Novel Design Algorithm for Low Complexity Programmable FIR Filters Based on Extended Double Base Number System

Jiajia Chen, *Member, IEEE*, Chip-Hong Chang, *Senior Member, IEEE*, Feng Feng, Weiao Ding and Jiatao Ding

***Abstract*—Coefficient multipliers are the stumbling blocks in programmable finite impulse response (FIR) digital filters. As the filter coefficients change either dynamically or periodically, the search for common subexpressions for multiplierless implementation needs to be performed over the entire gamut of integers of the desired precision, and the amount of shifts associated with each identified common subexpression needs to be memorized. The complexity of a quality search is thus beyond the existing design algorithms based on conventional binary and signed digit representations. This paper presents a new design paradigm for the programmable FIR filters by exploiting the Extended Double Base Number Systems (EDBNS). Due to its sparsity and innate abstraction of the sum of binary shifted partial products, the sharing of adders in the time-multiplexed multiple constant multiplication block of the programmable FIR filters can be maximized by a direct mapping from the quasi-minimum EDBNS. The multiplexing cost can be further reduced by merging double base terms. Logic synthesis results on more than one hundred programmable filters with filter taps ranging from 10 to 100 and coefficient word lengths of 8, 12 and 16 bits show that the average logic complexity and critical path delay of the programmable FIR filters designed by our proposed algorithm have been reduced by up to 47.81% and 14.32%, respectively over the existing design methods.**

***Index Terms* — FIR Filter, Double Based Number System, Programmable Filter, Digital Signal Processing.**

I. Introduction

FINITE Impulse Response (FIR) filters offer many advantages, such as easily attainable linear phase response, computational efficiency in multi-rate applications and desirable numerical property for finite precision and fractional arithmetic [1]-[4]. Adaptive filters, in particular, are inevitable in many important applications in communications, image processing, computer vision, data acquisition and control [5]–[9]. With any adaptive filter, there is a requirement for a programmable filter, which is a primary reason behind the increasing dominance of digital instead of analog system implementations. In applications such as multi-rate decimation [6], discrete cosine transform [7], [8], channelization [9], [10], high efficiency video coding (HEVC) [11], wide bandwidth photonic filter [12] and high-rate communication [13], the filter coefficients need to be run-time reconfigurable by the error feedback signal or adaptable to varying filtering specifications in real time. To reduce the throughput rate from N clock cycles to one clock cycle, the weighted sum of the past and present input samples with changing coefficient values in an N-tap adaptive filter is implemented on field programmable gate array (FPGA) with N multiply-and-accumulate (MAC) units [14]-[17]. In [15], MAC units are used in the FIR filter architecture for the design of discrete wavelet transforms. In [16], a dot-product unit is designed using the multiplier core on FPGA and a mechanism to reallocate the partial products for better resource utilization is proposed. As multipliers are much slower and consume substantially more area than adders, programmable filter raises the cost and latency of the system, as exemplified by the dominating complexity of coefficient computation in channel equalization [10] and the HEVC decoding time contributed by the adaptive loop filter [11].

To avoid the time consuming and area intensive multipliers, they are replaced by simpler arithmetic operators in the form of shift-and-add network based on the transposed direct form implementation. The shifted partial products of the filter coefficients need to be recalculated before they are delayed and accumulated by the structural adders. Hence, the shifters are not fixed but vary with the changing partial products. The partial products also need to be either computed on demand or pre-computed and pre-stored for selection, which results in a Time Multiplexed Multiple Constant Multiplication (TM-MCM) block [8], [18]. Design heuristics [19]–[23] have been proposed to detect and eliminate the common subexpressions in the fixed filter coefficient set to reduce the implementation complexity of the multiple constant multiplication block. Unfortunately, the search for common subexpressions in a coefficient set is itself a complex process which cannot be easily implemented in hardware. This means that the shifted partial products cannot be dynamically modified in real time for a time varying coefficient set without some form of precomputation and storage. Design algorithms for Common Subexpression Elimination (CSE) [20] help, however, in

Manuscript received May 2, 2014; revised July 21, 2014; accepted August 5, 2014.

The authors Jiajia Chen, Feng Feng, Weiao Ding and Jiatao Ding are with Singapore University of Technology and Design, Singapore. (e-mail: jiajia_chen@sutd.edu.sg)

The author Chip-Hong Chang is with the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore. (e-mail: echchang@ntu.edu.sg)

reducing the number of partial products to be pre-computed and hence the storage requirement and implementation complexity of the shifters and adders in the TM-MCM block of the programmable filter.

CSE algorithms [5], [9], [24]-[27] were developed to design programmable FIR filters with the coefficients represented in binary or Canonical Signed Digit (CSD) [28] representations. Algorithms [5] and [9] used the traditional pattern matching techniques historically adopted for high-level synthesis [29]. Due to the unspecified time-varying coefficients of programmable filters, the computational complexity is exceedingly high. The enormous search space renders these heuristics ineffective and inefficient. To overcome this problem, the computational sharing multiplier architecture [5] arbitrarily identifies all 4-bit binary expressions of odd integers from 0001 to 1111 as common subexpressions. Similarly, all 3-bit binary expressions from 001 to 111 are generated as common subexpressions in [9] and [24], and multiplexers are used to select the subexpressions to form the partial products to be summed into the updated coefficient values. Frequency response masking is used to develop the reconfigurable multi-mode filter bank in [9] whereas Constant Shifts Method and Programmable Shifts Method are proposed in [24]. The latter also provides the flexibility of changing the word length of the filter coefficients dynamically. These methods skip the common subexpression search, but the straightforward and simplistic approach of selecting common subexpressions has an inherently high opportunity cost – many reusable partial products were omitted.

Owing to the huge search space over the complete gamut of integer values of a given precision, the intricacy of common subexpression sharings within and across multiple sets of coefficients in a TM-MCM block is beyond the capability of existing CSE algorithms and fixed-coefficient filter design methodologies even for moderate coefficient word length and filter taps. As the shifters are no longer fixed, more succinct number representation than CSD and binary are explored. DBNS and multidimensional logarithmic number system (MDLNS) are considered for the design of DSP operators such as constant multiplier [30]-[32]. In [30], logarithmic and double base number representations are used to synthesize inexact fractional filter coefficients for time-invariant filters. To minimize the additions of each coefficient multiplier, multiple-radix DBNS representation is used by limiting the maximum power of the second base number in [31]. In this paper, the double base number representation is uniquely exploited to maximize the subexpression sharings for all filter coefficients of a given word length in a more general time-varying filter implementation problem that has no leverage of the fixed quantized coefficients at design time. This work is an extension of our antecedent work [33], which is the first ever attempt to harness the sparseness of the canonic DBNS representation for the complexity reduction of TM-MCM block. In this paper, we have extended the Double Base Number System (DBNS) [34] to encapsulate the binary shifts in tandem with several most frequently encountered common subexpressions. A new formulation of the common subexpression search problem as a quasi-minimized extended DBNS (EDBNS) generation problem is proposed, which has led to considerable reduction in the number of distinct partial products for a given word length of programmable coefficients. With this number system, an efficient architecture for the implementation of TM-MCM is derived as shown in Fig. 1. It consists of a power-of-*b* generator (POBG), *N* blocks of power-of-*b* selector (POBS) and *N* blocks of double base coefficient generator (DBCG), where *b* is the second base number in EDBNS and *N* is the number of taps. In our design, those unique partial product terms can be generated incrementally with one adder each. The sizes of the multiplexers in the programmable units and the lookup tables for the selector logic are further minimized by exploiting the unique properties of EDBNS in the Exponential Diophantine Equation.

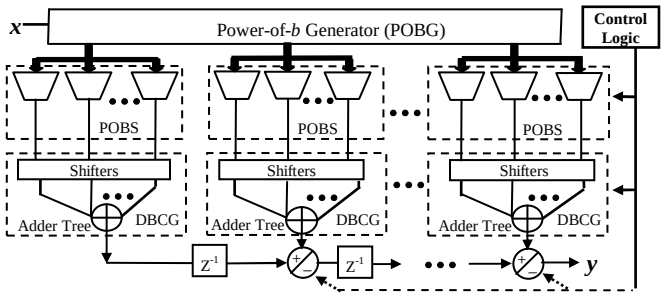

Fig. 1. Transposed form FIR filter with programmable coefficients.

II. Extended Double Base Number System

The output sequence of an *N*-tap finite impulse response (FIR) digital filter can be computed by the following discrete time convolution.

$$y[n] = h[n] * x[n] = \sum_{i=0}^{N-1} h[i]x[n-i] \tag{1}$$

where $x[n]$ and $y[n]$ are the n-th time domain input and output data samples. The samples $h[i]$, $i = 0, 1, \ldots, N-1$, also known as the filter coefficients, are the impulse response of the filter transfer function $H(z)$, i.e., $H(z) = \sum_{i=0}^{N-1} h[i]z^{-i}$.

For fixed point implementation, each coefficient is quantized into a finite precision integer. The quantized coefficient, denoted by c, can be written as a sum of Power-of-Two (POT) terms as follows:

$$c = \sum_{i=0}^{w-1} b_i \cdot 2^i \tag{2}$$

where $b_i \in \{0,1\}$ for binary representation and $b_i \in \{0,1,-1\}$ for signed digit representation. w is the word length required by the representation to express all the integer coefficients of the filter.

The discrete convolution in (1) can be implemented using a network of hardwired shifters and adders. The number of adders required is determined by the number of nonzero POT terms in the binary or signed digit representation of all coefficients of the filter. It can be greatly reduced by sharing common POT terms (also known as common subexpressions) [20]. The search space for the design is feasible if the filter coefficients are fixed. If the coefficient values are allowed to change, such as those in an adaptive filter, then all the 2^w w-bit integers need to be represented. The number of nonzero POT terms in (2) is

binomially distributed and the frequency of occurrences of r nonzero POT terms in all integers of w-bit binary representation is given by C_r^w . On average, $\frac{1}{2^w}\sum_{r=0}^{w} r \cdot C_r^w$ nonzero POT terms are required to represent a w-bit integer in binary. For $w = 8$, this number is 4. The number of nonzero POT terms can be saved by 33% on average if canonic signed digit (CSD) representation [28] is used, which means that the average number of signed POT terms for an 8-bit integer can be reduced to 2.67.

The sums of two adjacent nonzero POT terms in binary (i.e., bit pattern "11") and CSD (i.e., bit patterns "$10\bar{1}$"), and their negation corresponding to the decimal numbers 3 and −3, appear to be the most frequently occurred common subexpressions in digital filter design. By recursively decomposing c in (2) into the sum of $(2^{\alpha+1}+2^{\alpha}) = 3\times2^{\alpha}$ and a smaller integer of $c - 3\times2^{\alpha}$, and factoring out 3 in each iteration, a sparse Double Base Number System (DBNS) can be obtained. This DBNS is defined in [35] for any integer c as:

$$c = \sum_{\alpha,\beta} d_{\alpha,\beta} 2^{\alpha} 3^{\beta} \tag{3}$$

where $d_{\alpha,\beta} \in \{0,1\}$, and α and β are the exponents of 2 and 3 of the double base (or 2-integer) term, respectively.

For any integer c, the DBNS with the least number of nonzero double base terms is called the canonic DBNS [35]. It should be noted that unlike CSD, the integer representation in canonic DBNS is not unique. To avoid the ambiguity of the definition of canonicity, we will refer to canonic DBNS as the minimum DBNS. Due to the sparsity of the minimum DBNS, any integer within a given range can be generated with fewer additions of double base terms (i.e., 3^{β} shifted by α for $d_{\alpha,\beta} \neq 0$) and it will be shown in Section IV that each distinct double base term can be generated using only one adder.

The subexpression "101" of two adjacent nonzero POT terms in binary and CSD representations and its negation corresponding to the decimal integers 5 and −5 are also frequently encountered. To incorporate this additional common subexpression, we propose to extend the DBNS to include the next prime 5 as a choice for the second base. To differentiate it from the accustomed form of DBNS defined in [34] with 2 and 3 as the only base numbers, we called this form of DBNS the Extended Double Based Number System (EDBNS). Its representation for any positive integer c is defined as:

$$c = \sum_{t=1}^{T} 2^{\alpha_t} b_t^{\beta_t} \tag{4}$$

where $b_t \in \{3, 5\}$, α_t and β_t are respectively the non-negative exponents of 2 and b_t of the t-th nonzero double base term, and T is the total number of nonzero double base terms. Negative coefficient can be expressed in sign-magnitude form with its magnitude expressed in this EDBNS and its sign used to configure its structural adder as adder or subtractor to achieve the same effect as using signed EDBNS with configurable adder/subtractor.

It can be shown by exhaustive enumeration that every positive integer in the range (0, 256) can be expressed in EDBNS with three or less double base terms. The numbers of occurrences of EDBNS representations with one, two and three double-base terms over the entire range are 26, 147 and 82, respectively. On average, only 2.21 double base terms are required to represent any integer in the range [0, 255]. This is 44.75% and 17.23% lower than the binary and CSD representations, respectively.

Similar to the definition of minimum DBNS, the minimum EDBNS representation of an integer c is an EDBNS representation of c with the minimum number of nonzero double base terms T_{min}. The minimum EDBNS can be constructed as an abstraction of the maximum sharings of two adjacent POT terms in binary and CSD representations over the range of w-bit integers representable by the EDBNS.

III. Generation of quasi-Minimum EDBNS

A negative coefficient can be replaced by a positive coefficient by subtracting the positive coefficient instead of adding the negative coefficient at the structural adder block. Let w be the word length of the unsigned binary representation of the largest coefficient magnitude of a programmable FIR filter. The CSE problem for the TM-MCM block can then be recast into the problem of searching for a minimum number of distinct power-of-b integers, i.e., b^{β}, where $b \in \{3, 5\}$ and β is a positive integer, to generate the minimum or near minimum EDBNS representations for all the integers in the range $(0, 2^w)$.

The highest number of double base terms T_{max} for a valid EDBNS representation of any w-bit integer c is $T_{max} = c$, which is the case when the powers of both bases of all the terms are zero. If only the power of the second base b is always 0, then (4) reduces to (2), which gives a tighter upper bound of T_{max}. If there is no restriction imposed in the number of terms T, there are many possible ways to express a w-bit positive integer in DBNS [31] or EDBNS. For example, it is reported in [34] that the integer 127 has a total of 783 different DBNS representations. The minimum EDBNS representation of each integer in the range $(0, 2^w)$ has different number of terms T. Let $T_{min}(w)$ be the maximum number of double base terms among the minimum EDBNS representations of all the w-bit integers. To minimize the number of adders required to add up the double base terms, $T = T_{min}(w)$.

Let $F_{min}(T, w)$ be the smallest set of positive power-of-b ($b \in \{3, 5\}$) integers that appear in any terms of the EDBNS representations of w-bit integers with T or less double base terms. To obtain $F_{\min}$, all possible power-of-b sets that can be used to represent all the w-bit integers in EDBNS with T or less terms are sought. Among them, the set that has the minimum number of power-of-b integers is selected as $F_{\min}$. If there are two or more sets that have the same minimum number of power-of-b integers, the one with more power-of-3 integers is selected as $F_{\min}$ to avoid complicating the search and comparison. Table I shows $T_{min}(w)$ and $F_{min}(T_{min}, w)$ obtained by this method for $w = 8$, 12 and 16.

TABLE I. F_{min} with the $T = T_{min}$ for 8-bit, 12-bit and 16-bit coefficients.

w		8	12	16
T_{min}		3	4	4
F_{min}	$b = 3$	3,9,27	3,9,27,81	3,9,27,81,243,729,2187,6561
	$b = 5$	5	5	5,25,125

Obviously, with $T = T_{min}(w)$, the number of adders required to add up the double base terms of all w-bit coefficients can be minimized by the minimum EDBNS. However, this may also result in a larger set of distinct power-of-b integers to guarantee that the minimum EDBNS exists for all w-bit integers. A good trade-off is made by the following proposition.

Proposition 1: The minimum EDBNS representations of all w-bit integers are first generated to obtain $F_{min}(T_{min}, w)$. Then, T is incremented to $T_{min}(w) + 1$ and the quasi-minimum EDBNS representations of T terms are generated. If $|F_{min}(T, w)| < |F_{min}(T_{min}, w)|$, where $|F|$ denotes the cardinality of F, then T is further incremented and the process continues until $|F_{min}(T–1, w)| = |F_{min}(T, w)|$.

The cardinality of F_{min} may decrease when T increases above T_{min}. As the cardinality of F_{min} stops to shrink with increasing T, the reduction in the quantity and bit width of adders required to realize the distinct power-of-b terms in F_{min} ceases. Further increment of T will only increase the redundancies of one or more EDBNS representations for all w-bit integers. Hence, the search for EDBNS representations can stop as it will not lead to more efficient hardware implementation. The pseudo code for the EDBNS search algorithm is presented in Fig. 2.

```
EDBNS(w) {
    C = set of all w-bit coefficients;
    exp3_max = ⌊w/log2 3⌋  // max exponent of power-of-3 integer < 2^w
    exp5_max = ⌊w/log2 5⌋  // max exponent of power-of-5 integer < 2^w
    T = Tmin(w); // min. no. double base terms for w-bit integers
    F_current = Fmin(T, w); //  current min. set of power-of-b integers
    F_next = ∅; // next F_min
    S = ∅; // initialize set of EDBNS representations found
    EDBNS_array = ∅; // Initialize storage for EDBNS representations of C
    P = insert(P, 0, 0, 1); // Initial entry for storage of double base terms
    while (F_next ≠ F_current) {
        for (i from 1 to w–1) {
                for (j from 1 to exp3_max)
                    while (2^i 3^j < 2^w) P = insert(P, i, j, 2^i 3^j);
                for (j from 1 to exp5_max)
                    while (2^i 5^j < 2^w) P = insert(P, i, j, 2^i 5^j); }
        while (C is not empty) {
            for (i from 1 to T) {
                S = Gen_EDBNS(P, i);
                I = EDBNS2Int(S) ⋂ C;
                if (I is not empty) {
                    EDBNS_array = insert(EDBNS_array, I);
                    C = C – I ; } } }
        F_current = F_next;
        T = T+1;
        F_next = Fmin(T, w); }
    return EDBNS_array;
}
```

Fig. 2. Proposed search algorithm for quasi-minimum EDBNS.

The function **EDBNS**(w) returns an array of EDBNS representations for all the w-bit integers in C. The function **Tmin**(w) returns $T_{min}(w)$ for w-bit integers. The function **Fmin(***T, w*) returns the smallest set of power-of-b integers that can be used to represent all w-bit integers with T or less terms in EDBNS. The function **insert**(P, i, j, t) appends the exponents, i and j, of the two bases and the resultant double base term t into the array P. The function **Gen_EDBNS**(P, i) generates EDBNS representations with i unique double base terms from P. If each of the i double base terms is obtained in any order, there will be $P_i^i = i!$ permutations of i terms from the same EDBNS representation of an integer. Replicated EDBNS representations due to the permutation of the double base terms are avoided by controlling the loop indices in **Gen_EDBNS**. The function **EDBNS2Int**(S) converts the EDBNS representations in the array S into a set of integers. The function **insert**(*EDBNS_array*, I) adds the subset I of coefficients expressed in EDBNS into the array *EDBNS_array*.

This search algorithm reduces the search complexity by seeking only the EDBNS representations in the reduced space from T_{min} to a value of T that satisfies the criterion of $|F_{min}(T–1, w)| = |F_{min}(T_{min}, w)|$. Efficient EDBNS representations with bigger T but smaller F_{min} will never be missed, which guarantees that the few most efficient EDBNS representations are generated out of many redundant ones without an exhaustive search for all possible combinations. The EDBNS representations sought by our algorithm represent a very small subset of all the EDBNS representations of w-bit integers. For example, for w = 8, only two sets of EDBNS representations with T = 3 and 4 double base terms are sought and generated, from which the set with the more succinct representations is selected. Table II illustrates the frequencies of occurrences of power-of-b integers in 8-bit, 12-bit and 16-bit integers respectively, when $T = T_{min}$. The percentage of occurrences of each power-of-b integer b^β over all power-of-b integers of EDBNS is also listed.

TABLE II. The occurrences of b^β with $T = T_{min}$ for 8-, 12- and 16-bit integers.

w	b	$\beta = 1$	$\beta = 2$	$\beta = 3$	$\beta = 4$	$\beta = 5$	$\beta = 6$	$\beta = 7$	$\beta = 8$
8	3	377	79	77	0	0	0	0	0
		58.91%	12.34%	12.03%	0%	0%	0%	0%	0%
	5	107	0	0	0	0	0	0	0
		16.72%	0%	0%	0%	0%	0%	0%	0%
12	3	7338	1696	1893	0	0	0	0	0
		50.67%	11.71%	13.07%	0%	0%	0%	0%	0%
	5	1825	1729	0	0	0	0	0	0
		12.60%	11.94%	0%	0%	0%	0%	0%	0%
16	3	104644	17606	17244	12746	12583	10768	10902	10790
		42.83%	7.21%	7.06%	5.22%	5.15%	4.41%	4.46%	4.42%
	5	15184	16756	15075	0	0	0	0	0
		6.22%	6.86%	6.17%	0%	0%	0%	0%	0%

IV. Proposed Programmable FIR Filter Design Algorithm By EDBNS

The proposed programmable filter architecture is shown in Fig. 1. The structural adder block is fixed for a given number of taps N. The POBG block uses the input sample $x[n]$ to produce the set of unique power-of-b integers obtained from the quasi-minimum EDBNS search algorithm presented in Section III for all w-bit coefficients. The set of coefficients $h[i]$, i = 0, 1, … , N–1 to be convoluted with the input sample $x[n]$ establishes the control logic of the N POBS blocks to select the desired power-of-b terms from the POBG block. Each POBS block consists of T multiplexers and each multiplexer produces either a value of 0, 1 or a power-of-b integer multiple of the input $x[n]$. In each successive DBCG block, the selected $b_t^{\beta_t}$ terms are shifted by α_t bit positions according to the

quasi-minimum EDBNS representation of $h[i]$. These T or fewer $2^{\alpha_t} b_t^{\beta_t}$ terms are summed in parallel to produce a product of $h[i]$ and $x[n]$. The outputs of the N DBCG blocks are then delayed and accumulated in the final structural adder block to produce the transpose equivalent of (1), which is given by:

$$y[n] = h[0]x[n] + z^{-1}\left(h[1]x[n] + z^{-1}\left(h[2]x[n] + \cdots\right)\right) \quad (5)$$

A. Optimization of POBG block

Only one POBG block is needed as the set $F_{min}(w)$ of power-of-b integers, b^{β}, $b \in \{3, 5\}$, generated by the algorithm in Fig. 2 can be reused by all the N taps.

The power-of-b integers in F_{min} are generated in ascending order of their exponents β in the EDBNS search algorithm of Fig. 2. Since 3 = 2 + 1, the multiplication of an input variable x by 3 can be calculated by $3x = x << 1 + x$ using only one adder. Similarly, the product of x and 5 can be calculated by $5x = (2^2 + 1)x = x << 2 + x$. In general, the product of an input variable x and every element b^{β} of F_{min} can be generated by the following recursions for $\beta > 1$:

$$3^{\beta} x = 3^{\beta-1}(2+1)x = 3^{\beta-1} x << 1 + 3^{\beta-1} x \quad (6)$$

$$5^{\beta} x = 5^{\beta-1}(2^2+1)x = 5^{\beta-1} x << 2 + 5^{\beta-1} x \quad (7)$$

Using the above recursion, each successive power-of-b integer multiplier needs only one adder to implement. As the shift amount is fixed, it can be hardwired without incurring additional logic. Figs. 3(a) and (b) show the implementations of the POBG for F_{min} = {3, 9, 27, 81} and F_{min} = {5, 25, 125, 625}, respectively. Since all the elements in F_{min} are odd integers, they are not divisible by any power-of-two integers and cannot be generated from any other elements directly without requiring at least one adder. Therefore, the adder cost for this method of implementation is the optimum.

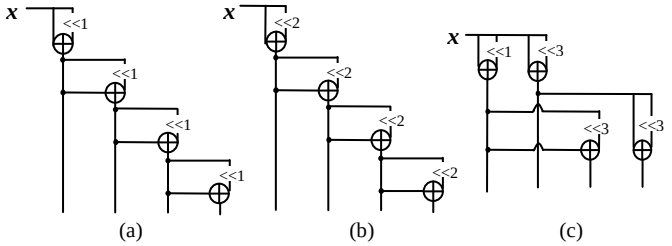

Fig. 3. POBG block implementation (a) F_{min} = {3, 9, 27, 81} without logic depth optimization (b) F_{min} = {5, 25, 125, 625} without logic depth optimization (c) F_{min} = {3, 9, 27, 81} with logic depth optimization

From, Table II, it is evident that the number of elements for b = 3 outweighs that for b = 5 in F_{min}. Thus the critical path of POBG block is dominated by the power-of-three integer multipliers. Without incurring additional adder and logic, the logic depth of the POBG can be further reduced for b = 3 by generating 3^2x by $x << 3 + x$ in parallel with $3x$ in the first adder step. Subsequent products of x and the odd and even power-of-three multipliers 3^{β} for $\beta > 2$ can also be generated in pair concurrently from their respective preceding pair of odd and even power-of-three multipliers in each adder step by

$$3^{\beta} x = 3^{\beta-2} x << 3 + 3^{\beta-2} x \quad (8)$$

Fig. 3 (c) shows the reduced logic depth minimum adder cost implementation for F_{min} = {3, 9, 27, 81}. This method of implementation reduces the logic depth of the POBG for F_{min} of w = 16 in Table II by half from 8 to 4.

B. Minimization of POBS by EDBNS reduction properties and Exponential Diophantine Equation (EDE)

Each of the N POBS blocks consists of a bank of T multiplexers with inputs feeding from the POBG. The products of every power-of-b integers and the input signal x from the POBG are routed to the inputs of these multiplexers according to their frequencies of occurrences in the EDBNS representation of the coefficient. The same partial product may appear in the inputs of several multiplexers of the same POBS block if it occurs more than once in the EDBNS representation of the same coefficient. The implementation cost of a POBS block is determined by the number of multiplexers and the complexity of each multiplexer. The complexity of a multiplexer is approximately proportional to its number of inputs and the bit width of the inputs. Therefore, one way to reduce the hardware cost of a POBS block is to minimize the total number of input lines to the T multiplexers. Although T has been constrained as a whole in the generation of the EDBNS representations for all w-bit coefficients, the EDBNS representations of some coefficients may have fewer than T terms. Therefore, some elements in F_{min} do not have to be connected to the inputs of all T multiplexers. Further reduction in the total number of inputs to the multiplexers of a POBS block can be achieved with the help of the following two reduction rules [35] for DBNS. The second property is also valid for b = 5 of EDBNS.

Property 1: *Sum of two consecutive double base terms with the same α.*

$$2^{\alpha}3^{\beta} + 2^{\alpha}3^{\beta+1} = 2^{\alpha+2}3^{\beta} \quad (9)$$

Property 2: *Sum of two double base terms with the same β.*

$$2^{\alpha}3^{\beta} + 2^{\alpha+1}3^{\beta} = 2^{\alpha}3^{\beta+1} \quad (10)$$

$$2^{\alpha}5^{\beta} + 2^{\alpha+2}5^{\beta} = 2^{\alpha}5^{\beta+1} \quad (11)$$

Obviously, every application of Property 1 reduces the number of double base terms by one, and the same situation happens for Property 2 provided that the merged higher order term $b^{\beta+1}$ exists in F_{min}. These two properties can be applied recursively until the number of double base terms in an EDBNS cannot be reduced further. This process can be generalized and modeled as an Exponential Diophantine Equation (EDE) [36].

$$\begin{aligned} &2^{\alpha_1}3^{\beta_1} + 2^{\alpha_2}3^{\beta_2} + \ldots + 2^{\alpha_k}3^{\beta_k} + 2^{\alpha'_1}5^{\beta'_1} + \ldots + 2^{\alpha'_{k'}}5^{\beta'_{k'}} \\ &= 2^{i_1}3^{j_1} + 2^{i_2}3^{j_2} + \ldots + 2^{i_l}3^{j_l} + 2^{i'_1}5^{j'_1} + \ldots + 2^{i'_{l'}}5^{j'_{l'}} \end{aligned} \quad (12)$$

where $l + l' < k + k'$.

If (12) can be solved without introducing any new element into F_{min}, the existing EDBNS for the coefficient, which is the left hand side (LHS) of (12) will be replaced by the right hand side (RHS) of (12) with less number of terms. With the aid of *Properties 1* and *2* and the EDE solutions proved in Section 3 of [35], we can match the solutions of (12) against the EDBNS representations for all w-bit coefficients. Once a solution is found, the EDBNS will be replaced by the solution with a reduced number of terms. Eventually, some coefficients are

represented with a reduced number of terms even though T for the overall w-bit coefficient sets remains unchanged. The unused input lines to the multiplexers of the corresponding POBS block can then be removed. The reduction process is summarized by the pseudo code in Fig. 4.

```
POBS_minimize(EDBNS_array, w) {
    EDE_LHS = ∅; // Initialize LHS of EDE
    EDE_RHS = ∅; // Initialize RHS of EDE
    for (i from 1 to 2^w–1) {
        j = nTerms(EDBNS_array, i);/ number of double base terms of i
        nPOT = 0; //Initialize the number of shifted power-of-3 terms
        nPOF = 0; //Initialize the number of shifted power-of-5 terms
        while (nPOT + nPOF < j) {
            if (nPOT < j – 1) {
                nPOT = nPOT + 1; // increment nPOT
            } else {
                nPOF = nPOF + 1; // increment nPOF
                nPOT = 0; // reset nPOT }
            LHS = get(EDBNS_array, i);
            RHS = Solve_EDE(LHS, nPOT, nPOF);
            if (RHS ≠ ∅) {
                replace(EDBNS_array, i, RHS);
                j = nTerms(EDBNS_array, i);
                if (nPOF ≠ 0) break; // exit while loop
                nPOF = nPOF + 1; // increment nPOF
                nPOT = 0; // reset nPOT } } }
    return EDBNS_array;
}
```

Fig. 4. Proposed POBS reduction algorithm.

The functions **nTerms**(*EDBNS_array*, *i*) and **get**(*EDBNS_array*, *i*) return the number of double based terms and the EDBNS expression respectively for the integer at the *i*-th row of EDBNS_array. The function **Solve_EDE**(*LHS*, *nPOT*, *nPOF*) solves the EDE with *nPOT* binary shifted power-of-three terms and *nPOF* binary shifted power-of-five terms by applying *Properties 1* and *2*. The function **replace**(*EDBNS_array*, *i*, *RHS*) replaces the current EDBNS at the *i*-th row of EDBNS_array by the solution *RHS* of EDE. The current minimum number of terms *j* and the expression *LHS* on the left hand side of EDE is always updated with the latest solution found so that subsequent solution for the same integer, if found, will always have a smaller number of terms. If *LHS* is already the minimum EDBNS, the function **Solve_EDE** will always return a null value. For each integer *i*, the EDE is solved by incrementing *nPOT* before *nPOF*. If the EDE is solved before the increment of *nPOF*, the program continues to search for a better solution by incrementing *nPOF* and resetting *nPOT* until either a solution is found or the sum of *nPOT* and *nPOF* reaches the number of terms of the current EDBNS of *i*. As an example, using this algorithm for the 8-bit design when $T = T_{min} = 3$ and $F_{\min}$ set is chosen to be {3, 9, 27, 5}, 136 out of the 255 integers can be implemented using two terms only. Another 26 integers can be implemented using only one term. Only 93 integers have to be expressed using three terms and cannot be further reduced. Thus, the input of each multiplexer in the POBS block is reduced from six to four compared to previous works [9] [24] where all factors from the precomputer block are connected to the subsequent multiplexers as inputs.

C. Overall design flow and work out example

The shifters in DBCG block shift the T outputs from POBS to generate the double base terms $2^{\alpha_t} b_t^{\beta_t}$ for $t = 1, 2, \ldots, T$, and the subsequent adder tree sums up the T terms to produce the coefficient multiplier output $h[i]x[n]$ of each filter tap. Given T and $F_{\min}$, the EDBNS representations of all w-bit coefficients that result in the minimum number of different α_t for each of the T POBS outputs are selected. Thus, each shifter needs only to realize k different amounts of shift, where k is determined by the minimum number of distinct exponents α_t that can appear in the t-th double base term. Each POBS output is hardwire-shifted by k different numbers of bits and fed into the k data inputs of a multiplexer. One of these k shifted versions of the POBS output will be selected from each multiplexer to compose the EDBNS representation of the programmed coefficient.

The control signals to the multiplexers and the programmable shifters are generated by an external look up table (LUT). Because the EDBNS representation of any even integers can be obtained by a double base scalar multiplication of a 2^{α} factor and an odd integer, i.e., $c = 2^{\alpha} \times c'$ where c is an even number and c' is known as a fundamental [1], the control information of the even integers are stored together with their fundamentals in the LUT, plus a 2^{α} factor stored for each even number. This will reduce the LUT size by almost half. The complete design procedure is summarized as follows:

Step 1: Compute T_{min} and F_{min} for all w-bit coefficients. Generate the EDBNS array for all the w-bit coefficients using the search algorithm presented in Section III.

Step 2: Implement the POBG block by producing all power-of-b integers in F_{min} using the method described in Section IV.A.

Step 3: Design the POBS block with T multiplexers. Each power-of-b integer from the POBG block is first connected to an input of k different multiplexers, where k is the maximum number of times that power-of-b value can appear in the EDBNS representation of any coefficient. Then, minimize the number of input lines to the multiplexers of POBS block by the algorithm presented in Fig. 4.

Step 4: Design T programmable shifters for the DBCG block. Extract the amount of shifts α for each power-of-b integer and store it in the LUT addressable by the fundamentals.

Step 5: Sum the T double base terms in DBCG by a carry save adder (CSA) tree to reduce the delay.

Design Example: Consider the design of a programmable FIR filter with 8-bit coefficients. According to Table I, $T_{min} = 3$. By fixing $T = T_{min} = 3$, $F_{min} = \{3, 9, 27, 5\}$ is obtained. By relaxing T to $T_{min} + 1 = 4$, $F_{min} = \{3\}$ is obtained. It is obvious that the cardinality of F_{min} cannot be further reduced by incrementing T. Two sets of EDBNS representations for 8-bit coefficients can be generated using $F_{min} = \{3\}$ for $T = 4$ and $F_{min} = \{3, 9, 27, 5\}$ for $T = 3$. The former EDBNS expresses all possible 8-bit coefficient values using only one power-of-b integer in POBG but four double base terms are to be added in each DBCG block. The latter EDBNS uses four power-of-b integers in POBG to express all possible 8-bit coefficient values but only three double base terms need to be added in each DBCG block.

Using the EDBNS with F_{min} = {3, 9, 27, 5} and T = 3 as an example, the POBG block is designed as shown in Fig. 5, where 3x, 5x and 9x are produced using only one adder each while 27x is generated from 3x by 3x << 3 + 3x with one additional adder. The number of adders is 4 and the logic depth is only 2. Since T = 3, only three multiplexers are needed in each POBS block. To reduce the input lines to the multiplexers, the EDEs for the EDBNS of all 8-bit coefficients are solved using the algorithm presented in Fig. 4. The number of double base terms is reduced for the EDBNS representation of each coefficient that has an EDE solution. For example, 147 is initially expressed as $2^7 3^0 + 2^4 3^0 + 2^0 3^1$. The EDE $2^7 3^0 + 2^4 3^0 + 2^0 3^1 = 2^{\alpha_1} 3^{\beta_1} + 2^{\alpha_2} 3^{\beta_2}$ has a solution of $\alpha_1 = 0$, $\alpha_2 = 4$, $\beta_1 = 1$ and $\beta_2 = 2$. Hence, the EDBNS representation for 147 is reduced to $2^0 3^1 + 2^4 3^2$ and one term has been saved. Upon minimizing the members of the EDBNS array, each multiplexer of the POBS block needs only four instead of six data inputs, including 0 (for 0 valued coefficient) and x (for $b^0 x$). Finally, each multiplexer output is connected to a programmable shifter in DBCG to generate a double base term, and the three double base terms are added up by an adder tree of two adders. Together with the tap delay registers and configurable structural adder/subtractors, the final design of this programmable filter is shown in Fig. 5.

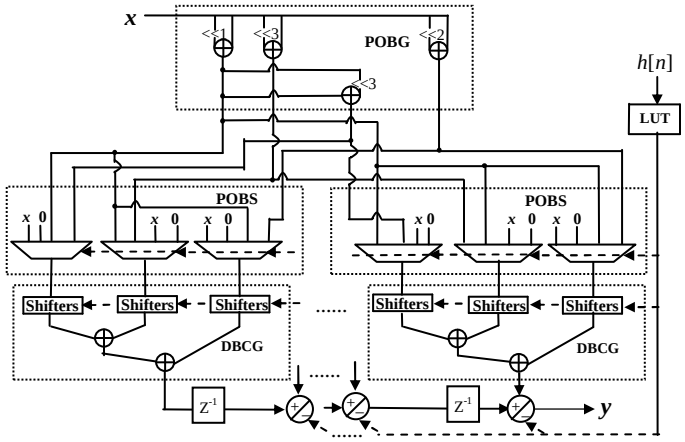

Fig. 5. 8-bit programmable FIR filter designed by proposed algorithm.

V. Synthesis Results and Discussion

The proposed EDBNS generation as well as the POSG and POBS block optimization algorithms are implemented in Matlab. More than one hundred programmable filters with the number of filter taps ranging from 10 to 100 in step size of 10 and coefficient word lengths of 8, 12 and 16 bits are designed by our proposed method and two existing computation sharing programmable FIR filter design methods, [5] and [9]. The LUT and control logic circuit are independent of the filter taps and represent a small fraction of the total cost. They are not detailed in [5] and [9] and therefore cannot be included in the circuits for comparison. An input word length of 8 bits is assumed as this is a commonly used ADC resolution. All the programmable filter architectures are described in VHDL codes and synthesized by Mentor Graphics LeonardoSpectrum using 0.18μm standard cell library. As the number of structural adders and the registers in the tap-delay line are identical in all three design methods, only the synthesis results of the TM-MCM blocks of these programmable filters are compared.

Based on *Proposition 1*, more than one set of quasi-minimum EDBNS representations with different T and F_{min} are generated by our search algorithm for a given coefficient word length. Table III shows the synthesized areas in equivalent gate counts of the TM-MCM designs obtained by our proposed EDBNS based algorithm in Fig. 2 with different values of T for 8-bit, 12-bit and 16-bit coefficients.

TABLE III. Gate counts of programmable filters of 1 to 100 taps designed with quasi-minimum EDBNS of different T for 8-bit, 12-bit and 16-bit coefficients.

N	w = 8		w = 12		w = 16		
	T = 3	T = 4	T = 4	T = 5	T = 4	T = 5	T = 6
1	406	271	568	497	1433	841	576
10	2526	2402	3556	3785	8517	5418	4640
20	4861	4774	6833	7423	16264	10435	9112
30	7176	7129	10083	11056	23939	15414	13596
40	9501	9490	13358	14708	31687	20437	18095
50	11850	11853	16642	18348	39407	25462	22569
60	14157	14208	19900	21973	47119	30450	27055
70	16482	16573	23160	25610	54818	35457	31533
80	18831	18932	26459	29253	62602	40489	36020
90	21159	21280	29710	32876	70304	45493	40502
100	23483	23653	32975	36527	78014	50496	45000

On average, the gate count for T = 4 and F_{min}= {3} is about the same as that for T = 3 and F_{min} = {3, 9, 27, 5} for programmable filters with 8-bit coefficients. For 12-bit coefficients, the designs implemented by EDBNS with T = 5 and F_{min} = {3, 9, 27} has lower gate count when the number of filter taps is smaller than 10, whereas the designs implemented by the EDBNS with T = 4 and F_{min} = {3, 9, 27, 81, 5} is more hardware efficient when the number of taps increased above 10. For 16-bit coefficients, the designs implemented by the EDBNS with T = 6 and F_{min} = {3, 9, 27} have the least hardware cost compared with those implemented by the EDBNS with T = 4 and T = 5. The results indicate that the most efficient TM-MCM implementation is neither dictated by T nor F_{min} alone. The complexity of the POBG block is governed by the number of power-of-b integers, which is the cardinality of F_{min}. On the other hand, the multiplexer cost of each POBS block is affected by the word length of the power-of-b integers and the number of double base terms T. Since only one POBG block is required irrespective of the number of filter taps N as opposed to one POBS block for each tap, the total hardware cost will be dominated by the efficient implementation of POBS block more than the efficient implementation of POBG block as N grows. This is observed in the case of 12-bit coefficients when N > 10. However, it should be highlighted that a larger T does not necessarily incurs higher multiplexer cost in the POBS block. This is because the complexity of the POBS block is also dependent on the size and number of power-of-b integers. Those designs with larger T but more efficient POBS blocks have consistently lower overall complexity regardless of N, as evinced by the 16-bit coefficient designs with T = 6.

Using the best design for each filter from Table III, we compare the gate counts of the programmable filters designed by our method against those designed by [5] and [9] in Tables

IV to VI for 8-bit, 12-bit and 16-bit coefficients, respectively. The percentage area reduction of our methods over [5] and [9] are calculated in the last two columns labeled 'ΔA [5]' and 'ΔA [9]', respectively. The equivalent gate counts against the number of filter taps *N* are also plotted in Figs. 6 to 8 to show the trend of improvements as *N* increases.

TABLE IV. Comparison of gate counts of 8-bit coefficient programmable filters of 10 to 100 taps designed by [5], [9] and the proposed method.

N	[5]	[9]	This	ΔA[5](%)	ΔA [9] (%)
10	2814	3709	2402	14.64%	35.24%
20	5238	7240	4774	8.86%	34.06%
30	7694	10798	7129	7.34%	33.98%
40	10178	14358	9490	6.76%	33.90%
50	12600	17889	11853	5.93%	33.74%
60	15062	21461	14208	5.67%	33.80%
70	17504	25006	16573	5.32%	33.72%
80	19967	28558	18932	5.18%	33.71%
90	22403	32103	21280	5.01%	33.71%
100	24869	35669	23653	4.89%	33.69%

TABLE V. Comparison of gate counts of 12-bit coefficient programmable filters of 10 to 100 taps designed by [5], [9] and the proposed method.

N	[5]	[9]	This	ΔA[5](%)	ΔA [9] (%)
10	4902	6003	3556	27.46%	40.76%
20	9473	11872	6833	27.87%	42.44%
30	14044	17719	10083	28.20%	43.09%
40	18615	23593	13358	28.24%	43.38%
50	23185	29447	16642	28.22%	43.48%
60	27724	35301	19900	28.22%	43.63%
70	32276	41183	23160	28.24%	43.76%
80	36850	47058	26459	28.20%	43.77%
90	41520	52912	29710	28.44%	43.85%
100	46091	58766	32975	28.46%	43.89%

TABLE VI. Comparison of gate counts of 16-bit coefficient programmable filters of 10 to 100 taps designed by [5], [9] and the proposed method.

N	[5]	[9]	This	ΔA[5](%)	ΔA [9] (%)
10	6714	8755	4640	30.89%	47.00%
20	13132	17345	9112	30.61%	47.47%
30	19487	25963	13596	30.23%	47.63%
40	25861	34561	18095	30.03%	47.64%
50	32238	43159	22569	29.99%	47.71%
60	38593	51785	27055	29.90%	47.76%
70	45068	60390	31533	30.03%	47.78%
80	51487	68994	36020	30.04%	47.79%
90	57847	77585	40502	29.98%	47.80%
100	64221	86218	45000	29.93%	47.81%

The results show that the TM-MCM blocks of the programmable FIR filters designed by our proposed EDBNS method have much lower logic complexity. Its average reductions in gate count over [5] are 6.96%, 28.16% and 30.16% for 8-bit, 12-bit and 16-bit coefficients, respectively. Its average gate count reductions over [9] are 33.96%, 43.21% and 47.64% for 8-bit, 12-bit and 16-bit coefficients, respectively. The reduction can be as high as 30.89% over [5] for $N = 10$ and 47.81% over [9] for $N = 100$ for the 16-bit coefficient filters. The savings are attributable to the sparsity of the quasi-minimum EDBNS, which effectively eliminates the high redundancies of common subexpressions encapsulated in its power-of-*b* terms. The adders and multiplexers of the POBG and POBS blocks have also been reduced significantly by our proposed techniques. As the number of filter taps grows, the net saving of the POBS block multiplies, which leads to a more prominent overall area reduction.

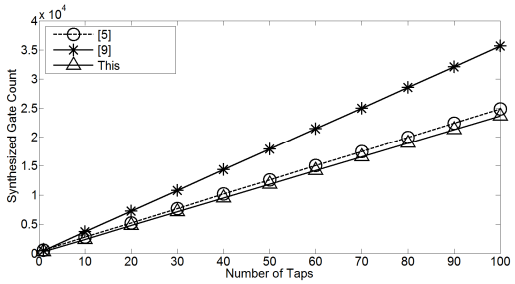

Fig. 6. Gate count versus number of taps for 8-bit programmable FIR filters designed by [5], [9] and our proposed method.

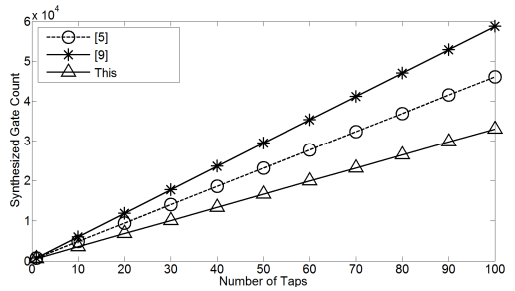

Fig. 7. Gate count versus number of taps for 12-bit programmable FIR filters designed by [5], [9] and our proposed method.

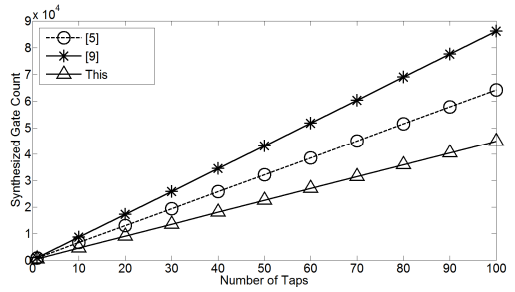

Fig. 8. Gate count versus number of taps for 16-bit programmable FIR filters designed by [5], [9] and our proposed method.

The throughput rate is another important criterion especially for programmable filter as it determines the rate of adaptation and how fast the input signal can be sampled [37]. If the critical path delay is longer than the time interval between two samples, the input samples need to be buffered. Owing to the parallel architecture of the *N* identical POBS blocks in an *N*-tap filter, the delay is independent of *N* but dependent on the word length of the coefficients. The critical path delays of the TM-MCM blocks of 8-bit, 12-bit and 16-bit coefficient programmable filters designed by our method, [5] and [9] are compared in Table VII. The percentage reduction in delay over [5] and [9] are calculated in the last two rows labeled 'ΔD [5]' and 'ΔD [9]', respectively.

TABLE VII. Comparison of the critical path delays in *ns* of *w*-bit programmable filters designed by [5], [9] and our proposed EDBNS method.

w	8	12	16
[5]	3.37	3.80	4.42
[9]	3.16	3.97	5.03
This	2.59	4.14	3.57
ΔD [5] (%)	23.15%	–8.95%	19.23%
ΔD [9] (%)	18.04%	–4.11%	29.03%

The POSG block lies in the critical path and its logic depth is a major contributor to the delay of the filter. Thanks to the reduced logic depth implementation method of the POSG block, although the hardware reduction by our method over [5] for 8-bit coefficient filters is not as significant, its critical path delay is much shorter. It should be noted that F_{min} = {3, 9, 27} is used for the 16-bit EDBNS while F_{min} = {3, 9, 27, 81, 5} is used for the 12-bit EDBNS. Due to the larger discrete adders used for the generation of 81*x* of F_{min} in the POBG block, the delay of our 12-bit coefficient filter architecture is longer than those of [5] and [9] and even our 16-bit programmable filter (although the POBG blocks for the 12-bit and 16-bit coefficients have the same adder depth). On average, the critical path delay of our TM-MCM block is reduced by 11.14% and 14.32% in comparison with those designed by [5] and [9], respectively.

Taken into consideration the importance of both logic complexity and logic depth, the area-time (AT) complexity, measured in terms of the product of equivalent gate count and critical path delay in *ns*, are plotted in Fig. 9 for the 50-tap filters with coefficient word lengths of 8, 12 and 16 bits. It is evident that the AT complexity of the filters designed by our proposed EDBNS method is lower than those of [5] and [9] by significant margins of 31.0% and 49.9%, respectively.

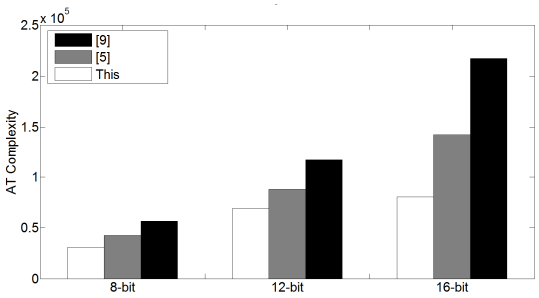

Fig. 9. Comparison of AT complexity of 50-tap filters with 8-bit, 12-bit and 16-bit coefficients designed by [5], [9] and our proposed method.

Comparison of multiplierless programmable and parallel MAC FIR filters is not provided in existing publications with a tacit understanding that adder is cheaper, faster and more power efficient than multiplier. Nonetheless, an 8-bit 100-tap programmable FIR filter using the MAC approach is designed to compare with our method. Both designs are mapped to Xilinx Artix 7 FPGA using Xilinx ISE Design Suite 14.7. Excluding the programmable shifters, our method consumes 7210 logic slices and has a critical path delay of 6.37 *ns*, which is around 37.8% and 41% lower than the MAC method. With the inclusion of the programmable shifters, our design has a critical path delay of 7.45 *ns*, which is still considerably faster than the MAC method by 31% but uses about 19.1% more logic slices. The cost of the programmable shifters could be reduced by further optimization and better mapping. As it stands, the speed of programmable shift-add network makes it a prerogative choice for real-time adaptive system.

VI. Conclusion

Vector-scalar multiplications with programmable scalars are commonly found in application-specific digital circuits. Design methodologies aiming at minimizing their implementation cost have been intensively studied by many researchers. This paper presents a radically different approach to this problem for the minimization of the TM-MCM block of programmable FIR digital filters based on the extended form of DBNS. An algorithm for the generation of quasi-minimum EDBNS has been proposed. The obtained EDBNS can be directly mapped to an efficient TM-MCM architecture consisting of only adders, multiplexers, programmable shifters and a LUT. Synthesis results show that the proposed algorithm is capable of synthesizing programmable FIR filters with an average logic complexity reduction of up to 47.81% and critical path delay reduction of up to 14.32% over its contending designs.

Acknowledgement

This research and the material reported in this document are supported by the SUTD-MIT International Design Centre (IDC) at Singapore University of Technology and Design.

References

[1] A. G. Dempster and M. D. Macleod "Use of minimum-adder multiplier blocks in FIR digital filters," *IEEE Trans. Circuits Syst. II*, vol. 42, no. 9, pp. 569-577, Sep. 1995.

[2] Y. Wang and K. Roy, "CSDC: a new complexity reduction technique for multiplierless implementation of digital FIR filters," *IEEE Trans. Circuits Syst. I*, vol. 52, no. 9, pp. 1845-1853, Sep. 2005.

[3] C. Y. Yao, Wei. C. Hsia and Y. H. Ho, "Designing hardware-efficient fixed-point FIR filters in an expanding subexpression space," *IEEE Trans. on Circuits and Syst. I*, vol. 61, no. 1, pp. 202-212, Jan. 2014.

[4] Y. Pan and P. K. Meher, "Bit-Level optimization of adder-trees for multiple constant multiplications for efficient FIR filter implementation", *IEEE Trans. on Circuits and Syst. I*, vol. 61, no. 2, pp. 455-462, Feb. 2014.

[5] J. Park, W. Jeong, H. M. Meimand, Y. Wang, H. Choo and K. Roy, "Computation sharing programmable FIR filter for low-power and high-performance applications," *IEEE J. of Solid-state Circuits*, vol. 39, no. 2, pp. 348-357, Feb. 2004.

[6] E. Grayver and B. Daneshrad, "Low power, area efficient programmable filter and variable rate decimator," *in Proc. IEEE Int. Symp. Circuits and Syst.*, pp. 341-344, Geneva, Switzerland, May 2000.

[7] S. S. Demirsoy, R. Beck, A. G. Dempster and I. Kale, "Reconfigurable implementation of recursive DCT kernels for reduced quantization noise," in *Proc. IEEE Int. Symp. on Circuits and Syst.*, Bangkok, Thailand, pp. 289-292, May 2003.

[8] J. Chen and C. H. Chang, "High-level synthesis algorithm for the design of reconfigurable constant multiplier," *IEEE Trans. on Compute Aided Designs*, vol. 28, no. 12, pp. 1844-1856, Dec. 2009.

[9] R. Mahesh and A. P. Vinod, "Reconfigurable low area complexity filter bank architecture based on frequency response masking for nonuniform channelization in software radio receivers", *IEEE Trans. on Aerospace and Electronic Syst.*, vol. 47, no. 2, pp. 1241-1255, Apr. 2011.

[10] T. Chen, Y. V. Zakharow and C. Liu, "Low-complexity channel-estimate based adaptive linear equalizer." *IEEE Signal Processing Lett.*, vol. 18, no. 7, pp. 427-430, Jul. 2011.

[11] I. Hautala, J. Boutellier and J. Hannuksela, "Programmable lowpower implementation of the HEVC adaptive loop filter," *IEEE Int. Conf. Acoustics, Speeach and Signal Processing*, pp. 2664-2668, Vancouver, Canada, May 2013.

[12] E. J. Norberg, R. S. Guzzon, J. S. Parker, L. A. Johansson and L. A. Coldren, "Programmable photonic microwave filters monolithically integerated in InP-InGaAsP," *J. of Light Wave Technology*, vol. 29, no. 11, pp. 1611-1619, Jun. 2011.

[13] M. R. Zahabi, V. Meghdadi, J. P. Cances and A. Saemi, "Mixed-signal matched filter for high-rate communication systems," *IET Signal Processing*, vol. 2, no. 4, pp. 354-360, Dec. 2008.

[14] M. S. Prakash and R. A. Shaik, "Low-area and high-throughput architecture for an adaptive filter using distributed arithmetic," *IEEE Trans. on Circuits and Syst. II*, vol. 60, no. 11, pp. 781-785, Nov. 2013.

[15] J. Chilo and T. Lindblad, "Hardware implementation of 1D wavelet transform on an FPGA for infrasound signal classification," *IEEE Trans. on Nuclear Sci.*, vol. 55, no. 1, Feb. 2008.

[16] W. Kamp and A. Brainbridge-Smith, "Multiply accumulate unit optimized for fast dot-product evaluation," *in Proc. Int. Conf. on Field-Programmable Tech.*, pp. 349 – 352, Kitakyushu, Japan, Dec. 2007.

[17] M. Cieplucha, "High performance FPGA-based implementation of a parallel multiplier-accumulator," *in Proc. 20th Int. Conf. on Mixed Design of Integrated Circuits and Syst.*, pp. 485 – 489, Gdynla, Poland, Jun. 2013.

[18] P. Tummeltshammer, J. C. Hoe, and M. Püschel, "Time-multiplexed multiple-constant multiplication," *IEEE Trans. on Compute Aided Designs*, vol. 26, no. 9, pp. 1551-1563, Sep. 2007.

[19] R. I. Hartley, "Subexpression sharing in filters using canonic signed digit multipliers," *IEEE Trans. on Circuits Syst. II*, vol. 43, no. 10, pp. 677-688, Oct. 1996.

[20] R. Paško, P. Schaumont, V. Derudder, S. Vernalde and D. Ďuračková, "A new algorithm for elimination of common subexpressions," *IEEE Trans. Compute Aided Designs*, vol. 18, no. 1, pp. 58-68, Jan. 1999.

[21] C. H. Chang, J. Chen and A. P. Vinod, "Information theoretic approach to complexity reduction of FIR filter design," *IEEE Trans. on Circuits and Syst. I*, vol. 55, no. 8, pp. 2310-2321, Sept. 2008.

[22] M. M. Peiro, E. I. Boemo, and L. Wanhammar, "Design of high-speed multiplierless filters using a nonrecursive signed common subexpression algorithm," *IEEE Trans. on Circuits Syst. II*, vol. 49, no. 3, pp. 196-203, Mar. 2002.

[23] F. Xu, C. H. Chang, C. C. Jong, "Design of low-complexity FIR filters based on signed-powers-of-two coefficients with reusable common subexpressions," *IEEE Trans. Compute Aided Designs*, vol. 26, no. 10, pp. 1898-1907, Oct. 2007.

[24] R. Mahesh and A. P. Vinod, "New reconfigurable architectures for implementing FIR filters with low complexity," *IEEE Trans. Compute Aided Designs*, vol. 29, no. 2, pp. 275-288, Feb. 2010.

[25] K. Khoo, A. Kwentus and A. N. Willson, "A programmable FIR digital filter using CSD coefficients," *IEEE J. of Solid-state Circuits*, vol. 31, no. 6, pp. 869-874, Jun. 1996.

[26] N. T. A. El-Kheir, M. S. El-Kharashi, and M. A. EL-Moursy, "A low power programmable FIR filter using sharing multiplication technique," in *Proc. IEEE Int. Conf. IC Design & Tech.*, pp. 1-4, Austin, USA, May 2012.

[27] Z. Tang, J. Zhang and H. Min, "A high-speed, programmable, CSD coefficient FIR filter," *IEEE Trans. on Consumer Electronics*, vol. 48, no. 4, pp 834-837, Nov. 2002.

[28] R. W. Reitwiesner, "Binary arithmetic," in *Advances in Computers*, New York: Academic Press, vol. 1, pp. 231-308, 1960.

[29] G. De Micheli, *Systhesis and Optimization of Digital Circuits*. New York: McGraw-Hill, Inc., 1994.

[30] V. S. Dimitrov, J. Eskritt, L. Imbert, G. A. Jullien and W. C. Miller, "The use of the multi-dimensional logarithmic number system in DSP applications," *in Proc. 15th IEEE Symp. on Computer Arithmetic*, pp. 247-254, Vail, CO, USA, Jun. 2001.

[31] V. S. Dimitrov, L. Imbert and A. Zakaluzny, "Multiplication by a constant is sublinear," *in Proc. 18th IEEE Symp. on Computer Arithmetic*, pp. 261-268, Montpellier, France, Jun. 2007.

[32] J. Adikari, V. S. Dimitrov and L. Imbert, "Hybrid binary-ternary number system for ellipic curve cryptosystems," *IEEE Trans. on Computers*, vol. 60, no. 2, pp. 254 – 265, Feb. 2011.

[33] J. Chen and C. H. Chang, "Design of programmable FIR filters using canonical double based number representation," *in Proc. IEEE Int. Symp. on Circuits and Syst.*, pp. 1183-1186, Melbourne, Australia, Jun. 2014.

[34] V. S. Dimitrov, G. A. Jullien and R. Muscedere, *Multiple-Base Number System: Theory and Applications*. CRC press, 2012.

[35] V. S. Dimitrov, G. A. Jullien and W. C. Liller, "Theory and applications of the double-base number system," *IEEE Trans. on Computers*, vol. 48, no. 10, pp. 1098 – 1106, Oct. 1999.

[36] T. N. Shorey and R. Tijdeman, *Exponential Diophantine Equations*. Cambridge Univ. Press, 1986.

[37] X. Chen, F. J. Harris, E. Venosa and B. D. Rao, "Non-maximally decimated analysis/synthesis filter banks: applications in wideband digital filtering," *IEEE Trans. on Signal Processing*, vol. 62, no. 4, pp. 852-867, Feb. 2014.

Jiajia Chen received his B. Eng. (Hons) and Ph.D. from Nanyang Technological University, Singapore, in 2004 and 2010, respectively. From Auguest 2010 to March 2012, he was a lecturer in the School of Engineering at Ngee Ann Polytechnic, Singapore. Since April 2012, he has been a lecturer in Singapore University of Technology and Design. His research interest includes computational transformations of low-complexity digital filters, programmable filters, filter architectural optimization and EEG signal processing. Dr. Chen served as Web Chair of *Asia-Pacific Computer Systems Architecture Conference* 2005 and Technical Program Committee member of *European Signal Processing Conference* 2014.

Chip-Hong Chang (S'92–M'98–SM'03) received the B.Eng. (Hons.) degree from the National University of Singapore in 1989, and the M. Eng. and Ph.D. degrees from Nanyang Technological University (NTU) in 1993 and 1998, respectively. He served as a Technical Consultant in industry prior to joining the School of Electrical and Electronic Engineering (EEE) of NTU in 1999, where he is currently an Associate Professor. He holds joint appointments with the university as Assistant Chair of Alumni of the School of EEE from 2008 to 2014, Deputy Director of the Center for High Performance Embedded Systems from 2000 to 2011, and the Program Director of the Center for Integrated Circuits and Systems from 2003 to 2009. He has coedited one book, published four book chapters and more than 200 research papers in refereed international journals and conferences. His current research interests include hardware security and trust, residue number systems, low power arithmetic circuits and digital filter design.

Dr. Chang has served as Associate Editor of *IEEE Access* since 2013, *IEEE Transactions on Circuits and Systems-I* from 2010-2013, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* since 2011, *Integration, the VLSI Journal* since 2013 and *Microelectronics Journal* since 2014. He also guest edited several journal special issues and served in many international conference advisory and technical program committees. He is a Fellow of the IET.

Feng Feng has been studying for B.Eng. Degree in computer engineering in Singapore University of Technology and Design since 2013, and he is working as a research assistant at SUTD-MIT International Design Center at the same time. His current interest area is algorithm-based high level digital IC design.

Weiao Ding is is currently pursuing B. Eng. degree in computer engineering at Singapore University of Technology and Design. He works as a research assistant at SUTD-MIT International Design Center. His research interests include digital signal processing, digital filter design and digital circuit design.

Jiatao Ding is pursuing his B.Eng. degree in computer engineering at Singapore University of Technology and Design. He is currently working as a part-time research assistant at SUTD-MIT International Design Center. His research interests include digital filter design and implementation, CAD algorithms for high-level synthesis and new logical operator design.

.