International Journal of Engineering and Technology Innovation, vol. 7, no. 3, 2017, pp. 217 - 228

A Mathematical Framework for Online Constant Coefficient Multiplication

Georgina Binoy Joseph[1,*], R. Devanathan[2]

[1]Toc H Institute of Science & Technology, Kochi, India.

[2]Hindustan Institute of Technology & Science, Hindustan University, Chennai, India.

Received 16 November 2016; received in revised form 08 April 2017; accepted 09 April 2017

Abstract

Single and Multiple constant multiplications are key operations in several digital signal processing algorithms. This paper develops a mathematical framework for a novel adaptation of the parallel shift-and-add multiplication algorithm for online arithmetic. Based on this adaptation, online constant coefficient multipliers for single constant multiplication (SCM) and multiple constant multiplications (MCM) of streaming floating-point inputs are presented. A finite impulse response filter implementation on Xilinx Virtex 6 Field programmable gate array (FPGA) is used as an example to illustrate the merits of these filters. The results of this implementation show that online multipliers reduce resource utilization, online delay and increase clock frequency in comparison to existing designs. Online multiple constant multipliers also show an average reduction of 65% in the number of slice LUTs and 37% in the number of slice registers required when compared to existing digit-serial multiple constant multipliers. Thus, the proposed online arithmetic operators appear to be good alternatives for constant coefficient multiplication.

Keywords: real-time, online arithmetic, digital signal processing, single constant multiplication, multiple constant multiplications, field programmable gate array, floating-point

1. Introduction

Processing of digital signals in real-time requires efficient online arithmetic operations on streaming data. Online arithmetic operators process inputs and generate outputs serially most significant digit first (MSDF). This reduces the complexity of the operators and the interconnections between different modules which results in a reduction in area and power dissipation. The drawback is that the number of cycles required for receiving input and delivering output increases. The execution of successive operations can be overlapped to compensate for this online delay. The resulting latencies are comparable with that of pipelined large bit-width parallel operators.

Hardware implementations of single and multiple constant multiplications (MCM) are important features of several DSP algorithms, such as finite impulse response filters and discrete transforms. These implementations lead to improved performance and reduced resource utilization and power consumption. Such multiplications are characterized by fixed multiplicands and do not need general purpose multipliers. The shift-and-add algorithm provides an efficient way of implementing multiplication by constants without using multipliers. Graph based methods and common sub-expression sharing are some of the techniques used in MCM algorithms.

While prior research on online MCM (OMCM) is not available, one can cite bit-parallel and digit-serial (least significant digit first (LSDF)) implementations of constant coefficient multipliers which appears closest to OMCM. Bull et al. [1] proposed the Bull-Horrocks (BH) algorithm for the synthesis of graphs that represent parallel implementations of coefficient multiplier arrays of digital filters. In the Bull-Horrocks Modified (BHM) algorithm proposed by Dempster et al. [2], the use of

* Corresponding author. E-mail: georgina@tistcochin.edu.in

Tel.: +919566110163

subtraction is facilitated by allowing fundamentals larger than those in the target set. RAG-n, a heuristic algorithm also presented in the same paper, is a hybrid algorithm that uses the Canonic Signed Digit (CSD) representation for MCM synthesis.

The $\mathcal{A}$ operation, on two constants u, v is defined by Voronenko et al. [3] as $\mathcal{A}_p(u,v) = |2^{l_1}u + (-1)^s 2^{l_2}v|2^{-r}$. The set $p = (l_1, l_2, s, r)$ is called the parameter set of $\mathcal{A}_p$. Imposing constraints on the variables in p will result in different classes of MCM problems. The BH algorithm imposes constraints that do not allow right shifts ($r = 0$) and restrict the values of l_1, l_2 and s such that $\mathcal{A}_p(u,v) < s$, the current constant being synthesized. The BHM algorithm permits only one of the left shifts l_1, l_2 to be non-zero so that $\mathcal{A}_p(u,v)$ results in odd values. It also permits $\mathcal{A}_p(u,v) \leq 2\max(\mathrm{A})$, where A = the largest constant in the set of MCM constants. The Hcub algorithm presented by Voronenko et al. [3] produces better results than the algorithms in [1-2] by using maximum-benefit and cumulative-benefit functions for the selection of fundamentals, but it is computationally more expensive.

Exact and approximate algorithms developed by Aksoy et al. [4-8], cast the MCM problem as an Integer Linear Programming problem based on Boolean networks. Solutions to the minimum area problem with or without delay constraints are obtained for parallel and digit-serial implementations. The results are compared with the RSAG-n and the RASG-n algorithms proposed by Johansson et al. [9-10]. The RSAG-n algorithm chooses the intermediate fundamentals that require minimum number of shifts, while the RASG-n algorithm adapts the RAG-n to serial inputs.

FPGAs provide a platform for efficient hardware implementations of operations required in DSP algorithms: (1) the bit level granularity of FPGAs permits the choice of standard and non-standard number representations with just the right number of bits and the right number of operations on these bits leading to efficient hardware implementations. (2) Logic can be customized to build specialized operators that are tailored to a particular application. (3) FPGAs can be used to process computations spatially rather than temporally as in processors allowing us to use both parallelism and pipelining to accelerate computations and increase throughput as emphasized in the book by Vanderbauwhede et al. [11]. (4) FPGAs are hardware programmable with dynamic reconfiguration capability. Floating-point (FP) implementations are more suitable than fixed point representations for digital signal processing (DSP) applications that require a large dynamic range. Two methods for implementing multipliers of floating-point input operands with integer coefficients in FPGAs - one method using adder arithmetic and the other using embedded multipliers – are presented by Kumm et al. [12]. Bit-parallel implementations of single and multiple constant multiplications using a Look up Table method for FPGAs is proposed by Faust et al. [13].

Online multiplication being MSDF arithmetic enables the choice of the required amount of precision in the output. The computation can be terminated once the required precision of the application has been achieved leading to truncated multipliers with variable output precision. This is in line with the "compute just right" concept advocated by Vanderbauwhede et al. [11]. Computing just right exploits the built-in error resilience of the neuromorphic or DSP application and, thus, reduces area and latency and conserves power. This is the underlying principle of approximate computing [14-15]. The challenge in implementing shift-and-add constant coefficient multipliers for online multiplication is the difficulty in implementing left shifts as a left shifter implementation becomes a non-causal system in online arithmetic. Therefore, a technique that uses right shifts instead of left shifts is proposed in this paper.

The main contributions of the paper are: (1) Online multipliers for multiplying a floating-point input operand with a single constant coefficient multiplier (OSCM) using graph-based techniques are presented. A novel technique to adapt the shift-and-add algorithm to online arithmetic is proposed. (2) A mathematical framework for online multiple constant multiplication (OMCM) is presented. Constraints that are imposed are the maximum adder and online delay constraints. The key parameters of these multipliers – online delay δ, clock frequency and resource utilization are analyzed and compared. (3) The proposed multipliers are used in the implementation of an online FIR filter on FPGAs and comparative results are

presented. (4) A comparison of the resource utilization of OMCM networks with digit-serial implementations is presented.

To summarise the rest of the paper, Section 2 presents OSCM using carry-save (CS) number representation. Section 3 presents a mathematical framework for the synthesis of multiple constant carry-save online floating point multipliers. In Section 4, implementations of a FIR filter based on online single and multiple constant multiplications are presented. The implementation results are presented in Section 5. The conclusions drawn from the comparison of the alternative designs are presented in Section 6.

2. Online Single Constant Multiplication

Constant coefficient multipliers implemented using the shift-and-add technique can be represented by a directed acyclic graph (DAG). The DAG of a general shift-and-add multiplier is shown in Fig. 1. The flow of data in the graph is unidirectional from the input x to the output y. Each node except the input node in the graph is an adder, and each constant $s_i = 2^i$ on the edge represents a left shift of i. The graph in Fig. 1 represents the equation $y = c.x = s_i.x + s_j.x$ where $c = s_i + s_j$, $s_i = 2^i, s_j = 2^j$.

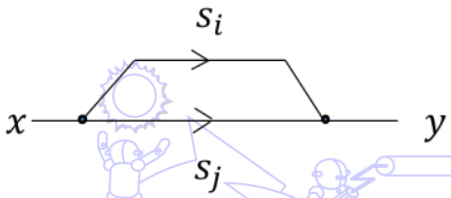

Fig. 1 Graph representation of shift-and-add multiplication

For parallel implementations, such multipliers can be implemented using left shifters and adders. However, hardware implementations of left shifters are not suitable for online arithmetic because a left shift in MSDF arithmetic would make the systemnon-causal.

Online constant coefficient multiplication can be implemented using right shifts and additions. The parameter set for the $\mathcal{A}$ operation in online arithmetic is $p = (l_1, l_2, s, r)$where $l_1 = -j$, $l_2 = -k, s = 0, r = +l$, where the scaling factor is 2^l. The carry-save representation is preferred over the signed-digit representation to prevent the use of costlier subtraction and reduce the online delay as shown by Joseph et al. [16]. Online addition can be performed using online full adders adding 4 operands, olFA4, proposed by Olivares et al. [17]. A right shift of $(x \gg i)$ can be easily implemented by delaying x by i time units. As the right shift implies division by 2^i, the output is a scaled version of $y[n]$. Theorem 1 forms the basis of a novel adaptation of the shift-and-add multiplication technique to online processing using the parameter set for online arithmetic. This enables the substitution of left shifts with right shifts. The output is thus scaled by the factor $2^{k+\lfloor log_2\, \alpha \rfloor}$, which can be taken care by adjusting the exponent.

Theorem 1: *Online multiplication of an input vector by the constant $\varphi = \alpha\lambda$,where α is an odd number, $\alpha \neq 1$ and $\lambda = 2^k$, $k = 0,1,2,3, \ldots$ is the largest of the power of two factors of φ,can be implemented by adding right shifted (delayed) versions of the input and multiplying by a factor $2^{k+\lfloor log_2\, \alpha \rfloor}$.*

Proof: Let $\alpha = 2^{k_1} + 2^{k_2} + 2^{k_3} + \cdots + 2^{k_m}$, $k_1 > k_2 > k_3 \ldots > k_m = 0$

$$= 2^{k_1}\,(1 + 2^{-(k_1 - k_2)} + 2^{-(k_1 - k_3)} + \cdots + 2^{-(k_1 - k_m)}).$$

As $\varphi\, x = \alpha\lambda x$ and $\lambda = 2^k$, $y = \varphi\, x = 2^{k+k_1}(x + \sum_{i=2}^{m} 2^{-(k_1 - k_i)}\, x) = 2^{k+\lfloor log_2\, \alpha \rfloor}(x + \sum_i 2^{-(k_1 - k_i)}\, x)$

The term $2^{-(k_1-k_i)}x$ represents a right shifted version of x. In online arithmetic, this represents x delayed by $k_1 - k_i$ time units. Thus, $\varphi\, x$ can be written as the sum of shifted versions of x multiplied by 2^{k+k_1} where $k_1 = \lfloor \log_2 \alpha \rfloor$ is the largest power of 2 in the binary representation of α. The above expression can be represented by the graph in Fig. 2.

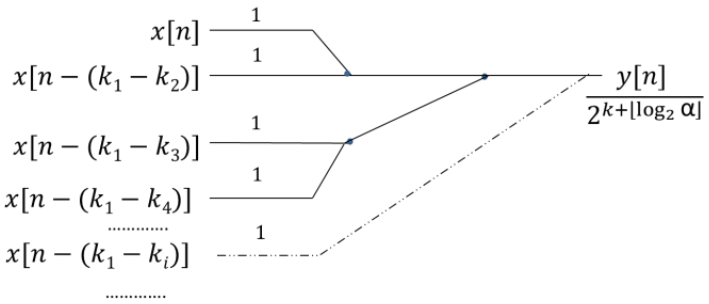

Fig. 2 Multiplication by $\varphi = \alpha\lambda$, $\lambda = 2^k, \alpha \neq 1$

2.1. Multiplication by multiples of 3: 3λ, $\lambda = 2^k$, $k = 0,1,2,3,\ldots$

Multiplication by multiples of 3 is used to illustrate the proposed technique based on Theorem 1. The constant 3 can be represented as $3\lambda, \lambda = 2^k, k = 0$

$$3x = (2^{k_1} + 2^{k_2})x = 2^{k_1}\left(x + 2^{-(k_1-k_2)}x\right), k_1 = 1, k_2 = 0$$

Therefore, the inputs to the adder for implementing multiplication by 3 are $x[n]$ and $x[n]$ delayed by $(k_1 - k_2)$ time units, that is, $x[n-(k_1-k_2)] = x[n-1]$. Fig. 3 represents a multiplier implementing multiplication by 3. The output scaling factor is $\frac{1}{2^{k+\lfloor \log_2 3 \rfloor}} = \frac{1}{2^{0+1}} = \frac{1}{2}$.

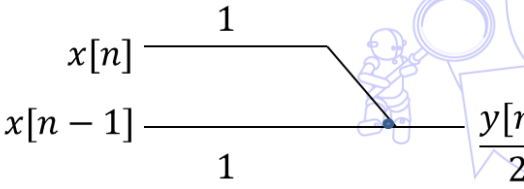

Fig. 3 Multiplication by 3

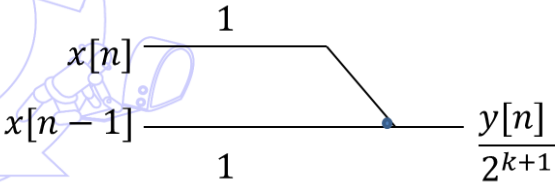

Fig. 4 Multiplication by 3 λ

The generalization is shown in Fig. 4 for multiplication by a constant 3×2^k, where k is an integer such that $k \geq 0$. The online delay of this operation, which is the online delay of the olFA4, is such that $\delta = 2$.

2.2. Exponent adjustment

The scaling of the output can be taken care of by adjusting the exponent. The exponent has to be incremented by $Exp\ Adj = k + \lfloor log_2\ \alpha \rfloor = \lfloor log_2\ \varphi \rfloor$ which can be carried out using a 3 operand online full adder with an online delay of 1.

2.3. Normalization of floating point outputs

Normalization can be carried out by determining the bit growth. The maximum bit growth will be 2φ, as the mantissa of IEEE-754 FP numbers have a range of [1, 2]. The actual bit growth can be determined by detecting the position of the leading one in the output. If the bit growth is ϵ, the exponent can be normalized by adding $Exp\ Adj + \epsilon$ to it. The mantissa can be normalized by scaling, that is delaying it by ϵ time units. This increases the online delay to $1 + e + m_d$, where e is the number of exponent bits and m_d is the online delay of mantissa shift-and-add network. When single constant coefficient multipliers are used in DSP algorithms such as in FIR filters, the maximum bit growth up to the final stage can be predicted and the normalization operation can be performed once as a final step. A separate exception handling module does exception handling.

2.4. Online delay

Online delay is used as a parameter to determine if direct implementation of constant coefficient multiplication based on Theorem 1 is more suitable than an implementation of large constants using smaller fundamentals. This enables the search algorithm to choose optimal fundamentals. The online delay of multiplication by a constant is given by Theorem 2.

Theorem 2: *The online delay of the online multiplication implementation of Theorem 1 is* $\delta = 2 \times \lceil log_2 \rho \rceil$, *where the Hamming weight of* α *is* ρ.

Proof: The depth of the adder tree required to sum ρ terms is $\lceil log_2 \rho \rceil$. Each adder (olFA4) has an online delay = *2*. Therefore, the online delay of the online SCM is $\delta = 2 \times depth\ of\ adder\ tree = 2 \times \lceil log_2 \rho \rceil$

Corollary 1: *If* $2^{v-1} < \varphi < 2^v$, *then the maximum number of adders required is* $v - 1$ *and the maximum online delay is* $\delta_{max} = 2 \times \lceil log_2 v \rceil$.

Proof: As $2^{v-1} < \varphi < 2^v$, φ can be written as $\varphi = 2^{v-1} + \theta$ where θ lies in the range $[1, 2^{v-1})$. The maximum number of non-zero digits in θ is $v - 1$ thus requiring the addition of v terms. Therefore, the maximum number of adders $= v - 1$. The depth of the adder tree to add v terms is $\lceil log_2 v \rceil$. Hence the maximum online delay $\delta_{max} = 2 \times \lceil log_2 v \rceil$.

The special case of Theorems 1 and 2 when $\alpha = 1$ is stated in Corollary 2 next.

Corollary2: *Online Single Constant Multiplication by* $\varphi = \alpha\lambda$, *where* $\alpha = 1$ *and* $\lambda = 2^k$, $k = 0,1,2,3, \ldots$ *is the largest power of two factor of* φ, *does not require the use of an adder network and online delay is zero*.

Proof: A simple shift operation is sufficient to carry out multiplication by powers of 2. Thus adders are not required. In the case of online arithmetic, the shift need not be performed and hence there is no online delay. The multiplication factor from Theorem 1 is $2^{k+\lfloor \log_2 1 \rfloor} = 2^k$.

3. Online Multiple Constant Multiplication

Online multiple constant multiplications (OMCM) involve multiplication of a single input by different constant coefficients. Fig. 5 shows a graph representation of an instance of the MCM problem given by the equations:

$$y_1 = s_1 x + s_2 x,\ y_2 = s_3 x + s_4 x,\ \ y = s_5 y_1 + s_6 y_2$$

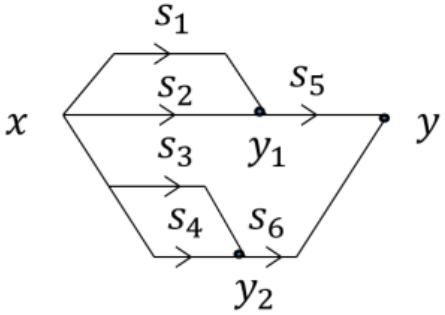

Fig. 5 Graph representation of an instance of the MCM problem

The intermediate nodes represent the output of multiplication by different constants. The intermediate results are called fundamentals [2] and can be used to produce the results of multiplication by larger constants. By combining results in this way, networks that use fewer resources can be synthesized. For example, the online adder-network in Fig. 6 realizes an instance of the MCM problem for multiplication of a single input $x[n]$ by multiple constants 3, 11 and 15.

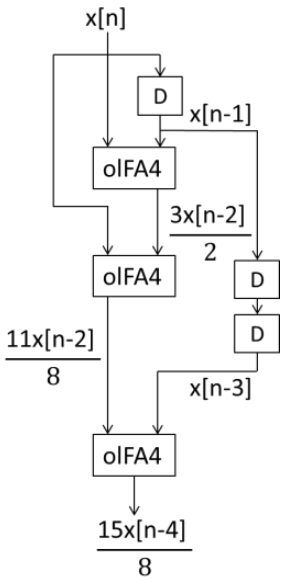

Fig. 6 Online adder network for an instance of MCM

The realization uses intermediate results:

$$11x[n] = 8x[n] + 3x[n], 15x[n] = 4x[n] + 11x[n]$$

The search for fundamentals to obtain an optimal MCM network is NP-complete. Constraints have to be imposed on the search to obtain a solution in polynomial time. The number of online adders required and the online delay are two metrics used to select fundamentals for synthesizing larger constants. Theorems 3-6 establish the effect of the choice of type and number of fundamentals on these two metrics. The proposed online MCM technique uses a structure similar to the RAG-n algorithm. It has an optimal part that attempts to synthesize the constants with one extra adder per constant and a heuristic part that operates under a maximum adder or maximum online delay constraint.

3.1. Summation of intermediate constant and shifted input

Theorems 3 and 4 together with Corollaries 3 and 4 state some basic results involved in the number of adders required and the online delay when a larger constant is formed by adding an intermediate smaller constant and shifted input.

Theorem 3: *If a larger constant φ_2 is formed by adding a smaller constant φ_1 and 2^k, $k = 0,1,2,3,\ldots$, then the number of olFA4s n required for OMCM is $n_1 + 1$, where n_1 is the number of olFA4s required to implement multiplication by φ_1.*

Proof: $\varphi_2 = \varphi_1 + 2^k$ (1)

2^k is a shift operation and does not require adders. Therefore, $\varphi_2 x = \varphi_1 x + 2^k x$ requires only one extra olFA4 to perform the addition of $\varphi_1 x$ and $2^k x$. Hence, $n = n_1 + 1$.

Corollary 3: *Considering φ_2 formed as in (1), OMCM does not increase the number of olFA4s for $\varphi_2 x$ and can result in a reduction in the number of olFA4s required if $n_2 > 1$, where n_2 is the number of olFA4s required to implement single constant multiplication by φ_2.*

Proof: Let the number of olFA4s required to implement $\varphi_i x$ be n_i. Then n_i will be greater than or equal to 1. For $n_2 = 1$, number of olFA4s $= n_1 + n_2 = n_1 + 1 = n$. Thus, there is no increase in the number of olFA4s. For $n_2 = 2$, number of olFA4s $= n_1 + n_2 = n_1 + 2$. In general, for $n_2 = p$, number of olFA4s $= n_1 + n_2 = n_1 + p$. Reduction in the number of olFA4s $= n_1 + p - n = p - 1$.

Theorem 4: *If a larger constant φ_2 is formed by adding a smaller constant φ_1 and 2^k, $k = 0,1,2,3,\ldots$, then the online delay for OMCM implementation of $\varphi_2 x$ is $\delta_{2MCM} = \delta_1 + 2$, where the online delay due to multiplication by φ_1 is δ_1.*

Proof: From Theorem 3, only one extra addition is required to implement $\varphi_2 x$ if $\varphi_1 x$ is already implemented. $\delta_{2MCM} = \delta_1 + online\ delay\ of\ one\ olFA4 = \delta_1 + 2$.

Corollary4: *OMCM does not increase the online delay only if* $\lceil log_2 \rho_1 \rceil + 1 \leq \lceil log_2 \rho_2 \rceil$, *where* ρ_1 *and* ρ_2 *are the Hamming weights of* φ_1 *and* φ_2 *respectively.*

Proof: Online delay for multiplication by $\varphi_i = \alpha_i \lambda$ is $\delta_i = 2 \times depth\ of\ adder\ tree = 2 \times \lceil log_2 \rho_i \rceil$, where ρ_i is the number of nonzero terms in the binary representation of α_i. $\delta_{2MCM} = \delta_1 + 2 = 2 \times \lceil log_2 \rho_1 \rceil + 2$. $\delta_{2MCM} \leq \delta_2$ only if $2 \times \lceil log_2 \rho_1 \rceil + 2 \leq 2 \times \lceil log_2 \rho_2 \rceil$, i.e. $\lceil log_2 \rho_1 \rceil + 1 \leq \lceil log_2 \rho_2 \rceil$.

3.2. Summation of q intermediate constants

Theorems 5 and 6 together with Corollaries 5-7 state some basic results involved in the number of adders required and the online delay when q fundamentals are used to form a larger constant.

Theorem 5: *If a larger constant* φ_ϑ *is formed by adding smaller constants* φ_i, *i.e.* $\varphi_\vartheta = \varphi_1 + \varphi_2 + \cdots + \varphi_q = \sum_{i=1}^{q} \varphi_i$ *where* $\varphi_1 < \varphi_2 < \cdots < \varphi_q < \varphi_\vartheta$, *then* $q - 1$ *additions are required to obtain* φ_ϑ *if* φ_i , $i = 1, 2, \ldots, q$ *are already computed.*

Proof: For $\varphi_\vartheta = \varphi_1 + \varphi_2$

$\varphi_\vartheta x = \varphi_1 x + \varphi_2 x$. As $\varphi_1 x$ and $\varphi_2 x$ have already been implemented; only one extra olFA4 is required for adding $\varphi_1 x$ and $\varphi_2 x$ to obtain $\varphi_\vartheta x$. The number of olFA4s for MCM, $n = n_1 + n_2 + 1$, where n_i is the number of olFA4s required to implement multiplication by $\varphi_i, i = 1, 2$.

For $\varphi_\vartheta = \varphi_1 + \varphi_2 + \cdots + \varphi_q = \sum_{i=1}^{q} \varphi_i$;

$\varphi_\vartheta x = \sum_{i=1}^{q} \varphi_i x$ In this case, $n = n_1 + n_2 + \cdots + n_q + \mathrm{q} - 1 = \sum_{i=1}^{q} n_i + q - 1$.

Corollary 5: *OMCM does not increase the number of olFA4s for* $\varphi_\vartheta x = \varphi_1 x + \varphi_2 x$ *and can result in a reduction in the number of olFA4s required if* $n_\vartheta > 1$ *where* n_ϑ *is the number of olFA4s required to implement multiplication by* φ_ϑ.

Proof: Let n_i be the number of olFA4s required to implement $\varphi_i x$ .Then, $n_i \geq 1$. For $n_\vartheta = 1$, number of olFA4s $= n_1 + n_2 + n_\vartheta = n_1 + n_2 + 1 = n$. Thus, there is no increase in the number of olFA4s. For $n_\vartheta = 2$, number of olFA4s $= n_1 + n_2 + 2$. In general, for $n_\vartheta = p$, number of olFA4s $= n_1 + n_2 + n_\vartheta = n_1 + n_2 + p$. Reduction in the number of olFA4s $= n_1 + n_2 + p - n = p - 1$.

Corollary 5: *OMCM does not increase the number of olFA4s for* $\varphi_\vartheta x = \varphi_1 x + \varphi_2 x + \cdots + \varphi_q x = \sum_{i=1}^{q} \varphi_i x$ *if* $n_\vartheta = q - 1$ *and can result in a reduction in the number of olFA4s required if* $n_\vartheta > q - 1$. *Thus, an upper bound on the value of* q *for* $\varphi_\vartheta = \sum_{i=1}^{q} \varphi_i$ *to reduce the number olFA4s required is* $q < n_\vartheta + 1$.

Proof: For $n_\vartheta = p$, number of olFA4s $= n_1 + n_2 + \cdots + n_q + n_\vartheta = \sum_{i=1}^{q} n_i + p$. Difference in the number of olFA4s required $= \sum_{i=1}^{q} n_i + p - n = \sum_{i=1}^{q} n_i + p - (\sum_{i=1}^{q} n_i + q - 1) = p - (q - 1)$. There is no increase in the number of olFA4s required if $n_\vartheta = p = q - 1$. There will be a reduction in the number of olFA4s required if $n_\vartheta > q - 1$

Theorem 6: *If a larger constant* φ_ϑ *is formed by adding smaller constants* φ_i, *i.e.* $\varphi_\vartheta = \varphi_1 + \varphi_2 + \cdots + \varphi_q = \sum_{i=1}^{q} \varphi_i$ *where* $\varphi_1 < \varphi_2 < \cdots < \varphi_q < \varphi_\vartheta$, *then the online delay for the OMCM implementation of* $\varphi_\vartheta x$ *is* $\delta_{\vartheta MCM} = max(\delta_1, \delta_2, \ldots, \delta_q) + 2 \times \lceil log_2 q \rceil$, *where the online delay due to multiplication by* $\varphi_i, i = 1,2,..q$ *is* δ_i.

Proof: From Theorem5, $q - 1$ extra additions are required to implement $\varphi_\vartheta x$ if $\varphi_1 x, \varphi_2 x, \ldots, \varphi_q x$ are already implemented.

$$\delta_{\vartheta MCM} = max(\delta_1, \delta_2, \ldots, \delta_q) + 2 \times depth\ of\ adder\ tree\ for\ q - 1\ additions = max(\delta_1, \delta_2, \ldots, \delta_q) + 2 \times \lceil log_2 q \rceil$$

Corollary7: *OMCM does not increase the online delay only if* $max(\lceil log_2 \rho_1 \rceil, \lceil log_2 \rho_2 \rceil, \dots, \lceil log_2 \rho_q \rceil) + \lceil log_2 q \rceil \leq \lceil log_2 \rho_\vartheta \rceil$ *where* ρ_i *is the Hamming weight of* $\varphi_i, i = 1,2,..q$.

Proof: $\delta_i = 2 \times depth\ of\ adder\ tree = 2 \times \lceil log_2 \rho_i \rceil$. $\delta_{\vartheta MCM} = max(\delta_1, \delta_2, \dots, \delta_q) + 2 \times \lceil log_2 q \rceil$ from theorem 6.

$= max(2 \times \lceil log_2 \rho_1 \rceil, 2 \times \lceil log_2 \rho_2 \rceil, \dots, 2 \times \lceil log_2 \rho_q \rceil) + 2 \times \lceil log_2 q \rceil$.

$\delta_{\vartheta MCM} \leq \delta_\vartheta$ only if $max(2 \times \lceil log_2 \rho_1 \rceil, 2 \times \lceil log_2 \rho_2 \rceil, \dots, 2 \times \lceil log_2 \rho_q \rceil) + 2 \times \lceil log_2 q \rceil \leq 2 \times \lceil log_2 \rho_\vartheta \rceil$.

i.e. if $max(\lceil log_2 \rho_1 \rceil, \lceil log_2 \rho_2 \rceil, \dots, \lceil log_2 \rho_q \rceil) + \lceil log_2 q \rceil \leq \lceil log_2 \rho_\vartheta \rceil$. Hence the result.

Based on the results of Theorems 3-6 the following rules can be stated.

3.3. Rules for online multiple constant multiplications (OMCM)

Rule 1: The choice of whether to perform OSCM for $\varphi_2 x$ or OMCM as $\varphi_2 x = \varphi_1 x + 2^k x$ is governed by the trade-off between the impact on the number of olFA4s $n = n_1 + 1$ as given by Theorem 3 and the effect on online delay $\delta_{2MCM} = \delta_1 + 2$ as given by Theorem 4.

Rule 2: The choice of whether to perform OSCM for $\varphi_\vartheta x$ or OMCM as $\varphi_\vartheta x = \varphi_1 x + \varphi_2 x + \cdots + \varphi_q x = \sum_{i=1}^{q} \varphi_i x$ is governed by the trade-off between the impact on the number of olFA4s $n = n_1 + n_2 + \cdots + n_q + q - 1 = \sum_{i=1}^{q} n_i + q - 1$ as given by Theorem 5 and the effect on online delay $\delta_{\vartheta MCM} = max(\delta_1, \delta_2, \dots, \delta_q) + 2 \times \lceil log_2 q \rceil$ as given by Theorem 6.

3.4. Separation cycles and minimum theoretic initiation interval

Online arithmetic permits overlapping the processing of successive instances of input vectors to increase throughput. However, careful consideration of the impact of the online delay of exponent adjustment and mantissa multiplication is required to determine the minimum theoretic initiation interval between two successive instances of the input vector. The separation cycles are the clock cycles required before the digits of the next instance of the input vector can be input to the online module. The number of separation cycles is equal to the online delay of the module.

3.4.1. Separation cycles required for exponent adjustment operation

Let the online delay of the operation adjusting the exponent = δ_e. Then, the number of separation cycles required before the next instance of the exponent can be input to the exponent adjustment module is δ_e cycles. However, since after the exponent digits of the FP number are input, the mantissa digits are input to the mantissa multiplier module, the minimum number of separation cycles s_e required reduces to

$$s_e = \begin{cases} \delta_e - m \ if\ \ \delta_e - m > 0 \\ 0 \ if\ \ \delta_e - m \leq 0 \end{cases} \tag{2}$$

3.4.2. Separation cycles required for mantissa multiplication operation

Let the online delay of the mantissa shift-and-add network = m_d. Then, the number of separation cycles required before the next instance of the mantissa can be input to the shift and add network is m_d cycles. However, since after the mantissa digits of the FP number are input, the sign and exponent digits of the next instance are input to the sign and exponent adjustment modules; the minimum number of separation cycles s_m required reduces to

$$s_m = \begin{cases} m_d - (1+e) \ if\ \ m_d - (1+e) > 0 \\ 0 \ if\ \ m_d - (1+e) \leq 0 \end{cases} \tag{3}$$

3.4.3. Minimum theoretic initiation interval

The initiation interval is the total number of clock cycles required from the first digit of the input to the last digit of the output of a particular instance of the FP vector. The minimum theoretic initiation interval ii between two successive FP input vectors is $ii = r + max(s_e, s_m)$, where r is the number of digits in the FP number including the sign digit. The initiation interval has a direct impact on throughput.

4. Online Single and Multiple Constant Multipliers in FIR Filters

In this section, the online multipliers described in Sections 2 and 3 are used in FIR filter implementations to illustrate their application.

4.1. Direct form FIR filter implementation

Fig. 7 shows the Direct Form implementation of a FIR filter given by the equation.

$$y[n] = 3x[n] + 11x[n-1] + 15x[n-2] + 11x[n-3] + 3x[n-4] \quad (4)$$

The multiplicands for each multiplier are different and hence this filter was realized using individual online single constant multipliers for each coefficient. The outputs of the multipliers are summed up using an online carry-save full adder tree.

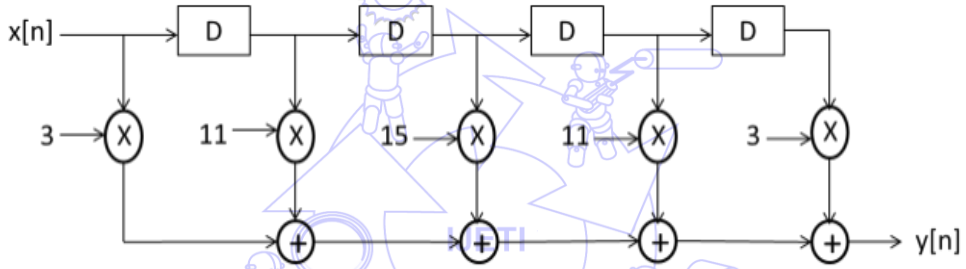

Fig. 7 Direct form FIR filter

4.2. Direct transposed form FIR filter implementation

The Direct Transposed Form of the same filter is shown in Fig. 8. As the input to each constant multiplier is the same, the constant coefficient multipliers of this filter form were therefore realized using the online Multiple Constant Multiplier network shown in Fig. 8.

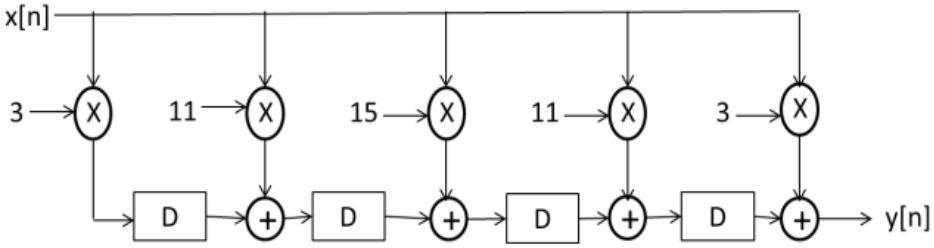

Fig. 8 Direct transposed form FIR filter

5. Implementation and Simulation

The synthesized constants were implemented on the Virtex family of FPGAs. The implementation results for the Virtex 6 (xc6vlx75t-3ff484) FPGA are presented below. Mathematical analysis was used to validate the designs which were also verified by simulation using test benches that simulated the designs at all corners. Timing and I/O constraints were set in the User Constraint Process of Xilinx ISE. Post Place and Route, the XPower Analyzer tool was invoked to evaluate power consumption.

5.1. Online single constant multipliers

Online single constant multipliers were implemented for multiplication by $\varphi = \alpha\lambda$, $\lambda = 2^k$, $k = 0, 1, 2, 3\ldots$ The results of the implementation are presented in Table 1. The online delay δ presented in Table 1 is the online delay m_d of the shift-and-add network. Online delay with normalization will be $m_d + n_d$, where the normalization delay $n_d = 1 + e$, where e is the number of bits in the exponent.

Table 1 Online Single Constant Multiplication by φ

$\Phi = \alpha\lambda$	δ	Exp Adj*	Delay (ns)	Power-delay product (ns-mW)	Clock frequency (MHz)	LUTs
1λ	0	k + 0	1.280	1292.74	781.25	26
3λ	2	$k+1$	1.348	1361.28	741.84	31
5λ	2	$k+2$	1.348	1361.29	741.84	30
7λ	4	$k+2$	1.468	1482.78	681.20	40
9λ	2	$k+3$	1.268	1280.43	788.64	32
11λ	4	$k+3$	1.633	1648.95	612.37	40
13λ	4	$k+3$	1.439	1453.29	694.93	42
15λ	4	$k+3$	1.920	1939.62	520.83	47

*The exponent of the result of the multiplication has to be incremented by Exp Adj $= \lfloor log_2 \varphi \rfloor$ as given in 2.2.

5.2. Online constant coefficient multipliers FIR filter implementation

Table 2 presents a comparison of OSCM and OMCM based mantissa module implementations of the FIR filters of Fig. 7 and Fig. 8. The Direct Transposed form (OMCM) shows on an average 23% improvement in clock frequency and uses 89% fewer slice LUTs and 91% fewer slice registers than existing designs. The existing designs used for comparison are FPGA IP core based designs - one using distributed memory and a second one using dedicated multipliers, as well as an implementation using the approximate RoBA multiplier presented by Zendegani et al. [15]. The Verilog implementation available for Virtex 6 FPGAs is used for IP core based designs. The increase in δ of the IP core based design can be attributed to the fact that the IP core multipliers have been designed for parallel arithmetic and hence use of these multipliers for online arithmetic requires a conversion of serial data to parallel. This can be circumvented by high-speed serial I/Os that have gigabit serializer–deserializer architectures. However, the use of parallel addition and multiplication results in an increase in delay and resource utilization.

Table 2 FIR filter mantissa module implementation

FIR Filter Form	δ	Power- delay product (ns-mW)	Clock frequency MHz	No. of Slice LUTs	No. of Slice Registers	No. of Slices
Direct Form – online (OSCM)	11	2069.292	488.520	50	84	29
Direct Transposed Form – online (OMCM)	11	1982.799	513.875	27	30	19
Direct Form Using FPGA IP Core (Distributed Memory)	34	2793.547	363.108	283	350	119
Direct Form Using FPGA IP Core (Dedicated Multipliers with DSP48E1s)	34	2984.43	340.252	205 (DSP48E1s = 2)	350	101
Direct Form Using Approximate RoBA multipliers [15]	34	2105.512	481.232	260	364	104

5.3. OMCM versus Digit- serial constant coefficient multipliers

The resource utilization of the online multiple constant multiplication network is compared with the digit-serial (LSDF) MCM implementations of Hcub + ILP-DS algorithm [4], the Exact CSE algorithm [5] and the MINAS-DS algorithm [18] in terms of the number of slice LUTs and registers for sets of varying numbers of constants (see Figs. 9-10). Sets of randomly generated 12 bit constants were generated using the *randi* function in MATLAB.

The Exact CSE, Hcub and the MINAS-DS algorithms use the SD number representation which on an average requires 34% fewer non-zero bits compared to the binary CS representation. However, the SD number representation requires the use of costly online SD subtractors. On an average, the Exact CSE, Hcub and MINAS-DS algorithms require 68%, 64% and 63% more slice LUTs respectively compared to the OMCM algorithm as can be seen from Fig. 9. Further, Fig. 10 shows that the Exact CSE, Hcub and MINAS-DS algorithms require 42%, 35% and 34% more slice registers respectively than the OMCM algorithm.

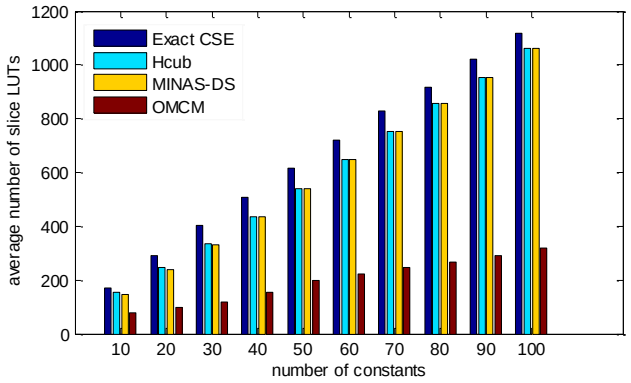

Fig. 9 Resource utilization - slice LUTs

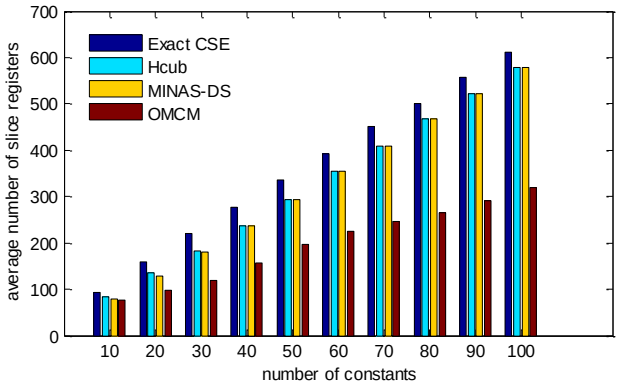

Fig. 10 Resource utilization - slice registers

To summarize the results, the proposed online constant coefficient multipliers are compared in Table 2 with constant coefficient multipliers implemented using Xilinx IP core based designs as well as approximate RoBA multipliers. Online designs are shown to use fewer FPGA resources, reduce online delay and increase clock frequency as shown in Table 2. Online constant coefficient multipliers have the added benefit of reduced interconnection complexity between modules. Online multiple constant multipliers are also compared with existing serial arithmetic constant coefficient multipliers. The reduced complexity of the proposed OMCM operators and the choice of a parameter set that select fundamentals suitable for online arithmetic are shown to result in a reduction in resource requirement as observed from Figs. 9-10.

6. Conclusion

Online arithmetic reduces operator complexity and the interconnections in modules. The shift-and-add algorithm that is commonly used for multiplication by constants requires left shifts. A left shift in online arithmetic makes the system implementation non-causal. A novel adaptation of the shift-and–add algorithm for serial MSDF data using right shifts rather than left shifts is presented in this paper. The outputs obtained are scaled due to the use of right shifts and can be corrected by adjusting the exponents of the FP outputs. The expected bit growth also has to be considered to normalize the output. The MCM problem is NP-complete and therefore realistic constraints have to be imposed on the problem to obtain an optimal solution in a reasonable time. Based on Theorems 3-6, a set of rules for online multiple constant multiplications are proposed that can be used to set constraints on the number of online full adders and online delay. Multiple constant multipliers show improvement in key parameters such as power-delay product and clock frequency and require fewer resources than single constant multipliers and bit-parallel implementations. The proposed online multiple constant coefficient multipliers show significant savings in resource utilization in comparison with digit-serial multipliers and approximate RoBA multipliers. Future work will develop algorithms for synthesis of OMCM constants under various constraints.

References

[1] D. R. Bull and D. H. Horrocks, "Primitive operator digital filters," IEE Proceedings G - Circuits, Devices and Systems, vol. 138, no. 3, pp. 401-412, June 1991.

[2] A. G. Dempster and M. D. Macleod, "Constant integer multiplication using minimum adders," IEE Proceedings - Circuits, Devices and Systems, vol. 141, no. 5, pp. 407-413, October 1994.

[3] Y. Voronenko and M. Püschel, "Multiplierless multiple constant multiplication," ACM Transactions on Algorithms, vol. 3, no. 2, p. 11, May 2007.

[4] L. Aksoy, E. Costa, P. Flores, and J. Monteiro, "Exact and approximate algorithms for the optimization of area and delay in multiple constant multiplications," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 27, no. 6, pp. 1013-1026, June 2008.

[5] L. Aksoy, E. O. Güneş, and P. Flores, "Search algorithms for the multiple constant multiplications problem: exact and approximate," Microprocessors and Microsystems, vol. 34, no. 5, pp. 151-162, August 2010.

[6] L. Aksoy, E. Costa, P. Flores, and J. Monteiro, "Finding the optimal tradeoff between area and delay in multiple constant multiplications," Microprocessors and Microsystems, vol. 35, no. 8, pp. 729-741, November 2011.
[7] L. Aksoy, C. Lazzari, E. Costa, P. Flores, and J. Monteiro, "Optimization of area in digit-serial multiple constant multiplications at gate-level," Proc. IEEE Symp. Circuits and Systems (ISCAS 2011), IEEE Press, May 2011, pp. 2737-2740.
[8] L. Aksoy, C. Lazzari, E. Costa, P. Flores, and J. Monteiro, "High-level algorithms for the optimization of gate-level area in digit-serial multiple constant multiplications," Integration, the VLSI Journal, vol. 45, no. 3, pp. 294-306, June 2012.
[9] K. Johansson, O. Gustafsson, A. Dempster, and L. Wanhammar, "Algorithm to reduce the number of shifts and additions in multiplier blocks using serial arithmetic," Proc. IEEE Mediterranean Electrotechnical Conference, IEEE press, May 2004, pp. 197-200.
[10] K. Johansson, O. Gustafsson, and L. Wanhammar, "Multiple constant multiplication for digit-serial implementation of low power FIR filters," WSEAS Transactions on Circuits and Systems, vol. 5, no. 7, pp. 1001-1008, July 2006.
[11] W. Vanderbauwhede and K. Benkrid, High-performance computing using FPGAs, New York: Springer, 2013.
[12] M. Kumm, K. Liebisch, and P. Zipf, "Reduced complexity single and multiple constant multiplication in floating point precision," Proc. Conf. Field Programmable Logic and Applications (FPL 2012), IEEE press, August 2012, pp. 255-261.
[13] M. Faust and C. H. Chang, "Bit-parallel multiple constant multiplication using look-up tables on FPGA," Proc. IEEE Symp. Circuits and Systems (ISCAS 2011), IEEE press, May 2011, pp. 657-660.
[14] Q Wang, Y Li, B Shao, S Dey, and P Li, "Energy efficient parallel neuromorphic architectures with approximate arithmetic on FPGA, " Neurocomputing, vol. 221, pp. 146-158, January 2017.
[15] R. Zendegani, M. Kamal, M. Bahadori, A. Afzali-Kusha, and M. Pedram, "RoBA multiplier: A rounding-based approximate multiplier for high-speed yet energy-efficient digital signal processing," Transactions on Very Large Scale Integration (VLSI) Systems, vol. 25, no. 2, pp. 393-401, February 2017.
[16] G. B. Joseph and R. Devanathan, "Radix-2^h online floating point multipliers," Proc. IEEE Conf. Dallas Circuits and Systems (DCAS 14), IEEE press, October 2014, pp. 1-4.
[17] J. Olivares, J. Hormigo, J. Villalba, I. Benavides, E. L. Zapata, "SAD computation based on online arithmetic for motion estimation," Microprocessors and Microsystems, vol. 30, no. 5, pp. 250-258, August 2006.
[18] M. Faust and C. H. Chang, "Minimal logic depth adder tree optimization for multiple constant multiplication," Proc. IEEE Symp. Circuits and Systems (ISCAS 2010), IEEE press, May-June 2010, pp. 457-460.