block permitted to have only one non-zero term. Updating of the coefficients is then done by moving up or down in each block by one position. This, however, introduces substantial quantization errors as both the coefficients and the update terms are allowed to assume only a few possible values. Also, since the position of the non-zero digit in each group is time varying, additional, non-trivial overheads are necessary in the multiplication of data by each block. In [11], the coefficients are updated by 2's complement addition first, followed by conversion to canonic SPT/CSD for which an improved 2's complement-to-CSD converter has been proposed. The multiplier proposed in [11] for the filtering part, however, can not make use of the canonic SPT property of the coefficient, when used in a synchronous implementation, requiring provision of shift and add in each clock cycle.

Manuscript received March 13, 2011; revised July 29, 2011; accepted December 21, 2011. Date of publication June 20, 2012; date of current version August 24, 2012. This work has been supported in part by a research grant from the ministry of information technology, Government of India. This paper was recommended by Associate Editor M. Mondin.

S. Choudhary is with the Department of Electrical Engineering., University of Southern California, Los Angeles, CA 90024 USA (e-mail: sunavch@gmail.com).

P. Mukherjee is with the Department of Electrical and Computer Engineering., University of Maryland, College Park, MD 20742 USA (email: pmmukherjee.iit@gmail.com).

M. Chakraborty is with the Department of Electronics and Electrical Communication Engineering, Indian Institute of Technology, Kharagpur 721302, India (e-mail: mrityun@ece.iitkgp.ernet.in).

S. S. Rath is with the Texas Instruments, Bangalore 560093, India (email: shakti@ti.com).

Digital Object Identifier 10.1109/TCSI.2012.2185300

This paper aims to overcome the above limitations by proposing an alternative approach, in the context of adaptive filters belonging to the sign-LMS family[1]. In particular, it considers three popular sign-LMS based adaptive filters, namely, the sign-sign, the sign and the signed regressor algorithms [12], with respective weight update relations given as follows:

A) Sign-Sign Algorithm:

$$\mathbf{w}(l+1) = \mathbf{w}(l) + \mu\,\mathrm{sign}(\mathbf{x}(l))\,\mathrm{sign}(e(l)) \tag{2}$$

B) Sign Algorithm:

$$\mathbf{w}(l+1) = \mathbf{w}(l) + \mu\mathbf{x}(l)\,\mathrm{sign}(e(l)) \tag{3}$$

C) Signed Regressor Algorithm:

$$\mathbf{w}(l+1) = \mathbf{w}(l) + \mu\,\mathrm{sign}(\mathbf{x}(l))e(l) \tag{4}$$

where, $\mathbf{w}(l) = [w_0(l), w_1(l), \ldots, w_{p-1}(l)]^t$ is a p-th order filter weight vector at the l-th index, $x(l)$ is the filter input, $d(l)$ is a desired response, $\mathbf{x}(l) = [x(l), x(l-1), \ldots, x(l-p+1)]^t$, $e(l) = d(l) - \mathbf{w}^t(l)\mathbf{x}(l)$ is the output error, μ is the algorithm step size and "$\mathrm{sign}(\cdot)$" is the well known signum function. Both $x(l)$ and $d(l)$ are assumed to be available in the usual 2's complement form. Unlike the standard LMS algorithm, the sign-LMS algorithms (A)–(C) above are free of the multiplication term $\mathbf{x}(l)e(l)$ in the weight update loop, which saves the hardware cost by almost fifty percent. Of the algorithms (A)–(C), the sign-sign update relation for a tap weight, say, $w_j(n), \quad j = 0, 1, \ldots, p$, is given by $w_j(n+1) = w_j(n) \pm \mu$, meaning the only operations needed are addition/subtraction. For the sign and the signed regressor algorithms, the corresponding update term involves a product with the step size μ in the form of $\mu x(n-j)$ and $\mu e(n)$ respectively. To avoid an explicit multiplication, a standard practice for these two cases is to choose μ to be a power of two from its permissible range. The only multiplications present in the adaptive filter are then those occurring in the computation of the filter output $\mathbf{w}^t(l)\mathbf{x}(l)$.

To reduce the complexity of the above multiplications, the proposed treatment represents each filter coefficient $w_j(l), \quad j = 0, 1, \ldots, p-1$ in canonic SPT format and develops appropriate SPT addition algorithms that produce the updated weight $w_j(l+1)$ also in canonic SPT form. For this, first a bit serial adder is developed that takes both the input in canonic SPT format and generates the output also in canonic SPT, which is later generalized to the case where one of the inputs is changed to the 2's complement form, for which an alternative, relatively simpler architecture is feasible. The proposed addition algorithms enjoy certain optimizing properties arising out of the canonic SPT form of the input, as shown through a set of lemmas, which are used effectively to reduce the complexity of the logical blocks in the proposed adders. Multiplications involving $w_j(l)$ (e.g., $w_j(l)x(l-j)$) which occur in the filtering part, however, do not automatically lead to a low complexity realization, as the position of the nonzero SPT terms in the canonic SPT expression of $w_j(l)$ keep changing continuously over the index l. Towards this, a scheme is developed that pairs the adjacent digits of $w_j(l)$ to form non-overlapping digit pairs and develops circuits to multiply $x(l-j)$ by each digit pair separately. The fact that each digit pair of $w_j(l)$ can have at the most one non-zero digit is then exploited to reduce the complexity of multiplication effectively.

[1]Some preliminary results of this paper had been presented by the authors at ISCAS-2010, Paris, 2010 [14] and also at APSIPA ASC 2010, Singapore, 2010 [15].

II. Algorithm for Addition of Canonic SPT Numbers

A. Addition With Both Inputs in Canonic SPT

Let the two numbers which are to be added be $A = a_N a_{N-1} \cdots a_1 a_0 \quad (\equiv \sum_{i=0}^{N-1} a_i 2^i)$ and $B = b_N b_{N-1} \cdots b_1 b_0 \quad (\equiv \sum_{j=0}^{N-1} b_j 2^j)$ 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 non-zero 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 ($c_0 = 0$), 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 $(i+1)$-th clock cycle, in order to maintain the canonic SPT form of the bit serial output. 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 $sp_{i-1}, i = 0, 1, \ldots, N-1$ ($c_0 = 0$, $sp_{-1} = 0$), carry out the following steps at the i-th cycle:

Step 1 *(Addition): Add a_i, b_i, c_i to produce c_{i+1} and sp_i, as per the following: $c_{i+1} = sign(d)\lfloor \frac{|d|}{2} \rfloor$ and $sp_i = sign(d)(|d| mod\ 2)$, where d is the decimal equivalent of $a_i + b_i + c_i$.* [*Clearly, $c_{i+1}, sp_i \in \{1, -1, 0\}$.*]

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 cannot 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 digits, a_i, b_i, c_i and sp_{i-1} are used to generate sp_i, s_{i-1} and c_{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 evaluate the quantities c_{i+1}, sp_i and s_{i-1} respectively at the i-th cycle following the above algorithm. Then, the above algorithm results in a digit serial implementation as shown in Fig. 1, where f_c, f_{sp} and f_s are combinatorial blocks implementing the functions $f_c(a_i, b_i, c_i, sp_{i-1}), \quad f_{sp}(a_i, b_i, c_i, sp_{i-1})$ and $f_s(a_i, b_i, c_i, sp_{i-1})$ respectively. The truth table of each combinational block is easy to generate following the algorithm. Theoretically, each truth table can have $3^4 = 81$ rows, since each block has four inputs with each input taking one of the three values $1, -1, 0$. However, the proposed addition algorithm has certain optimizing properties as captured by Lemmas 1–3 below, which ensure that only a fraction of the above combinations are feasible while the remaining ones cannot come up. *This results in considerable savings in hardware as*

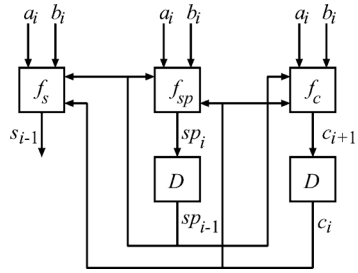

Fig. 1. Block Diagram for Bit-Serial Addition of Numbers in Canonic SPT form.

one can use the so-called "don't care" states for the invalid combinations.

Lemma 1: The three digits, a_i, b_i and c_i cannot be non-zero simultaneously.

Proof: Suppose that the three digits, 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 digits, a_{i-2}, b_{i-2} and c_{i-2} are non-zero. Proceeding recursively, for i even, this would then mean that the digits, a_0, b_0 and c_0 are non-zero 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 non-zero simultaneously. However, c_1 cannot 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 digits, a_i, b_i, c_i and sp_{i-1} is zero, then it has to be c_i.

Proof: Firstly, the case $i = 0$ does not arise here, since, for $i = 0$, at least two digits, namely, c_0 and sp_{-1} are zero. Suppose, for $i = 1, 2, \ldots, N-1$, $a_i = 0$ and $b_i \neq 0$, $c_i \neq 0$ and $sp_{i-1} \neq 0$. From the canonic SPT property, it then follows that $b_{i-1} = 0$. To have $sp_{i-1} \neq 0$, one of the two bits, a_{i-1} and c_{i-1} must be non-zero, 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 non-zero. Again, if $sp_{i-1} = 0$, we have, a_i, b_i and c_i non-zero 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 digits, 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: For $i = 0$, the proof is trivial. Suppose, for $i = 1, 2, \ldots, N-1$, $a_i = b_i = 0$ 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 digits, a_{i-1}, b_{i-1} and c_{i-1} have to be non-zero simultaneously which is, however, not permissible as per Lemma 1. Hence, proved.

The Lemmas 1–3 reduce the number of rows in each truth table from 81 to just 37, resulting in considerable simplification of the hardware. Finally, in a digital implementation, each SPT digit is represented by a two bit 2's complement word, as per the following: $(1)_{\mathrm{SPT}} = (01)_2$, $(0)_{\mathrm{SPT}} = (00)_2$ and $(\bar{1})_{\mathrm{SPT}} = (11)_2$. Such representation has the advantage in that one can use the same adder for carrying out subtractions like $A - B$, for which one simply has to take the 2's complement of each digit b_i of B before feeding to the adder.

B. Addition With One Input in Canonic SPT and the Other in 2's Complement

We consider the addition of an N digit or, equivalently, a $2N$ bit canonic SPT number A as given above, with an N bit 2's complement number $B : b_{N-1} \cdots, b_1, b_0$, with decimal value given by $-b_{N-1} + \sum_{i=1}^{N-1} b_{N-1-i} 2^{-i}, b_i \in \{1, 0\}$. For $i = 0, 1, \ldots, N-2$, each binary bit b_i can be treated as a SPT digit that takes just two values, 0 and 1, while for $i = N-1$ (i.e., the sign bit), b_{N-1} is equivalent to a SPT digit that takes values 0 and -1. The addition of A and B can then be carried out by the canonic SPT addition algorithm proposed above by feeding it with the input digits a_i and b_i in the i-th cycle[2] (this means, in the adder developed above, the b_i input is to be provided with $0b_i$ for $i = 0, 1, \ldots, N-2$ and with $b_{N-1}b_{N-1}$ for $i = N-1$). Note that the input B, though in SPT form, is not necessarily *canonic* in this case (i.e., if for any r, $b_r \neq 0$, there is no guarantee that both b_{r+1} and b_{r-1} must be zero). It is, however, easy to verify that the addition algorithm proposed in Section II.A will still produce the output $A + B$ in canonic SPT format only, by virtue of the *adjustments* suggested. Also note that the Lemmas 1–3 do not apply in this case as one of the inputs, namely, B is not given in canonic SPT form. However, the fact that the other input A is in canonic SPT and also that each bit of B is only binary, gives rise to a new set of optimizing properties, as reflected through the following lemmas, that can be used to reduce the number of rows in the truth table of each of the blocks f_c, f_{sp} and f_s of Fig. 1 for this case from 54 to 22.

Lemma 4: The three digits, a_i, c_i and sp_{i-1} cannot be non-zero simultaneously.

Proof: For $i = 0$, the proof is trivial, since $sp_{-1} = 0$, i.e., the combination of $a_0 \neq 0$, $c_0 \neq 0$ and $sp_{-1} \neq 0$ simultaneously does not arise. For $i = 1, 2, \ldots, N-1$, suppose $a_i \neq 0$, $c_i \neq 0$ and $sp_{i-1} \neq 0$ simultaneously. From the property of the canonic SPT format, this implies that $a_{i-1} = 0$. Also, for $i = 1, 2, \ldots, N-1$, b_{i-1} takes values 0 or 1 in the addition $a_{i-1} + b_{i-1} + c_{i-1}$. Therefore, for sp_{i-1} to be nonzero, we should have either (a) $b_{i-1} = 1$, $c_{i-1} = 0$ and $sp_{i-2} = 0$, or, (b) $b_{i-1} = 0$, $c_{i-1} = 1^*$ and $sp_{i-2} = 0$ (the condition $sp_{i-2} = 0$ ensures that no adjustment is applied to sp_{i-1} changing it zero). This, however, implies that $c_i = 0$ which is a contradiction. Hence, proved.

Note that if the addition $a_i + b_i + c_i$ produces a non-zero carry digit c_{i+1}, then the Lemma 4 ensures that no separate non-zero carry c_{i+1} will be generated by the adjustment of sp_i and sp_{i-1} (i.e., no conflict in carry will result). This is because a non-zero carry from $a_i + b_i + c_i$ can result under two possibilities, namely, (i) $a_i = c_i = 1^*$ ($b_i = 0$ or 1 as may be necessary to generate a non-zero carry), meaning, from Lemma 4, $sp_{i-1} = 0$ and thus, no adjustment needed on sp_i, and, (ii)

$a_i = 1, c_i = 0$, or, $a_i = 0, c_i = 1$, with $b_i = 1$ in both cases, meaning $sp_i = 0$ and again, no adjustment needed on sp_i.

Lemma 5: The carry digit c_i cannot be equal to -1

Proof: First note that for $c_i = -1$, $i = 1, 2, \ldots, N-1$ to hold, $b_{i-1} \neq 1$, (i.e., $b_{i-1} = 0$). This is because, with $b_{i-1} = 1$,

[2]The same algorithm can be used to carry out the subtraction $A - B$ also for which B is to be 2's complemented, i.e., the b_i input is to be replaced by $\bar{b}_i$ and the initial carry c_0 is to be taken as 1.

we can have $c_i = -1$ only if $a_{i-1} = c_{i-1} = sp_{i-2} = -1$ so that, after adjustment, sp_{i-1} becomes zero and c_i becomes -1. From Lemma 4, this is, however, not possible. Next, for $i = 0$, this Lemma is trivially satisfied since c_0 can only be either 0 (when we are computing $A + B$) or 1 (when we are computing $A - B$), i.e., $c_0 \neq -1$. For $i = 1, 2, \ldots, N-1$, $c_i = -1$, however, implies three possibilities, namely,

1) $a_{i-1} = 0$ and $c_{i-1} = sp_{i-2} = -1$ ($b_{i-1} = 0$ as per above), so that after adjustment, $sp_{i-1} \rightarrow 0$ and $c_i \rightarrow -1$.
2) $a_{i-1} = c_{i-1} = -1$ ($b_{i-1} = 0$ as per above), so that directly we have, $sp_{i-1} = 0$ and $c_i = -1$ (irrespective of sp_{i-2}).
3) $c_{i-1} = 0$ and $a_{i-1} = sp_{i-2} = -1$ ($b_{i-1} = 0$ as per above), so that after adjustment, $sp_{i-1} \rightarrow 0$ and $c_i \rightarrow -1$. [Note that in all the combinations A, B and C above, $sp_{i-1} = 0$.]

In the following, we show that none of A, B, or C is possible. First consider A, i.e., $c_{i-1} = sp_{i-2} = -1$. For $i = 1$, it is trivially infeasible as $c_0 \in \{1, 0\}$, i.e., $c_0 \neq -1$ and $sp_{-1} = 0$. For $i = 2, 3, \ldots, N-1$, this is possible if and only if $b_{i-2} = a_{i-2} = c_{i-2} = -1$ while $sp_{i-3} = 0$. However, $b_{i-2} \neq -1$ and hence it is not possible. Next we consider B. For $i = 1$, again it is trivially impossible as $c_0 \in \{1, 0\}$, i.e., $c_0 \neq -1$. For $i = 2, 3, \ldots, N-1$, $a_{i-1} = -1$ implies $a_{i-2} = 0$ (from the canonic SPT property) and $c_{i-1} = -1$ implies $b_{i-2} = 0$ (from above). Therefore, it is possible to have $c_{i-1} = -1$ if and only if $c_{i-2} = sp_{i-3} = -1$ so that, after adjustment, $sp_{i-2} \rightarrow 0$ and $c_{i-1} \rightarrow -1$. This is, however, same as case A above (with i replaced by $(i-1)$) and therefore is not possible. Finally, for C, first consider $i = 1$, i.e., $c_1 = -1$ which is not possible since $sp_{-1} = 0$. For $i = 2, 3, \ldots, N-1$, $a_{i-1} = -1$ implies $a_{i-2} = 0$ (from the canonic SPT property) and $sp_{i-2} = -1$ implies $sp_{i-3} = 0$ (as otherwise, after adjustment, $sp_{i-2} \rightarrow 0$). Since b_{i-2} cannot be -1, the only way one can have $sp_{i-2} = -1$ is $c_{i-2} = -1$ (along with $b_{i-2} = 0$). The combination $c_{i-2} = -1$ and $sp_{i-3} = 0$, however, means one of the three possibilities A, B or C with i replaced by $(i-2)$. Out of these, A and B are not feasible as proved above, meaning only possibility is C. Proceeding recursively backwards, this then means the following: (i) for i even, $c_i = c_{i-2} = \cdots = c_0 = -1$ and $sp_{i-1} = sp_{i-3} = \cdots = sp_{-1} = 0$, which is not possible since $c_0 \neq -1$, and, (ii) for i odd, $c_i = c_{i-2} = c_{i-4} = \cdots = c_1 = -1$ and $sp_{i-1} = sp_{i-3} = \cdots = sp_0 = 0$. Now, $c_1 = -1$ implies $b_0 = 0$. Since $a_0 \in \{1, 0, -1\}$, $c_0 \in \{1, 0\}$ and also, $sp_{-1} = 0$ meaning no adjustment needed on sp_0, it is easy to check that the addition $a_0 + b_0 + c_0$ cannot simultaneously produce $sp_0 = 0$ and $c_1 = -1$. Hence, proved.

Lemma 6: If $sp_{i-1} = -1$, then we always have $a_i = c_i = 0$.

Proof: Firstly, $sp_{i-1} = -1$ implies $sp_{i-2} = 0$. Because, if $sp_{i-2} = 1^*$, then the adjustment process in the $(i-1)$-th cycle will always produce $sp_{i-1} = 0$, i.e., if $a_{i-1} + c_{i-1} + b_{i-1}$ produces 1^* as the sum output, sp_{i-1} will be adjusted to 0 whereas if $a_{i-1} + c_{i-1} + b_{i-1}$ produces 0, no adjustment will be necessary and again, we will have $sp_{i-1} = 0$. On the other hand, for $sp_{i-2} = 0$, since no adjustment is necessary, we can have $sp_{i-1} = -1$ if and only if $a_{i-1} + c_{i-1} + b_{i-1}$ produces -1 as the sum output, which is, however, possible only for $a_{i-1} = -1$ and $c_{i-1} = b_{i-1} = 0$, since both b_{i-1} and c_{i-1} (from Lemma 5) are binary, taking values 0 or 1 only. Obviously, the addition $a_{i-1} + c_{i-1} + b_{i-1}$ in this case will not produce any carry, i.e., $c_i = 0$, and since $a_{i-1} = -1$, from the canonic SPT property of A, $a_i = 0$. Hence, proved.

An Alternate Circuit to Carry Out the Addition: As discussed above, the circuit of Fig. 1 can be used to implement the above addition algorithm, with certain modifications in the b_i input. However, for the case of addition of a canonic SPT number with a 2's complement number, an alternate realization is feasible that is simpler than the above. This realization uses two separate blocks, say, block 1 and block 2 for carrying out the steps 1 and 2 of the proposed algorithm respectively, with the logical complexities of each block simplified by appropriate use of the Lemmas 4–6. For this, first note, from Lemma 5, that $c_i \in \{0, 1\}$ and, separately, for $i = 0, 1, \ldots, N-2$, $b_i \in \{0, 1\}$, meaning we need just one bit to represent each of b_i and c_i (the case of b_i for $i = N-1$ shall be handled separately). The block 1 then takes two 1-bit inputs, b_i and c_i, and a two bit canonic SPT input $a_i = a_{i,1}a_{i,0}$, and performs the addition $a_i + b_i + c_i$ to generate a two bit, canonic SPT output $d_i = d_{i,0}d_{i,1}$ and an intermediate carry output $c1_{i+1}$. Block 2 takes two 2-bit canonic SPT inputs, namely, d_i (from Block 1) and $sp_{i-1} = sp_{i-1,1}sp_{i-1,0}$ (generated at its output in the previous cycle), and carries out the adjustments, as given in the step 2 of the algorithm, to produce two 2-bit canonic SPT outputs, sp_i and $s_i = s_{i,1}s_{i,0}$ and another intermediate carry $c2_{i+1}$.

Now, as seen earlier, a fallout of Lemma 4 is that both $c1_{i+1}$ and $c2_{i+1}$ cannot be non-zero simultaneously, meaning, only the following combinations are possible: (a) $c1_{i+1} = 1^*, c2_{i+1} = 0$, (b) $c1_{i+1} = 0, c2_{i+1} = 1^*$, and (c) $c1_{i+1} = 0, c2_{i+1} = 0$. The final carry c_{i+1}, which is given by $c1_{i+1} + c2_{i+1}$, is, however, binary from Lemma 5, with $c_{i+1} \in \{0, 1\}$. This is, however, possible if and only if $c1_{i+1}, c2_{i+1} \in \{0, 1\}$, i.e., both $c1_{i+1}$ and $c2_{i+1}$ are binary. In other words, the blocks 1 and 2 produce two one bit intermediate carry outputs $c1_{i+1}$ and $c2_{i+1}$ respectively and the final carry c_{i+1} is obtained by logical OR of $c1_{i+1}$ and $c2_{i+1}$.

The proposed adder architecture is shown in Fig. 2, while Table I shows the logical expressions of each block output bit in terms of the respective input bits [the truth table for each block is easy to construct and is not described here]. In the actual implementation, further simplifications are possible by looking for possible common terms in the logical expressions given in Table I, e.g., $\overline{sp}_{i-1,0}$ is common for $sp_{i,1}$ and $sp_{i,0}$, $\overline{sp}_{i-1,1}$ is common for $s_{i-1,1}$ and $c2_{i+1}$ etc. For the MSB, i.e., for $i = N-1$, firstly the truth table won't change for $b_i = 0$. For $b_i = 1$ (meaning "-1" under canonic SPT), it is easy to check that $d_{i,0}$ in the truth table will not change whereas $d_{i,1}$ will change from 0 to 1, only for the two cases: (a) $b_i = c_i = a_{i,1} = a_{i,0} = 1$ and (b) $b_i = 1, c_i = a_{i,1} = a_{i,0} = 0$. Clearly, $d_{i,1}$ in this case will be given by the logical OR of the $d_{i,1}$ output of block 1 with the signal $d'_i = b_i \cdot (a_{i,1} \cdot c_i + \bar{a}_{i,0} \cdot \bar{c}_i)$ [we ignore the carry out $c1_{i+1}$ for the MSB case under "no-overflow" assumption; also, in all the truth tables, "10" is taken as an invalid combination for any 2 bit canonic SPT digit, giving rise to the "don't care" states for the output].

III. The Proposed Implementation

Assume that the filter weights at the l-th index, $w_j(l), j = 0, 1, \ldots, p-1$ have the following canonic SPT representations: $w_j(l) = \sum_{i=0}^{N-1} w_{i,j}(l) 2^{-(N-1-i)}$, with $w_{i,j}(l) \in \{1, -1, 0\}$. As seen in Section I, the coefficient $w_j(l)$ is updated as

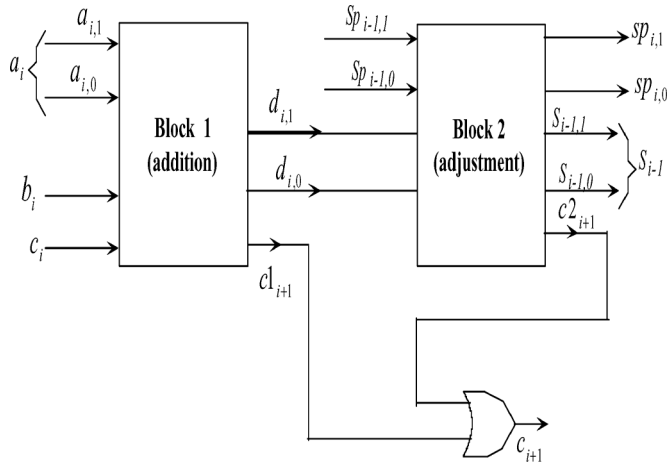

Fig. 2. Proposed bit serial adder for adding a canonic SPT number A with a 2's complement number B. The i-th cycle operations are shown where a_i : $a_{i,1}a_{i,0}$ is the i-th digit of A, b_i is the i-th bit of B and c_i is the i-th incoming carry bit, while s_{i-1} : $s_{i-1,1}s_{i-1,0}$ is the $(i-1)$-th output digit.

TABLE I
LOGICAL EXPRESSIONS OF THE BLOCK 1 AND 2 OUTPUT BITS IN TERMS OF RESPECTIVE INPUT BITS

Block 1

$$d_{i,1} = a_{i,1}.\overline{b}_i.\overline{c}_i,\ d_{i,0} = a_{i,0} \oplus b_i \oplus c_i.$$
$$c1_{i+1} = \overline{a}_{i,1}.[a_{i,0}.b_i + b_i.c_i + a_{i,0}.c_i].$$

Block 2

$$s_{i-1,0} = sp_{i-1,0}.$$
$$s_{i-1,1} = d_{i,0}.\overline{sp}_{i-1,1}.sp_{i-1,0} + \overline{d}_{i,0}.sp_{i-1,1}.$$
$$sp_{i,1} = d_{i,1}.\overline{sp}_{i-1,0}.$$
$$sp_{i,0} = d_{i,0}.\overline{sp}_{i-1,0}.$$
$$c2_{i+1} = \overline{d}_{i,1}.d_{i,0}.\overline{sp}_{i-1,1}.sp_{i-1,0}.$$

$w_j(l+1) = w_j(l) + \Delta w_j(l)$, where the update term $\Delta w_j(l)$ takes the following forms: (i) $\mu\,\mathrm{sign}(x(l-j))\,\mathrm{sign}(e(l)) \equiv \pm\mu$ (sign-sign algorithm), (ii) $\mu x(l-j))\,\mathrm{sign}(e(l)) \equiv \pm\mu x(l-j)$ (sign algorithm), and (iii) $\mu\,\mathrm{sign}(x(l-j))e(l) \equiv \pm\mu e(l)$ (signed regressor algorithm). For the sign and the signed regressor algorithms, a standard practice is to choose μ to be a power of 2 from within its permissible range, so that computation of $\Delta w_j(l)$ does not require any explicit multiplication by μ. Since $x(l)$, $d(l)$ and thus, $e(l)$ are usually available in 2's complement forms, the update term $\Delta w_j(l)$ in these cases also will be given in 2's complement format. For the sign-sign algorithm, however, since μ is fixed and known a priori, one can represent it either in 2's complement or in canonic SPT form offline. For the current treatment, we assume μ to be given in 2's complement, meaning $\Delta w_j(l)$ for all the three versions of the sign-LMS algorithm will have a 2's complement based representation as $\Delta w_j(l) = \sum_{i=0}^{N-1} \Delta w_{i,j}(l) 2^{-(N-1-i)}$, with $\Delta w_{i,j}(l) \in \{1, 0\}$ denoting the i-th bit of $\Delta w_j(l)$. The weight update block can then be implemented by feeding the bit serial adder of Fig. 2 with the inputs $w_{i,j}(l)$ and $\Delta w_{i,j}(l)$ serially over $i = 0, 1, \ldots N-1$, producing the sequence $w_{i,j}(l+1)$ at the output with a latency of one digit cycle.

Unlike the weight update operations, the filtering part in a sign LMS algorithm, however, requires computation of products like $w_j(l)x(l-j), j = 0, 1, \ldots, p-1$. Assume that each $x(l-j)$ is given by a N bit word in 2's complement form as, $x(l-j) : x_{N-1,l-j}x_{N-2,l-j}\cdots x_{1,l-j}x_{0,l-j}$. Then, from the canonic SPT nature of $w_j(l)$, the product $w_j(l)x(l-j)$ would involve just a few shifts and additions of the word $x_{N-1,l-j}x_{N-2,l-j}\cdots x_{1,l-j}x_{0,l-j}$, with the amount of shift determined by the position of non-zero SPT terms in $w_j(l)$. However, as the SPT expression of $w_j(l)$ changes from index to index, the shift values also change and it is not possible to design circuits that can implement such time varying shift operations. Instead, a better approach would be to consider digit pairs for each filter weight like $\{w_{0,j}(l), w_{1,j}(l)\}, \{w_{2,j}(l), w_{3,j}(l)\}$ etc. and observing that at the most one digit in each digit pair can take non-zero values, develop circuits to implement the action of each digit pair on the data word. For this, we propose a bit serial multiplier below which multiplies a canonic SPT number with a 2's complement number, generating the output in 2's complement form.

A Bit Serial Architecture for Multiplying 2's Complement and Canonic SPT Numbers: Assume that at the l-th word cycle, one input to the system is a N bit 2's complement word B, whose bits b_0 (LSB), $b_1, \ldots, b_{N-1}$(MSB) enter the system serially at $Nl+0, 1, \ldots, N-1$ bit cycles respectively (for convenience, we have dropped the index l from the bits $b_k, k = 0, 1, \ldots, N-1$). In the same word cycle, the other input fed serially to the multiplier is a N digit canonic SPT number $A : a'_{N-1}\cdots a'_1 a'_0$, with the i-th digit a'_i, $i = 0, 1, \ldots, N-1$ represented by a 2 bit 2's complement word $s_i a_i$ $(s_i, a_i \in \{0, 1\})$ as discussed earlier. The two bits s_i, a_i enter the system together at the $Nl+i$-th bit cycle. Also note that since $a'_i \in \{1, -1, 0\}$ (equivalently, $s_i a_i \in \{01, 11, 00\}$), the bit a_i provides the magnitude of the digit a'_i while s_i provides its sign. We take N to be even, i.e., $N = 2K$ for some integer K and form the bit pairs: $\{a_1, a_0\}, \{a_3, a_2\}, \ldots, \{a_{2K-1}a_{2K-2}\}$. It is easily seen that in each bit pair, at the most one bit can be binary 1 and other bit(s) is binary 0. The multiplication of B by A then proceeds as follows:

Assume that the N bit partial product of B by the word $a_{2i-1}a_{2i-2}\cdots a_1 a_0$ has been worked out for some integer $i, 0 \le i \le K-1$ $(a_{-1} = a_{-2} = 0)$ and is given by $\mathbf{pp}_i : pp_i^{N-1}\cdots pp_i^1 pp_i^0$ $(\mathbf{pp}_0 : 00\cdots 0)$. At the next, i.e., $(i+1)$-th step, we use the Horner's rule [2] to multiply B by the 2 bit word $a_{2i+1}a_{2i}$, taking into account the signs of the corresponding digits a'_{2i+1}, a'_{2i} and generating the incremental product $\mathbf{\Delta}_{i+1} : \Delta_{i+1}^{N-1}\cdots\Delta_{i+1}^1\Delta_{i+1}^0$, which is then added with $\mathbf{pp}_i$ with proper alignment. Now, multiplication of B by $a_{2i+1}a_{2i}$ involves evaluation of the two words, $Ba_{2i} : b_{N-1}\cdot a_{2i}\ldots b_1\cdot a_{2i}b_0\cdot a_{2i}$ and $Ba_{2i+1} : b_{N-1}\cdot a_{2i+1}\ldots b_1\cdot a_{2i+1}b_0\cdot a_{2i+1}$ ('$\cdot$' indicates logical AND operation). First consider the case when either both the digits a'_{2i+1}, a'_{2i} are zero or one of them is a 1 (the other one has to be zero from the canonic SPT property). In both cases, $s_{2i+1} = s_{2i} = 0$ and no negation of either Ba_{2i} or Ba_{2i+1} is required. The aforementioned incremental product $\mathbf{\Delta}_{i+1}$ is then computed as

$$\Delta_{i+1}^j = b_j \cdot a_{2i+1} + b_{j+1} \cdot a_{2i}, \quad j = 0, 1, \ldots, N-2 \quad (5)$$

(the LSB $b_0 \cdot a_{2i}$ is neglected to retain the N most significant bits of $\mathbf{\Delta}_{i+1}$), while for $j = N-1$, we carry out the sign extension $b_{N-1} \cdot a_{2i} \rightarrow b_{N-1} \cdot a_{2i}$ to have

$$\Delta_{i+1}^{N-1} = b_{N-1} \cdot a_{2i+1} + b_{N-1} \cdot a_{2i}. \tag{6}$$

Note that since at least one of the two bits a_{2i+1}, a_{2i} is guaranteed to be zero, simple AND-OR logic rather than a full fledged full adder will be enough to generate Δ_{i+1}^{j}, i.e., the '+' operation occurring in the computation of Δ_{i+1}^{j} will be a simple logical OR operation.

Next assume that one of the two digits a'_{2i+1}, a'_{2i} is a -1. This means that either Ba_{2i+1} or Ba_{2i} will have to be 2's complemented depending on whether s_{2i+1} or s_{2i} is 1, which actually means that we need to have provisions for 2's complementing both Ba_{2i+1} and Ba_{2i}. However, noting that one of the two words, Ba_{2i+1} or Ba_{2i} will be an all zero word, this computation can be simplified as explained below.

First assume that $s_{2i} = 1$, meaning $a_{2i} = 1, s_{2i+1} = a_{2i+1} = 0$. We then first compute the 2's complement of Ba_{2i} as $\overline{Ba_{2i}} + 1$ where $\overline{Ba_{2i}} : \overline{b_{N-1} \cdot a_{2i} \ldots b_1 \cdot a_{2i} b_0 \cdot a_{2i}}$ (i.e., the 1's complement of Ba_{2i}), neglect the LSB and extend the sign bit. If $b_0 = 1$, this leads to $\overline{B'a_{2i}}$, i.e., 1's complement of the sign extended word $B'a_{2i} : b_{N-1} \cdot a_{2i} b_{N-1} \cdot a_{2i} \cdots b_2 \cdot a_{2i} b_1 \cdot a_{2i}$, meaning the incremental product will be given as $\mathbf{\Delta}_{i+1} = \overline{B'a_{2i}} + Ba_{2i+1} \equiv \overline{B'a_{2i} + Ba_{2i+1}}$, i.e., 1's complement of $B'a_{2i} + Ba_{2i+1}$ (since Ba_{2i+1} is an all zero word). For $b_0 = 0$, 2's complementation of Ba_{2i} followed by neglecting of LSB and sign extension will similarly lead to $\overline{B'a_{2i}} + 1$, i.e., 2's complement of $B'a_{2i}$. In this case, we will have $\mathbf{\Delta}_{i+1} = \overline{B'a_{2i} + Ba_{2i+1}} + 1$, i.e., 2's complement of $B'a_{2i} + Ba_{2i+1}$. Finally, for the case of $s_{2i+1} = 1$, which also means $a_{2i+1} = 1, s_{2i} = a_{2i} = 0$, it is easy to check that $\mathbf{\Delta}_{i+1}$ will again be given by the 2's complement of $B'a_{2i} + Ba_{2i+1}$. Summarizing, for all the cases, namely, $s_{2i+1}s_{2i} = 00, 01, 10$, we evaluate $\mathbf{\Delta}_{i+1}$ as per (5) and (6). However, for $s_{2i+1}s_{2i} = 01$ and 10, the $\mathbf{\Delta}_{i+1}$ thus obtained will be an intermediate one, which is to be 2's complemented if either $s_{2i+1} = 1$, or $s_{2i} = 1$ and $b_0 = 0$, and 1's complemented if $s_{2i} = 1$ and $b_0 = 1$.

Once the computation of $\mathbf{\Delta}_{i+1}$ is over, it has to be added to $\mathbf{pp}_i$ with proper alignment. For this, we extend the sign bit of $\mathbf{pp}_i$ as $pp_i^{N-1} \rightarrow pp_i^{N-1} \rightarrow pp_i^{N-1}$ and discard its two LSBs pp_i^1 and pp_i^0, resulting in a N bit sign extended word $\mathbf{pp}'_i : pp_i^{N-1}, pp_i^{N-1}, pp_i^{N-1} \cdots pp_i^3 pp_i^2$. The $(i+1)$-th partial product is then evaluated as $\mathbf{pp}_{i+1} = \mathbf{pp}'_i + \mathbf{\Delta}_{i+1}$. This process would, however, require certain corrections as we had ignored the term $a_{2i} \cdot b_0$ in the computation of $\mathbf{\Delta}_{i+1}$. First consider the case when $s_{2i} = 0$. In this case, the conventional, Horner's rule based multiplication would have considered the sum $pp_i^1 + a_{2i} \cdot b_0$, which is ignored here and which generates the carry $pp_i^1 \cdot a_{2i} \cdot b_0$ that can ripple through the subsequent stages of addition up to the MSB and thus should not be neglected. Next, for the case when $s_{2i} = 1$ and thus $a_{2i} = 1$, 2's complementation of Ba_{2i} results in a LSB of 1 and 0 respectively for $b_0 = 1$ and $b_0 = 0$. It is then easy to verify that the expression $pp_i^1 \cdot a_{2i} \cdot b_0$ will also provide the carry generated by the addition of the above LSB with pp_i^1 and thus should be taken as the initial carry in the computation of $\mathbf{pp}_{i+1}$. Finally, it is easy to verify that the output will be available from the $(Nl + N - 1)$-th cycle onwards (with LSB first).

The proposed multiplier is shown in Fig. 3 for $N = 4$. Using switches, the four bits a_0, a_1, a_2 and a_3 are held in delay loops for four cycles, starting respectively at the following indices: $4l + 0, 1, 2, 3$. Same is done separately to the four sign bits, s_0, s_1, s_2 and s_3. For sign extension, we feed the bit to be extended to a delay and move a switch from input of the delay to the output in the next clock cycle. Also, for convenience, we follow the modular notation for timing instances in Fig. 3, i.e., $4l + 0 \equiv 4l + 4, 4l + 1 \equiv 4l + 5$ and $4l + 2 \equiv 4l + 6$. The bits $\Delta_1^j, j = 0, 1, 2, 3$ are generated serially at $4l + 1, 2, 3, 4(\equiv 0)$ respectively, by first implementing (5) and (6) (for $i = 0$) using an AND-OR logic combination as shown in Fig. 3. For 1's complementation of the intermediate $\mathbf{\Delta}_1$ thus generated, the corresponding bits Δ_1^j are fed to an Ex-OR gate with $s_0 + s_1$ as the other input, while for 2's complementation, a 1 is added to the Ex-OR output at $4l + 1$ (i.e., to the LSB), generated by the logic $\overline{b_0 \cdot a_0} \cdot (s_0 + s_1)$. This produces the partial product bits $pp_1^j, j = 0, 1, 2, 3$ at $4l + 1, 2, 3, 0$ respectively. For generating the bits $\Delta_2^j, j = 0, 1, 2, 3$, identical steps are followed except that these bits are generated respectively at the cycles $4l + 3, 0, 1, 2$, the control input of the Ex-OR gate is given by $s_2 + s_3$ and the logic $\overline{b_0 \cdot a_2} \cdot (s_2 + s_3)$ is used to add a 1 to the Ex-OR output (at $4l + 3$). The final result $y_j, j = 0, 1, 2, 3$ is obtained at cycles $4l + 3, 0, 1, 2$ (LSB first), by adding $\mathbf{pp}_1$ with $\mathbf{\Delta}_2$ after discarding the bits pp_1^0, pp_1^1, extending the bit pp_1^3 twice using two delays and a switch, and applying the correction term $pp_1^1 \cdot a_2 \cdot b_0$ as the initial carry to the full adder at cycle $4l + 3$.

Implementation Issues: As seen above, the output is available from the cycle $4l + 3$ onwards. In general, if two N bit numbers are multiplied, the output will be available bit serially over the cycles, $lN + N - 1, 0, 1, \ldots, N - 2$, i.e., there is a latency of $(N - 1)$ cycles. Each output will actually provide the result of the multiplication $w_j(l)x(l - j), j = 0, 1, \ldots, p - 1$. These are to be added and then subtracted from the desired response $d(l)$, all bit serially to yield $e(l)$. However, to maintain correctness, the bit sequence of $d(l)$ is to be delayed by $(N - 1)$ cycles. Also note that the sign bit of $e(l)$, needed to compute the update terms in (2) and (3) will be available only at the cycle $(lN + 2N - 2)$, meaning the bits of $w_j(l + 1)$ will not be available before cycle $(lN + 2N - 2)$. The next, i.e., $(l + 1)$-th data bits, however, enter the system from cycle $(lN + N)$ onwards, meaning the weight bits for $w_j(l + 1)$ will actually be required from cycle $(lN + N)$ only. To overcome this problem, we take recourse to the delayed mode of adaptation [13]. In particular, we delay the update term in the weight update equations by one data cycle, resulting in the following update equations: $\mathbf{w}(l + 1) = \mathbf{w}(l) + \mu \operatorname{sign}(\mathbf{x}(l - 1)) \operatorname{sign}(e(l - 1))$ (sign-sign algorithm) and $\mathbf{w}(l + 1) = \mathbf{w}(l) + \mu \mathbf{x}(l - 1) \operatorname{sign}(e(l - 1))$ (sign algorithm). This means the sign bit of $e(l - 1)$, available at cycle $lN + N - 2$ will be fed back to the bit serial adder, either to determine the sign of μ (in the case of the sign-sign algorithm, which will also require the available/stored sign bit of $x(l - 1 - j)$), or to determine the sign of $\mu x(l - 1 - j)$ (in the case of the sign algorithm). There is a gap of two cycles between the time when the sign bit of $e(l - 1)$ becomes available (i.e., $lN + N - 2$) and the time (i.e., $lN + N$) when the updated weight bits are generated. Of these two cycles, one is consumed by the bit serial adder which has a

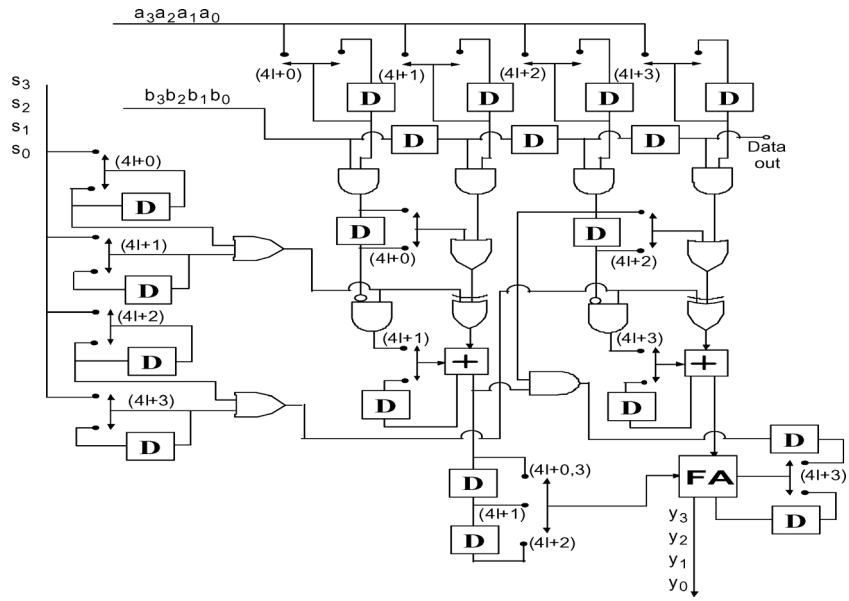

Fig. 3. Proposed bit serial multiplier (for four bit multiplicands).

latency of one cycle while for the other, we employ one delay (the additional delay in this case can be used to pipeline the two stages, namely, blocks 1 and 2 in the adder shown in Fig. 2, after necessary retiming of the relevant edges).

In the case of the weight update equation (4), while computation of the update term does not require the sign bit of $e(l)$, it still requires evaluation of the product $\mu e(l)$. As stated earlier, μ in this case (also in the case of the sign algorithm) is taken as a power of 2, say, 2^{-r}, meaning computation of $\mu e(l)$ will require extension of the sign bit of $e(l)$ r times and discarding of the lower r bits of $e(l)$. From this and also from the fact that the proposed bit serial adder has a latency of one cycle, this implies that in this case too, the updated weight bits for $w_j(l+1)$ will not be available at the beginning of the $(l+1)$-th data word, i.e., at cycle $lN+N$. As a result, in this case as well, we delay the update term by one data cycle, leading to $\mathbf{w}(l+1) = \mathbf{w}(l) + \mu \operatorname{sign}(\mathbf{x}(l-1))e(l-1)$. It may be noted that in the above implementation, we have considered the minimum unavoidable delay of one data cycle in the delayed mode of coefficient adaptation. However, by increasing the delay further [13] and noting that one word cycle delay actually amounts to N bit cycle delays, it is possible to pipeline the various operations in the weight update loop (e.g., addition of the multiplier outputs to form $y(n)$, subtraction of $y(n)$ from $d(n)$ to generate $e(n)$ etc.) to reduce the critical path.

The proposed bit serial architecture for a 4 tap, sign-sign based adaptive filter with wordlength $N=4$ is shown in Fig. 4. It generates the filter output by evaluating the two partial sums $\sum_{j=0}^{1} w_j(l)x(l-j)$ and $\sum_{j=2}^{3} w_j(l)x(l-j)$ separately, and then adding them to yield $y(l)$. This requires four multipliers acting in parallel on $x(l), x(l-1), x(l-2)$ and $x(l-3)$. The multiplier acting on the input, say, $x(l-j), j=0,1,2,3$ delays the corresponding bits by three bit cycles internally as can be seen from Fig. 3. These are then delayed further by one more cycle to generate the input bits for the next multiplier, i.e., the one with input $x(l-j-1)$ at the l-th index (note that one word cycle $\equiv 4$ bit cycles). The other input to the multiplier is given by the SPT digit for the respective filter weight (though shown by one line, it actually represents a 2 bit bus, with the MSB for the sign and the LSB for the magnitude of the SPT digit). The bits of $y(l)$ and also of $e(l)$ obtained by 2's complementing $y(l)$ and adding with the 3 bit cycle delayed version of $d(l)$, are generated at $4l+3, 0, 1, 2$. For the delayed mode of adaptation, the sign bit of $x(l-1)$, available at cycle $4l-1$ $(\equiv 3)$ is to be AND-ed with the sign bit of $e(l-1)$, available at $4l+2$. For this, we feed the 3 bit cycle delayed $x(l)$, along with $e(l)$ to an AND gate, sample the AND output at $4l+2$ and hold it in a delay loop for 4 cycles as a control bit, for determining the sign of the update term, i.e., μ, whose bits u_0, u_1, u_2 and u_3 are fed serially to the system as shown in Fig. 4. For updating any weight, the SPT digits of the weight and the bits of μ after determining its sign[3] are fed to a SPT—2's complent adder of the type shown in Fig. 2, over cycles $4l+2, 3, 0, 1$. Since the adder has a latency of one cycle, we delay the adder output further by one more bit cycle so as to produce the updated weight bits for the $(l+1)$-th data cycle from cycle $4l+0(\equiv 4)$ onwards. Also, as the addition starts at cycle $4l+2$, the current weight bits from cycle $4l+0$ are fed back to the adder after delaying by 2 bit cycles.

VLSI Realization: For this, we first considered the proposed canonic SPT—2's complement adder, shown in Fig. 2 and used in Fig. 4, separately and evaluated it against the conventional approach of 2's complement addition first followed by canonic SPT conversion, for which we considered the conversion circuit of [11]. Firstly, by manual inspection, it is seen that the critical path of the proposed adder uses two Ex-OR gates and one AND gate, whereas the same for the circuit of [11] uses three Ex-OR gates. Also, in terms of equivalent 2-input NAND gates, manual inspection revealed that the proposed adder has equivalent gate count of 17 whereas the circuit of [11] has 19.5. In order to have more realistic figures of the gate count and the critical path, both

[3]To control the sign of μ, each bit u_i is first fed to an Ex-OR gate with the other input provided by the above control bit, which is also taken as the initial carry bit c_0 in the addition process. This arrangement is, however, not shown separately in Fig. 4 and is assumed to be part of each adder block.

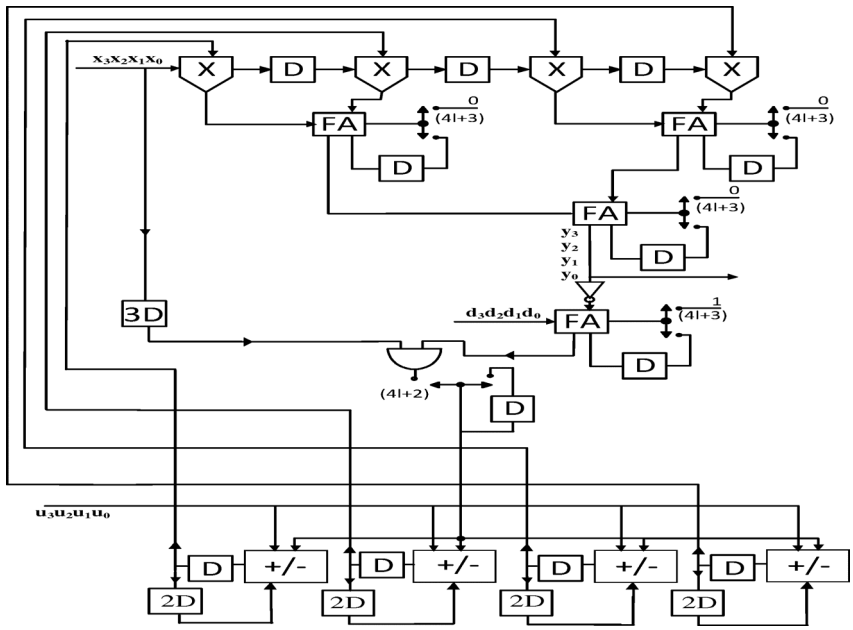

Fig. 4. Proposed Bit-Serial Realization of a 4 Tap Sign-Sign based Adaptive Filter with Wordlength $N = 4$.

TABLE II
ASIC SYNTHESIS SUMMARY FOR (A) THE PROPOSED CANONIC SPT-2'S COMPLEMENT ADDER, AND (B) THE APPROACH OF 2'S COMPLEMENT ADDITION FIRST FOLLOWED BY CANONIC SPT CONVERSION [11]

	Approach A	Approach B
Total equivalent gate count	17.25	20.25
Critical Path	372.1 ps	490 ps

TABLE III
ASIC SYNTHESIS SUMMARY OF THE SIGN-SIGN ALGORITHM: (A) THE PROPOSED BIT SERIAL REALIZATION, AND (B) THE CONVENTIONAL WORD SERIAL REALIZATION

	(A) Bit Serial	(B) Word Serial
Total equivalent gate count	520	3904
Critical Path	3.69 ns	22.22 ns

the techniques were synthesized for ASIC realization using the 65 nm cell library (synthesis tool used: Synopsys Design Compiler, Version 2010.12-SP3) and the synthesis results are shown in Table II. Clearly, the proposed method requires 14.81% less number of gates, while achieving 24% speed-up over the approach B.

Next, we tried to estimate the hardware reduction achieved by the proposed bit serial implementation of the sign-sign based adaptive filter vis-a-vis its conventional word serial version. For this, first the proposed bit serial scheme was synthesized using the above 65 nm cell library for a 8 tap filter, with coefficients represented in 8 digit canonic SPT form, and both the step size μ and the input data represented by 8 bit 2's complement words. Separately, a conventional word serial form of the same 8 tap sign-sign based adaptive filter was synthesized, with the coefficients, the step size and the input represented by 8 bit 2's complement words. Table III provides the synthesis summary for both the realizations. Clearly, for a 8 bit input, the proposed bit serial scheme achieves hardware reduction by 7.5 times when compared against the conventional word serial realization.

IV. DISCUSSIONS AND CONCLUSION

A bit serial realization of adaptive filters belonging to the sign-LMS family is presented. To reduce the complexity of the multiplications that arise in the filtering part, the filter coefficients are represented in the so-called canonic SPT (CSD) form. Care is also taken to update the coefficients in the canonic SPT domain only so that no additional steps to convert the coefficients from 2's complement to canonic SPT format at each time index become necessary. Towards this, we have proposed a bit serial adder that takes as input two numbers in canonic SPT form and produces an output also in canonic SPT, which is later generalized to the case where one of the inputs is changed to the 2's complement form. For both the cases, the canonic SPT property of the input is exploited to reduce the complexity of the adder. For the filtering part, a bit serial multiplier is developed for multiplying the data bits (2's complement) by the weight bits (canonic SPT). However, as the filter weights keep changing from index to index, the positions of the non-zero power of two terms in the canonic SPT expression of each coefficient also change with time and obviously, the corresponding time varying shift operations can not be realized by fixed hardware circuits. The proposed multiplier takes care of this by dividing the coefficient word into non-overlapping groups of two consecutive SPT digits each and then by observing that in each group, at the most one SPT term can be non-zero (i.e., ± 1). The partial product resulting from the multiplication of the 2's complement number by each group can then be realized using simple AND-OR logic rather than a full fledged full adder. Finally, the bit serial processing has an inherent latency that necessitates delayed mode of coefficient adaptation. In this paper, we have considered delay by one data cycle only, which is the minimum unavoidable delay. However, the delay can be increased further

and the resulting N fold increase in the bit cycle delays can be used to pipeline the weight update loop.

REFERENCES

[1] Y. C. Lim, J. B. Evans, and B. Liu, "Decomposition of binary integers into signed power-of-two terms," *IEEE Trans. Circuits Syst.*, vol. 38, no. 6, pp. 667–672, Jun. 1991.

[2] K. K. Parhi, *VLSI Digital Signal Processing Systems—Design and Implementation*. New York: Wiley, 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 Syst. I, Reg. Paper*, vol. 54, no. 6, pp. 1348–1357, Jun. 2007.

[4] S. Y. Park and N. I. Cho, "Design of multiplierless lattice QMF: Structure and algorithm development," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 55, no. 2, pp. 173–177, Feb. 2008.

[5] Z. G. Feng and 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 Syst. I, Reg. Papers*, vol. 55, no. 6, pp. 1536–1545, Jun. 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 Syst. I, Reg. Papers*, 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. Comput.-Aid. Des. Integr. Circuits Syst.*, 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," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 53, no. 6, pp. 1254–1264, Jun. 2007.

[10] K.-H. Chen, C.-N. Chen, and T.-D. Chiueh, "Grouped signed power-of-two algorithms for low-complexity adaptive equalization," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 52, no. 12, pp. 816–820, Dec. 2005.

[11] Y. Wang, L. S. DeBrunner, D. Zhou, and V. E. DeBrunner, "A multiplier structure based on a novel real-time CSD recoding," in *Proc. IEEE ISCAS-2007*, New Orleans, USA, May 27–30, 2007, pp. 3195–3198.

[12] S. Haykin, *Adaptive Filter Theory*. Englewood Cliffs, NJ: Prentice-Hall, 1986.

[13] G. Long, F. Ling, and J. G. Proakis, "The LMS algorithm with delayed coefficient adaptation," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 37, no. 9, pp. 1397–1405, Sep. 1989.

[14] S. Choudhary, P. Mukherjee, and M. Chakraborty, "A SPT treatment to the bit-serial realization of sign-LMS based adaptive filter," in *Proc. IEEE ISCAS 2010*, Paris, France, May 2010, pp. 2678–2681.

[15] S. Choudhary, P. Mukherjee, and M. Chakraborty, "A SPT based low complexity realization of the weight update loop of an adaptive filter," in *Proc. APSIPA ASC 2010*, Singapore, Dec. 2010, pp. 347–349.

[16] S. Choudhary, P. Mukherjee, and M. Chakraborty, "An algorithm for bit-serial addition of SPT numbers for multiplierless realization of adaptive equalizers," in *Proc. APSIPA ASC 2009*, Sapparo, Japan, Oct. 2009, pp. 438–440.

Sunav Choudhary was born in 1988 in Burdwan, India. He received the B.Tech. degree in electronics and electrical communication from the Indian Institute of Technology, Kharagpur, India, in 2010. He is currently pursuing the Ph.D. degree in the Communication Sciences Institute, University of Southern California, Los Angeles, and is a recipient of the Annenberg Graduate Fellowship.

His present research interests are in the field of sparse signal approximation and its applications to underwater acoustic communications.

Pritam Mukherjee received the B.Tech. degree from the Indian Institute of Technology, Kharagpur in 2010, with major in electronics and electrical communication engineering and minor in computer science and engineering. Presently, he is a graduate student in the Department of Electrical and Computer Engineering, University of Maryland, College Park, pursuing the Ph.D. degrees in communications and signal processing.

His current research interests are in Information Theory and Coding, especially in the secrecy context.

Mrityunjoy Chakraborty (M'94–SM'99) received the B.Eng. degree from Jadavpur University, Calcutta, in 1983, the Master of Technology degree from the Indian Institute of Technology (IIT), Kanpur, India, in 1985, and the Ph.D. degree from the IIT, Delhi, India, in 1994.

He joined IIT, Kharagpur as a faculty member in 1994, where he currently holds the position of a professor in Electronics and Electrical Communication Engineering. His teaching and research interests are in Digital and Adaptive Signal Processing, VLSI Signal Processing, Linear Algebra and DSP applications in Wireless Communications. In these areas, he has supervised several graduate theses, carried out independent research and has several well cited publications.

Prof. Chakraborty has been an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: REGULAR PAPERS (2004–2007, 2010–2012) and TRANSACTIONS ON CIRCUITS AND SYSTEMS—II: EXPRESS BRIEFS (2008–2009), apart from being an elected member of the DSP TC of the IEEE Circuits and Systems Society, a guest editor of the EURASIP JASP (special issue) and a TPC member of ICC (2007–2011) and Globecom (2008–2011). He is a co-founder of the Asia Pacific Signal and Information Processing Association (APSIPA) and currently is also the chair of the APSIPA TC on Signal and Information Processing Theory and Methods (SIPTM). He is the general chair and also TPC chair of the National Conference on Communications (NCC)-2012. He is a fellow of the Indian National Academy of Engineering (INAE).

Shakti Shankar Rath received the B.Tech. degree in electronics and electrical communication engineering from the Indian Institute of Technology (IIT), Kharagpur, India, in 2000, where he is currently pursuing the Ph.D. degree as an industry sponsored candidate.

Since 2000, he has been working at Texas Instruments, Bangalore, India, in the field of Analog IC Design where he presently holds the position of a senior technical leader. His current research interests are in Circuits and Systems for Digital and Adaptive Signal Processing.