Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2010, Article ID 697625, 17 pages
doi:10.1155/2010/697625

Research Article

Layout Aware Optimization of High Speed Fixed Coefficient FIR Filters for FPGAs

Shahnam Mirzaei,[1] Ryan Kastner,[2] and Anup Hosangadi[3]

[1] *Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106, USA*
[2] *Cadence Design Systems, University of California, San Diego, CA 95134, USA*
[3] *Department of Computer Science and Engineering, Cadence, La Jolla, CA 92093, USA*

Correspondence should be addressed to Shahnam Mirzaei, shahnam@mindspring.com

Received 14 April 2009; Revised 13 November 2009; Accepted 17 January 2010

Academic Editor: Liam Marnane

We present a method for implementing high speed finite impulse response (FIR) filters on field programmable gate arrays (FPGAs). Our algorithm is a multiplierless technique where fixed coefficient multipliers are replaced with a series of add and shift operations. The first phase of our algorithm uses registered adders and hardwired shifts. Here, a modified common subexpression elimination (CSE) algorithm reduces the number of adders while maintaining performance. The second phase optimizes routing delay using prelayout wire length estimation techniques to improve the final placed and routed design. The optimization target platforms are Xilinx Virtex FPGA devices where we compare the implementation results with those produced by Xilinx Coregen, which is based on distributed arithmetic (DA). We observed up to 50% reduction in the number of slices and up to 75% reduction in the number of look up tables (LUTs) for fully parallel implementations compared to DA method. Also, there is 50% reduction in the total dynamic power consumption of the filters. Our designs perform up to 27% faster than the multiply accumulate (MAC) filters implemented by Xilinx Coregen tool using DSP blocks. For placement, there is a saving up to 20% in number of routing channels. This results in lower congestion and up to 8% reduction in average wirelength.

1. Introduction

There has been a tremendous growth for the past few years in the field of embedded systems, especially in the consumer electronics segment. The increasing trend towards high performance and low power systems has forced researchers to come up with innovative design techniques that can achieve these objectives and meet the stringent system requirements. Many of these systems perform some kind of streaming data processing, which requires the extensive evaluation of arithmetic expressions.

FPGAs are being increasingly used for a variety of computationally intensive applications, especially in the realm of digital signal processing (DSP) [1–7]. Due to rapid increases in fabrication technology, the current generation of FPGAs contains a large number of configurable logic blocks (CLBs), and are becoming more feasible for implementing a wide range of arithmetic applications. The high nonrecurring engineering (NRE) costs and long development time for application specific integrated circuits (ASICs) make FPGAs attractive for application specific DSP solutions. Finite impulse response (FIR) filters are prevalent in signal processing applications. These functions are major determinants of the performance and of the device power consumption. Therefore it is important to have good tools to optimize FIR filters. Moreover, the techniques discussed in this paper can be incorporated in building other complex DSP functions, for example, linear systems like FFT, DFT, DHT, and so forth. Most of the DSP design techniques currently in use are targeted towards hardware synthesis, and do not specifically consider the features of the FPGA architecture [8–13]. The previous research primarily concentrates on minimizing multiplier block adder cost. In this paper, we present a method for implementing high speed FIR filters using only registered adders and hardwired shifts. A modified CSE algorithm is extensively used to reduce FPGA hardware. CSE is a compiler optimization that searches for instances of identical expressions (i.e., they all evaluate to the same

value), and analyses whether it is worthwhile replacing them with a single variable holding the computed value. The cost function defined in this modified algorithm explicitly considers the FPGA architecture [14]. This cost function assigns the same weight to both registers and adders in order to balance the usage of such components when targeting FPGA architecture. Common subexpression elimination is an optimization technique that searches for instances of an identical expression in an equation and analyses whether it is worthwhile replacing them with a single variable holding the computed value. This technique is widely used in optimizing compilers. Furthermore, the cost function is modified to consider the mutual contraction metric [15] in an attempt to optimize the physical layout of the FIR filter. It is shown that introducing this metric to the cost function affects the FPGA area.

The major contributions of this paper are as follows.

(1) The development of a novel algorithm (will be referred as modified CSE throughout the paper) for optimizing the fixed coefficient multiplier block for FIR filters for FPGA implementation. The modified CSE algorithm utilizes a modified cost function for common subexpression elimination that explicitly considers the underlying FPGA architecture.

(2) Taking interconnect delay into account which makes physical and logic co-synthesis feasible.

The rest of the paper is organized as follows: Section 2 introduces the related work. In Section 3, popular FIR filter architectures are described. Section 4 describes the optimization algorithm for area minimization using a modified CSE method. The CSE method presented in this section, leverages a novel cost function. This discussion is followed by the interconnection optimization algorithm. In Section 5, the experimental setup, the CAD flow, and the experimental results are presented. We conclude the paper in Section 6.

2. Related Work

Most signal processing and communication applications including FIR filters, audio, video and image processing use some sort of constant multiplication. The generation of a multiplier block from the set of constants is known as the multiple constant multiplication (MCM) problem. Finding the optimal solution, namely, the one with the fewest number of additions and subtractions, is known to be NP-complete [12]. There is a lot of work on deriving efficient structures for constant multiplications [16–21]. All of these techniques are based on computing constant multiplications using lookup tables and additions. The distributed arithmetic (DA) [20, 22, 23] method that is used by Xilinx Coregen is also based on lookup tables. The Xilinx CORE Generator has a highly parameterizable, optimized filter core for implementing digital FIR filters [20] that uses both DA and MAC based architectures. It generates a synthesized core that targets a wide range of Xilinx devices. The MAC based implementations make use of the embedded multipliers/DSP blocks on the FPGA devices.

While there has been a lot of work on optimizing constant multiplications using adders and employing redundancy elimination [8, 9, 24–26], they have not been effectively used for FIR filter design. The closest work to implementing filters with adders is in [27], where FIR filters are implemented using an add and shift method. Authors have used a canonical signed digit (CSD) encoding to discuss how high speed implementations can be achieved by registering each adder. Registering an adder output comes at no extra cost on an FPGA if an unused D flip flop is available at the output of each LUT.

Dempster and Macleod present a method using addition chains to reduce the number of adders in the multiplier block [10, 11, 28]. They have considered the concept of addition chains, and by enumerating all possible adder-chains with four or fewer adders they have found that multiplications by all constants up to 2^{12} can be computed using only four additions.

Gustafsson et al. [11] propose a generalized 5-adder approach, observing that 5 adders are sufficient to compute multiplications by constants with up to 19 bits. Though very expensive, this is a significant result that can be used in the optimal construction of constant multiplications, where the maximum size of the constants is 19 bits.

Gustafsson et al. present two approaches [28]. The first yields optimal results, that is, a minimum number of additions and subtractions, but requires an exhaustive search which significantly increases the running time of the search algorithm. Compared with previous optimal approach [11], redundancies in the exhaustive search cause the search time to be drastically decreased. The second is a heuristic approach based on signed-digit representation and subexpression sharing. The results for the heuristic are not optimal in very few cases. However, the optimal approach results in several solutions. It is possible to pick the best one according to the criteria such as relations between the number of adders, possible coefficients, and number of cascaded adders, and so forth.

In comparison with the other algorithms for common subexpression elimination [8, 9, 24, 25, 29], our method considers the structure of the FPGA slices (see Figure 5) and takes into account the cost of adders and registers when performing the optimization. Furthermore, we provide comprehensive evidence of the benefits of our technique through experimental results compared with those produced by industry standard tools.

Meyer-Baese et al. [30] present reduced adder graph (RAG-n) multiplierless filter design method which is an adder based optimization algorithm with good implementation results. These results are compared with DA where it achieves the average size reduction of 71%, cost improvement of 56% expressed as LEs/Fmax (Altera logic element/Maximum frequency) at the cost of drop in performance for 8% compared to DA. The paper specifically mentions that RAG-n works best when many small coefficients are available, while DA offers greater advantage when there are many large coefficients. Our method always shows better performance and area compared to DA regardless of coefficient sizes.

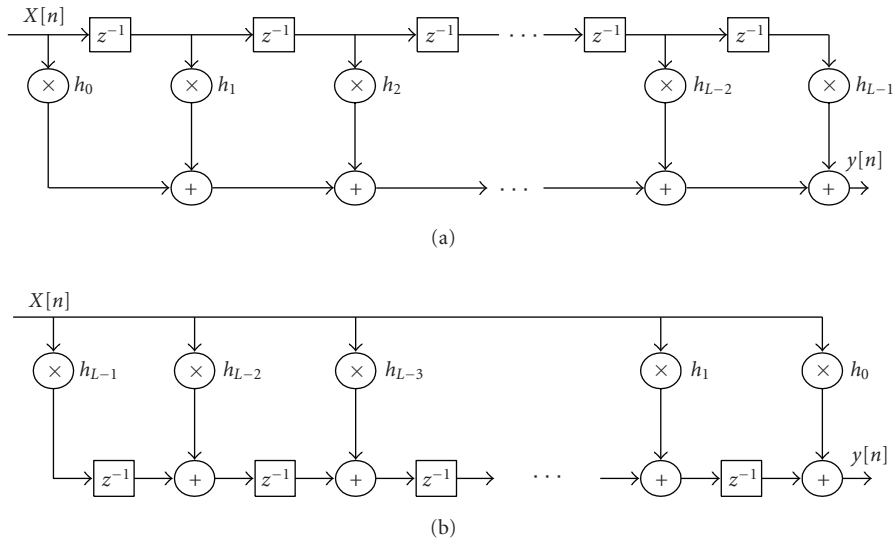

Figure 1: Mathematically identical MAC FIR filter structures: (a) The direct form of a finite impulse response (FIR) filter (b) The transposed direct form of an FIR filter.

Macpherson and Stewart [31] have introduced RSG (Reduced Slice Graph) algorithm as modified version of RAG-*n*. The results presented establish a clear area advantage of RSG over RAG-*n*. The classic research optimization metrics (such as RAG-*n*) of minimizing multiplier block adder cost has been demonstrated not to minimize FPGA hardware for full-parallel pipelined FIR filters. Reducing flip-flop count through minimizing multiplier logic depth has instead been shown to yield the lowest area solutions. Authors implemented two metric levels (primary and secondary) to allow one "best graph" to be selected. For each integer, the primary metric selects the subset of graphs with minimum logic depth and from that subset, the secondary metric selects the graph that minimizes bit-widths. Also, rather than starting with the lowest cost coefficients as RAG-*n* does, RSG takes the opposite approach and starts with the highest cost values and simply inserts the "best graph" required for each, ensuring no duplicate adders are created and that adder outputs are shared as far as possible.

SPIRAL [12] (also called Hcub) is an automatic code generation tool for DSP transforms. The code generated by SPIRAL can be used to generate FIR filters. It converts the series of multiplications by constants into minimum number of additions and shifts. Connecting this multiple constant multiplier block to a tapped delay line creates a FIR filter (see Figure 2). For the purpose of comparison, we used SPIRAL to generate FIR filters. It is important to understand that even though SPIRAL is optimal in terms of number of additions, it does not necessarily create the most efficient FPGA implementation since it does not explicitly consider the features of the FPGA architecture. As shown in Section 5.2, SPIRAL leads to low FPGA area/resource usage though relatively low multiplier/FIR filter performance. The main reason is that the multiplier block is not pipelined and depending on the coefficients used, the cascaded adder tree could synthesize to several levels of logic and consequently results in low performance. This is a good solution for software implementation but not necessarily for FPGA implementation.

3. Filter Architecture

In this section, a review of FIR filter architecture is presented. This is followed by the illustration of two major implementations of FIR filters that are widely used: MAC and DA methods.

Equation (1) describes an output of L tap FIR filter, which is the convolution of the latest L input samples. L is the number of coefficients of the filter impulse response $h[k]$, and $x[n]$ represents the input time series [32]:

$$y[n] = \sum_{k=0}^{L-1} h[k] \cdot x[n-k]. \quad (1)$$

The conventional tapped delay line realization of this inner product is shown in Figure 1 [31]. Figure 1(a) shows the direct implementation of (1). The transposed direct form of this filter is shown in Figure 1(b), which is obtained from the direct form by moving the registers outside the multiplier block. This implementation requires L multiplications and $L-1$ additions per sample. This can be implemented using a single MAC engine, but it would require L MAC operations before the next input sample can be processed. This serial implementation reduces the performance of the design significantly. Using a parallel implementation with L MACs increases the performance by a factor of L.

Most FPGAs include embedded multipliers/DSP blocks to handle these multiplications. For example, Xilinx Virtex II/Pro provides embedded multipliers while more recent FPGA families such as Virtex 4/5 devices offer embedded DSP blocks. In either case, there are two major limitations. First, the multipliers or DSP blocks can accept inputs with limited bit width, for example, 18 bits for Virtex 4 devices. A Virtex 5 device provides additional precision of 25 bit input

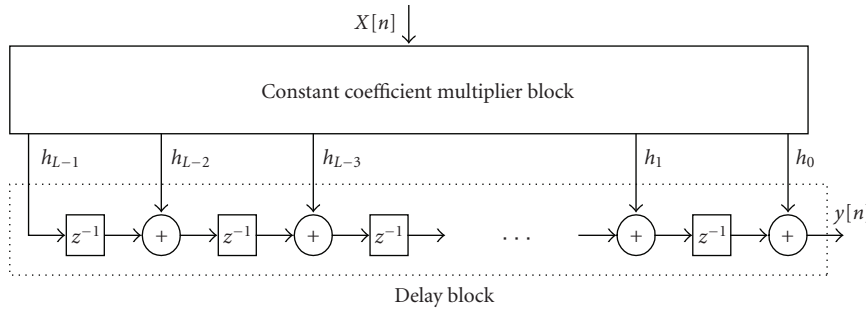

Figure 2: Constant multipliers of Figure 1(b) replaced by constant coefficient multiplier block.

for one of the operands. In the case of higher input width, Xilinx Coregen tools combines these blocks with CLB logic [33]. Experimental results show in most cases performance advantage compared to embedded multipliers/DSP blocks. Secondly, the number of these blocks is limited on each device. There are several applications such as data acquisition systems, or equalizers [13] that require long FIR filters with high number of taps that might be difficult (if not impossible) to implement using these embedded resources.

Since many FIR filters use constant coefficients, the full flexibility of a general purpose multiplier is not required, and the area can be reduced using techniques developed for constant multiplication [16–21]. A popular technique for implementing the transposed direct form of FIR filters is the use of a multiplier block instead of using multipliers for each constant (see Figure 2) [31]. The multiplications with the set of constants $\{h_k\}$ are replaced by an optimized set of additions and shift operations. Finding and factoring common subexpressions can further optimize the expressions. The performance of this filter architecture is limited by the latency of the largest adder.

An alternative to the MAC approach is DA which is a well known method to save resources and was developed in the late 1960's independently by Croisíer et al. [34] and Zohar [35]. The term "distributed arithmetic" is derived from the fact that the arithmetic operations are not easily apparent and often *distributed* across the terms. This can be verified by looking at (5) which is a rearranged from of (4). DA is a bit-level rearrangement of constant multiplication, which replaces multiplication with a high number of lookup tables and a scaling accumulator. Using a DA method, the filter can be implemented either in bit serial or fully parallel mode to tradeoff between bandwidth and area utilization. In essence, this replicates the lookup tables, allowing for parallel lookups. Therefore, the multiplication of multiple bits is performed at the same time.

Assuming coefficients $c[n]$ are known constants, and $x[n]$ is the input data, equation (1) can be rewritten as follows [32]:

$$y[n] = \sum_{n=0}^{N-1} c[n] \cdot x[n]. \tag{2}$$

Variable $x[n]$ can be represented by [32]

$$x[n] = \sum_{b=0}^{B-1} x_b[n] \cdot 2^b \qquad x_b[n] \in [0,1], \tag{3}$$

where $x_b[n]$ is the bth bit of $x[n]$ and B is the input width. Finally, the inner product can be rewritten as follows [32]:

$$\begin{aligned} y &= \sum_{n=0}^{N-1} c[n] \sum_{b=0}^{B-1} x_b[n] \cdot 2^b \\ &= c[0]\left(x_{B-1}\,[0]2^{B-1} + x_{B-2}\,[0]2^{B-2} + \cdots + x_0\,[0]2^0\right) \\ &\quad + c[1]\left(x_{B-1}[1]2^{B-1} + x_{B-2}\,[1]2^{B-2} + \cdots + x_0\,[1]2^0\right) \\ &\quad + \cdots + c[N-1]\left(x_{B-1}\,[N-1]2^{B-1} + x_{B-2}\,[0]2^{B-2}\right. \\ &\quad \left. + \cdots + x_0\,[N-1]2^0\right). \end{aligned} \tag{4}$$

In this case, each summation involves all bits from one variable. Each line computes the product of one of the constants multiplied by one of the input variables and then sums each of these results. Therefore, there are N summation lines, one for each of the constants $c[n]$. Equation (4) can be rearranged as follows [32]:

$$\begin{aligned} y &= (c[0]x_{B-1}[0] + c[1]x_{B-1}\,[1] + \cdots + c[N-1]x_{B-1} \\ &\quad \times[N-1])2^{B-1} + (c[0]x_{B-2}\,[0] + c[1]x_{B-2}\,[1] \\ &\quad + \cdots + c[N-1]x_{B-2}\,[N-1])2^{B-2} + \cdots + (c[0] \\ &\quad \times x_0[0] + c[1]x_0[1] + \ldots + c[N-1]x_0[N-1])2^0 \\ &= \sum_{b=0}^{B-1} 2^b \sum_{n=0}^{N-1} c[n] \cdot x_b[n]. \end{aligned} \tag{5}$$

This is the DA form of the inner product of equation (1). The key insight in this computation is that (5) consists of binary constants of the form of power of 2. This allows for the precomputation of all these values, storing them in a lookup table, and using the individual inputs x_i as an address into the lookup table. Here, each line calculates the final product by using one bit (of the weight) from all input values. This effectively replaces the constant multiplication with a lookup table. Then the computation corresponding to each line of the Equation (5) is performed by addressing the lookup table with the appropriate values as dictated by the individual input variables. Each line is computed serially

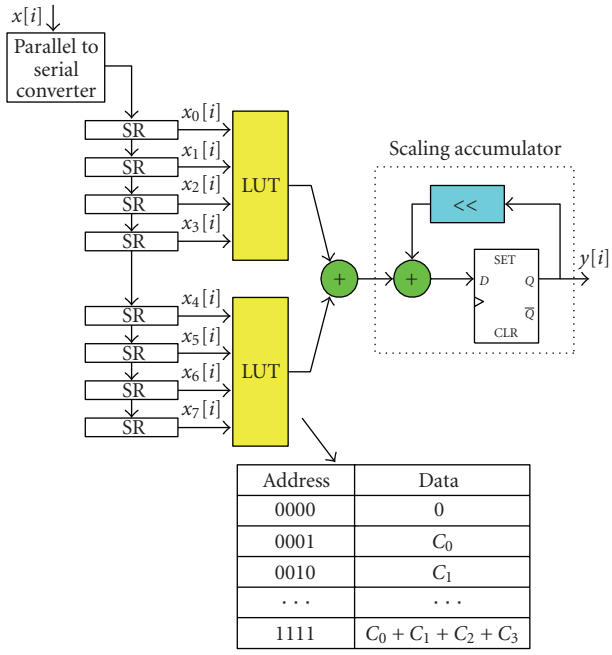

Figure 3: A serial DA FIR filter block diagram.

and the outputs are shifted by the appropriate amounts (i.e., 0, 1, 2,..., B−1 bits). Figure 3 presents a visual depiction of the DA version of inner product computation [23, 36]. The input sequence is fed into the shift register at the input sample rate. The serial output is presented to the RAM based shift registers at the bit clock rate which is B + 1 times (n is number of bits in a data input sample) the sample rate. The RAM based shift register stores the data in a particular address. The outputs of registered LUTs are added and loaded to the scaling accumulator from LSB to MSB, and the result is accumulated over time. For an n bit input, $n+1$ clock cycles are needed for a symmetrical filter to generate the output.

In a conventional MAC, with a limited number of MAC blocks, the system sample rate decreases as the filter length increases due to the increasing bit width of the adders and multipliers and consequently the increasing critical path delay. However, this is not the case with serial DA architectures since the filter sample rate is decoupled from the filter length. As the filter length is increased, the throughput is maintained but more logic resources are required. While the serial DA architecture is efficient by construction, its performance is limited by the fact that the next input sample can be processed only after every bit of the current input sample is processed. Each bit of the current input sample takes one clock cycle to process.

As an example, if the input bit width is 12, a new input can be sampled every 12 clock cycles. The performance of the circuit can be improved by using a parallel architecture that processes the data bits in groups. Figure 4 shows the block diagram of a 2 bit parallel DA FIR filter [23, 36]. The tradeoff here is between performance and area since increasing the number of bits sampled has a significant effect on resource utilization on the FPGA. For instance, doubling the number of bits sampled doubles the throughput and results in half the number of clock cycles. This change doubles the number of LUTs as well as the size of the scaling accumulator. The number of bits being processed can be increased to its maximum size which is the input length n. This gives the maximum throughput to the filter. For a fully parallel DA filter (PDA), the number of LUTs required would be enormous since by adding each bit, the number of LUTs is doubled.

A transposed direct form FIR filter as shown in Figure 1 consists of input/output ports, coefficients memory, delay units, and MAC units. The whole design is partitioned into two major blocks: the multiplier block and the delay block as illustrated in Figure 2. In the multiplier block, each input data sample $x[n]$, does not change until it is multiplied by all the coefficients to generate the y_i outputs. These y_i outputs are then delayed and added in the delay block to produce the filter output $y[n]$. The delay block consists of registers to store the intermediate results. The delay block design is straightforward and can not be optimized further. Therefore we focus our attention on the multiplier block. The constant multiplications are decomposed into hardwired shifts and registered additions. Assuming hardwire shifts are free, the additions can be performed using two input adders, which are arranged in the fastest adder tree structure. Also, due to using registered adders, the performance of the filter is only limited by the slowest adder. Registered adders come at the same cost of non-registered adders in FPGAs. This is due to the fact that each FPGA logic cell consists of a LUT and a register. Our add and shift method takes advantage of registered adders depicted in Figure 5 and inserts registers whenever possible (utilizing unused resources on the FPGA) to improve performance. Due to this fact, we show competitive performance for all size filters comparable with SPIRAL even though designs are not optimized for performance.

Our research considers two aspects of the FIR design: First, it presents the development of a novel algorithm for optimizing the multiplier block for FIR filters, using a modified algorithm for common subexpression elimination. Here, our goal is to produce a design that can provide the maximum sample rate with the least amount of hardware. Our algorithm takes into account the specific features of FPGA architecture to reduce the total number of occupied slices. The reduced number of slices also leads to a reduction in the total power on the FPGA. The implementation results in terms of FPGA area (registers, LUTs, slices), performance, and power consumption are compared against the industry standard Xilinx Coregen as well as SPIRAL [12] automatic software. Secondly, this paper presents an algorithm optimization that takes interconnect into account. The modified CSE algorithm generates the optimized solution for placement, routing, and power consumption. Here the prelayout wire length estimation techniques are incorporated in early stages of the design to improve the final placed and routed design by reducing FPGA hardware (registers, LUTs, slices), the routing congestion, and latency. In order to make this happen, the novel cost function in the first step is further modified to reduce the total wirelength of the routed design.

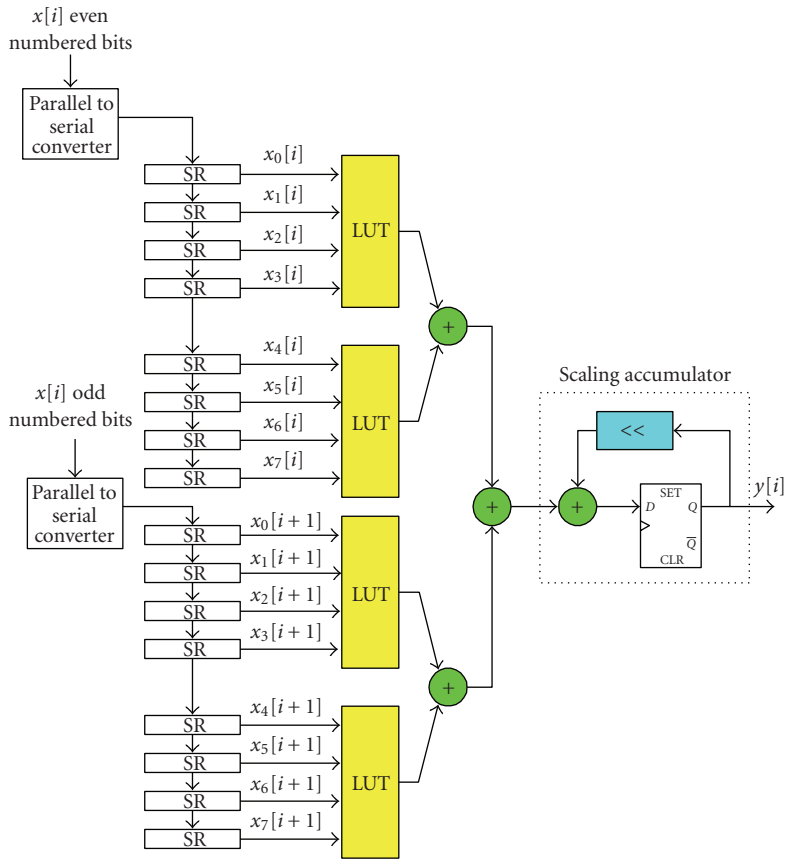

Figure 4: A 2 bit parallel DA FIR filter block diagram.

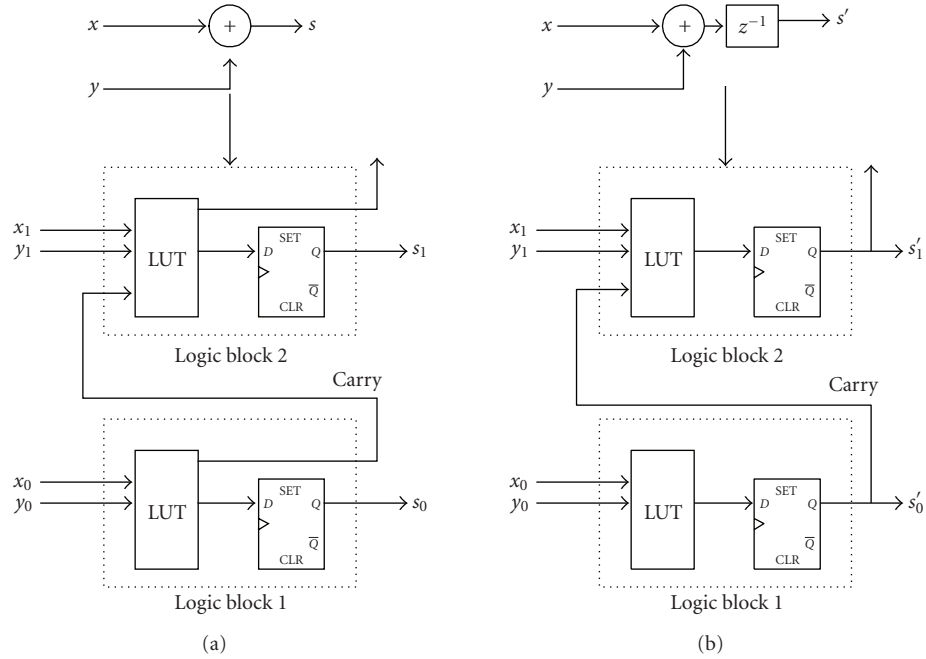

Figure 5: (a) Nonregistered output adder used by DA or other competing algorithms that do not take FPGA architecture into account. (b) Registered output adder used in add and shift method leveraging the new cost function that takes FPGA architecture into account.

4. Optimization Algorithm (Modified CSE Algorithm)

The goal of our optimization is to reduce the area of the multiplier block by minimizing the number of adders and any additional registers required for the fastest implementation of the FIR filter. In the following, a brief overview of the common subexpression elimination methods is presented in Section 4.1 with a detailed description in [14]. We then present two optimization algorithms. First, the area optimization algorithm presented in Section 4.2 which focuses on minimizing the FPGA area taking FPGA architecture into account. This is followed by a brief discussion on algorithm complexity in Section 4.2.1. Second, the interconnect optimization algorithm that focuses on minimizing the total wirelength and number of routing channels is presented in Section 4.3.

4.1. Overview of Common Subexpression Elimination. An occurrence of an expression in a program is a common subexpression if there is another occurrence of the expression whose evaluation always precedes this one in execution order and if the operands of the expression remain unchanged between the two evaluations. The CSE algorithm essentially keeps track of available expressions block (AEB) that is, those expressions that have been computed so far in the block and have not had an operand subsequently changed. The algorithm then iterates, adding entries to and removing them from the AEB as appropriate. The iteration stops when there can be no more common subexpressions detected.

The CSE algorithm uses a polynomial transformation to model the constant multiplications. Given a representation for the constant C, and the variable X, the multiplication $C * X$ can be represented as a summation of terms denoting the decomposition of the multiplication into shifts and additions as [37]

$$C * X = \sum_i \pm\left(XL^i\right). \tag{6}$$

The terms can be either positive or negative when the constants are represented using signed digit representations such as the CSD representation. The exponent of L represents the magnitude of the left shift and i represents the digit positions of the non-zero digits of the constants. For example the multiplication $7 * X = (100-1)_{\text{CSD}} * X = X \ll 3 - X = XL^3 - X$, using the polynomial transformation.

We use the *divisors* to represent all possible common subexpressions. A divisor of a polynomial expression is a set of two terms obtained after dividing any two terms of the expression by their least exponent of L. This is equivalent to factoring by the common shift between the two terms. Divisors are obtained from an expression by looking at every pair of terms in the expression and dividing the terms by the minimum exponent of L. For example, in the expression:

$$F = XL^2 + XL^3 + XL^5, \tag{7}$$

consider the pair of terms:

$$+XL^2 + XL^3. \tag{8}$$

The minimum exponent of L in the two terms is L^2. Dividing by L^2, the divisor:

$$X + XL \tag{9}$$

is obtained. From the other two pairs of terms:

$$XL^2 + XL^5, \quad XL^3 + XL^5 \tag{10}$$

we get the divisors

$$X + XL^3, \quad X + XL^2, \tag{11}$$

respectively. These divisors are significant, because every common subexpression in the set of expressions can be detected by performing intersections among the set of divisors.

4.2. Area Optimization Algorithm. Common subexpression elimination is used extensively to reduce the number of adders, which leads to a reduction in the area. Additional registers will be inserted, wherever necessary, to synchronize all the intermediate values in the computations. Performing common subexpression elimination can sometimes increase the number of registers substantially, and the overall area could possibly increase. Consider the two expressions F_1 and F_2 which could be part of the multiplier block:

$$\begin{aligned} F_1 &= \text{A} + \text{B} + \text{C} + \text{D}, \\ F_2 &= \text{A} + \text{B} + \text{C} + \text{E}. \end{aligned} \tag{12}$$

Figure 6(a) shows the original unoptimized expression trees. Both expressions have a minimum critical path of two addition cycles. These expressions require a total of six registered adders for the fastest implementation. Now consider the selection of the divisor $d_1 = (\text{A} + \text{B})$. This divisor saves one addition and does not increase the number of registers. Divisors (A + C) and (B + C) also have the same value, assuming (A + B) is selected randomly. The expressions are now rewritten as

$$\begin{aligned} d_1 &= (\text{A} + \text{B}), \\ F_1 &= d_1 + \text{C} + \text{D}, \\ F_2 &= d_1 + \text{C} + \text{E}. \end{aligned} \tag{13}$$

After rewriting the expressions and forming new divisors, the divisor $d_2 = (d_1 + \text{C})$ is considered. This divisor saves one adder, but introduces five additional registers, as can be seen in Figure 6(b). Two additional registers should be used on both D and E signals in order to synchronize them with the partial sum expression (A + B + C), such that new values for A, B, C, D and E can be read on each clock cycle. Therefore this divisor has a value of −4. A more careful subexpression elimination algorithm would only extract the common subexpression A + B (or A + C or B + C). This decreases the number of adders by one from the original, and no additional registers are required. No other valuable divisors can be found and the algorithm stops. We end up with the expressions shown in Figure 6(c).

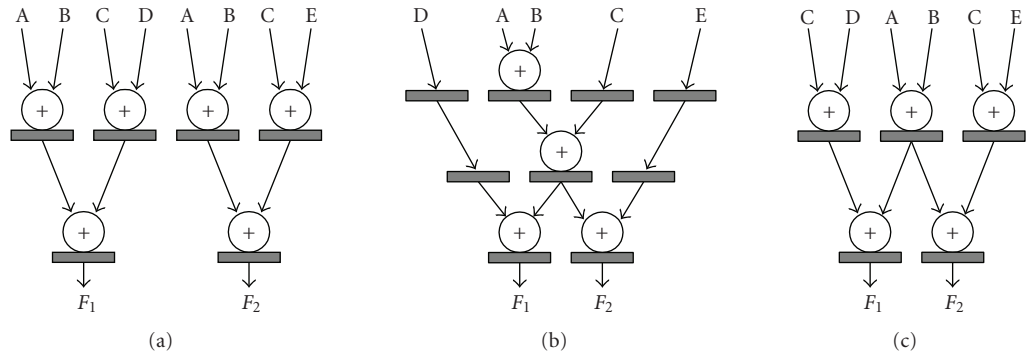

Figure 6: Extracting common subexpression (a) Unoptimized expression trees. (b) Extracting common expression (A + B + C) results in higher cost due to inserting additional synchronizing registers. (c) A more careful extraction of common subexpression (A + B) applied by our modified CSE algorithm results in lower cost.

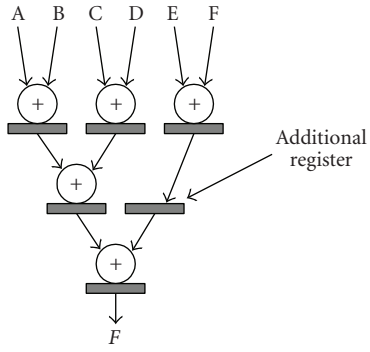

Figure 7: The fastest possible tree is formed and a synchronizing register is inserted, such that new values for the inputs can be read in every clock cycle.

FPGAs have a fixed architecture where every slice contains a LUT/flip flop pair. If either the LUT or flip flop are unused, then FPGA resource usage efficiency is reduced. For example, the structure shown in Figure 6(b) occupies more area than the one shown in Figure 6(a) in FPGA implementation even though it has fewer number of adders. The reason is that storage elements inside slices are used while the LUTs have not been utilized for the related logic. In this implementation, the slice utilization efficiency is reduced where only one of the register or LUT is used. The extraction of common subexpression shown in Figure 6(c) helps the simultaneous use of storage elements and LUTs and therefore more efficient use of FPGA area.

Another important factor is minimizing the number of registers required for our design. This can be done by arranging the original expressions in the fastest possible tree structure, and then inserting registers. For example, for the six term expression $F = A + B + C + D + E + F$, the fastest tree structure can be formed with three addition steps, which requires one register to synchronize the intermediate values, such that new values for A, B, C, D, E and F can be read in every clock cycle. This is illustrated in Figure 7.

The first step of the modified CSE algorithm is to generate all the divisors for the set of expressions describing the multiplier block. The next step is to use our iterative algorithm where the divisor with the greatest value is extracted. To calculate the value of the divisor, we assume that the cost of a registered adder and a register is the same. The value of a divisor is the same as the number of additions saved by extracting it minus the number of registers that have to be added. After selecting the best divisor, the common subexpressions can be extracted. We then generate new divisors from the new terms that have been generated due to rewriting, and add them to the dynamic list of divisors. The modified CSE algorithm halts when there is no valuable divisor remaining in the set of divisors. Algorithm 1 summarizes all the steps mentioned above as our optimized algorithm.

The modified CSE algorithm presented here is a greedy heuristic algorithm. In this algorithm for the extraction of arithmetic expressions, the divisor that obtains the greatest savings in the number of additions is selected at each step. To the best of our knowledge, there has been no previous work done for finding an optimal solution for the general common subexpression elimination problem, though recently there has been an approach for solving a restricted version of the problem using Integer Linear Programming (ILP) [38]. The ILP problem is formulated as a Boolean network that covers all possible partial terms. The inputs to this network are shifted versions of the value that serves as input to the multiplier block. Each adder and subtracter used to generate a given partial term is represented as an AND gate. All partial terms that represent the same numerical value are ORed together. There is a single output which is an AND over all the coefficients in the multiplier block. We cast this problem into a 0-1 Integer Linear Programming Authors direct this problem into a 0-1 Integer Linear Programming (ILP) problem by requiring: that the output is asserted, meaning that all coefficients are covered by the set of partial terms found; while minimizing the total number of AND gates that evaluate to one, that is, the number of adders/subtracters.

4.2.1. Complexity Analysis. The area optimization algorithm presented in Algorithm 1 spends most of its time in the first step on creating distinct divisors list and their associated frequencies. The second step of the algorithm is linear due

```
ReduceArea( {Pi} )
{
{Pi} = Set of expressions in polynomial form;
{D} = Set of divisors = φ;

//Step 1: Creating divisors and calculating minimum
number of registers required

for each expression Pi in {Pi}
{
{Dnew} = FindDivisors(Pi);
Update frequency statistics of divisors in {D};
{D} = {D} ∪ {Dnew};
Pi->MinRegisters = Calculate Minimum registers required
for fastest evaluation of Pi;
}

//Step 2: Iterative selection and elimination of best divisor
while(1)
{

Find d = Divisor in {D} with greatest Value;
// Value = Num Additions reduced – Num Registers Added;

if( d == NULL) break;
Rewrite affected expressions in {Pi} using d;

Remove divisors in {D} that have become invalid;
Update frequency statistics of affected divisors;

{Dnew} = Set of new divisors from new terms added
   by division;
   {D} = {D} ∪ {Dnew};
}
}
```

Algorithm 1: Modified CSE algorithm to reduce area: The divisors are generated for a set of expressions and the one with the greatest value is extracted. Then the common subexpressions can be extracted and a new list of terms is generated. The iterative algorithm continues with generating new divisors from the new terms, and add them to the dynamic list of divisors. The algorithm stops when there is no valuable divisor remaining in the set of divisors.

to the dynamic management of the set of divisors. The worst case complexity of the first step for a M element constant matrix occurs when all the N digits of each constant are non-zero resulting in $M * N$ terms. Since the number of divisors is quadratic in the number of terms, the total number of divisors generated for the series would be of $O(M^2 * N^2)$. This represents the upper bound on the total number of distinct divisors in $\{D\}$. Assume that the data structure for $\{D\}$ is such that it takes constant time to search for a divisor with given variables and exponents of L. Each time a set of divisors $\{D_{\text{new}}\}$ which has a maximum size of $O(M^2 * N^2)$ is generated in step 1, it takes $O(M^2 * N^2)$ to compute the frequency statistics with the set $\{D\}$.

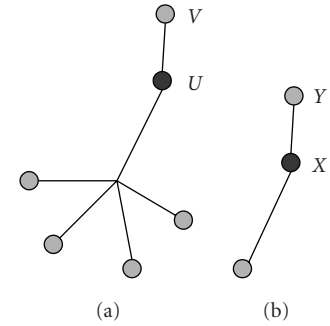

Figure 8: Multipin net (a) versus two pin net (b) [15]. Placement tools do not treat these two nets the same way causing small fan-out nets having stronger contraction compared to larger fan-out ones which results in the connection (U, V) to be shorter than connection (X, Y).

4.3. Interconnect Optimization Algorithm. Interconnect delay is the dominant factor in the overall performance of modern FPGAs. Pre-layout wire length estimation techniques can help in early optimizations and improve the final placed and routed design. Our modified CSE algorithm (see Algorithm 1) does not take interconnection into account, which can lead to sub-optimal final design. The goal is to improve our cost function for reduction in congestion, routability and latency.

We propose a metric to evaluate the proximity of elements connected in a netlist. This metric is capable of predicting short connections more accurately and deciding which groups of nodes should be clustered to achieve good placement results. Here, divisors are referred as nodes. In other words, we are trying to find the common subexpression that not only eliminates computation, but also results in to the best placement and routing. This metric is embedded into our cost function and various design scenarios are considered based on maximizing or minimizing the modified cost function on total wirelength and placement. Experiments show that taking physical synthesis into account can produce better results.

The first step to produce more efficient layout is to predict physical characteristics from the netlist structure. To achieve this, the focus will be on pre-layout wire length and congestion estimations using mutual contraction metric [15]. Consider two nodes U and X and their neighbors in Figure 8. Node U is connected to a multi-pin net whereas node X is connected to a two pin net. Placement tools do not treat these two nets the same way [15]. As a matter of fact, place and route tools put more optimization effort on small fan-out nets trying to shorten their length. Therefore, small fan-out nets have stronger contraction compared to larger fan-out ones. Eventually this causes the connection (U, V) to be shorter than connection (X, Y).

The contraction measure for groups of nodes quantifies how strongly those nodes are connected to each other.

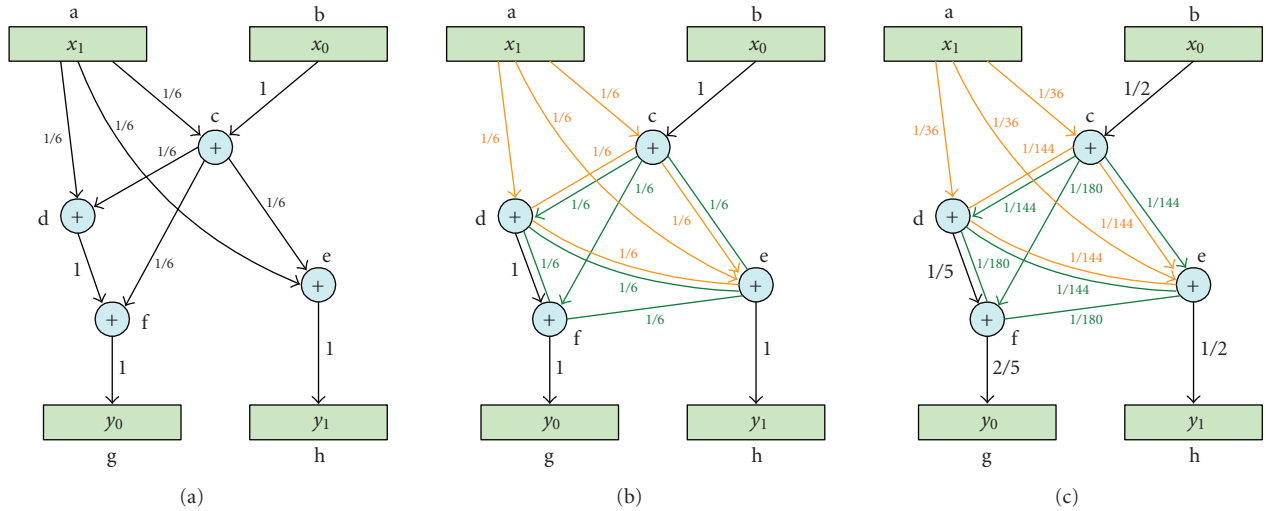

Figure 9: Calculating the edge weights according to modified CSE algorithm: (a) Divisors that are used multiple times are shown as multi-terminal nets with edge weights based on equation (14). (b) A clique is formed with recalculated weights using equation (16). (c) Final edge weights are calculated using mutual contraction using (16).

A group of nodes are strongly contracted if they share many small fan-out nets. In general a strong contraction means shorter length of connecting wires in placed design. Connectivity [39] and edge separability [40] are two other popular measures to estimate the optimized wire length for a placed design. However these measures do not reflect the different behavior of the placement tool towards the multi pin nets versus two pin nets.

In order to include mutual contraction in wire length prediction, a clique has to be formed for multi-pin nets. Given a graph with nodes N, a clique C is a subset of N where every node in C is directly connected to every other node in C (i.e., C is totally connected). Then a weight is defined for each edge of the clique, formed by the multi-pin net, according to (14) [15]:

$$w'(e) = \frac{2}{(d(i)-1) * d(i)}, \tag{14}$$

where $d(i)$, the degree of the edge i, is the number of nodes incident to i. A node incident to a net i of degree d has d−1 edges of weight $w'(e)$ connecting to the other nodes in i [15]. In Figure 8, node u connects to four neighbor nodes through a 5-pin net. So each connection of node u has a weight of $2/((5-1) * 5) = 0.1$ for total weight of 0.4 incident to u. The above equation states that a net with higher degree contributes less weight to its connected nodes. The relative weight of connection incident to nodes is defined by equation (15) [15] as follows:

$$w_r(u,v) = \frac{w'(u,v)}{\sum_x w'(u,x)}, \tag{15}$$

where $w'(u,x)$ is the summation on all nodes x adjacent to u. For example, for Figure 8, $w_r(u,v) = 1/(1+0.4) = 0.71$ and $w_r(x,y) = 1/(1+1) = 0.5$ which means connection (u,v) plays a bigger role in placement of node u than connection (x,y) does for node x. This suggests that mutual connectivity relationship among nodes plays an important role in predicting their relative placement and consequently optimizing the overall wirelength.

A more precise metric for mutual contraction is used, which is the product of the two relative weights to measure the contraction of the connection as in (16) [15]:

$$c_p(x,y) = w_r(x,y) * w_r\ (y,x). \tag{16}$$

This concept can be extended to measure the contraction of a node group. The original cost function using CSE method presented in Section 4.2 considers only area reduction as a constraint which is based on extracting the divisors in a polynomial. The new implementation incorporates the mutual contraction metric into modified CSE algorithm to predict wirelength during the optimization process to see if it is more efficient in terms of routing or congestion. This can be clarified by using an example.

Consider the circuit in Figure 9(a). Each divisor is used multiple times so it creates multi-terminal net. These divisors can be considered as nodes with multi-pin nets. For instance, node c has a 3 pin net, and the new edge weight will be as follows based on (14):

$$w'(e) = 2/(4 * 3) = 1/6. \tag{17}$$

In Figure 9(b), a clique is formed with new weights by using (15) and finally mutual contraction values are calculated and shown in Figure 9(c) using (16). This can be generalized to define the cost function for our FIR filter that considers the mutual contraction metric.

The cost function presented in the previous section considers only area reduction as a constraint. This cost function can be modified according to mutual contraction concept. We have defined different cost functions based on

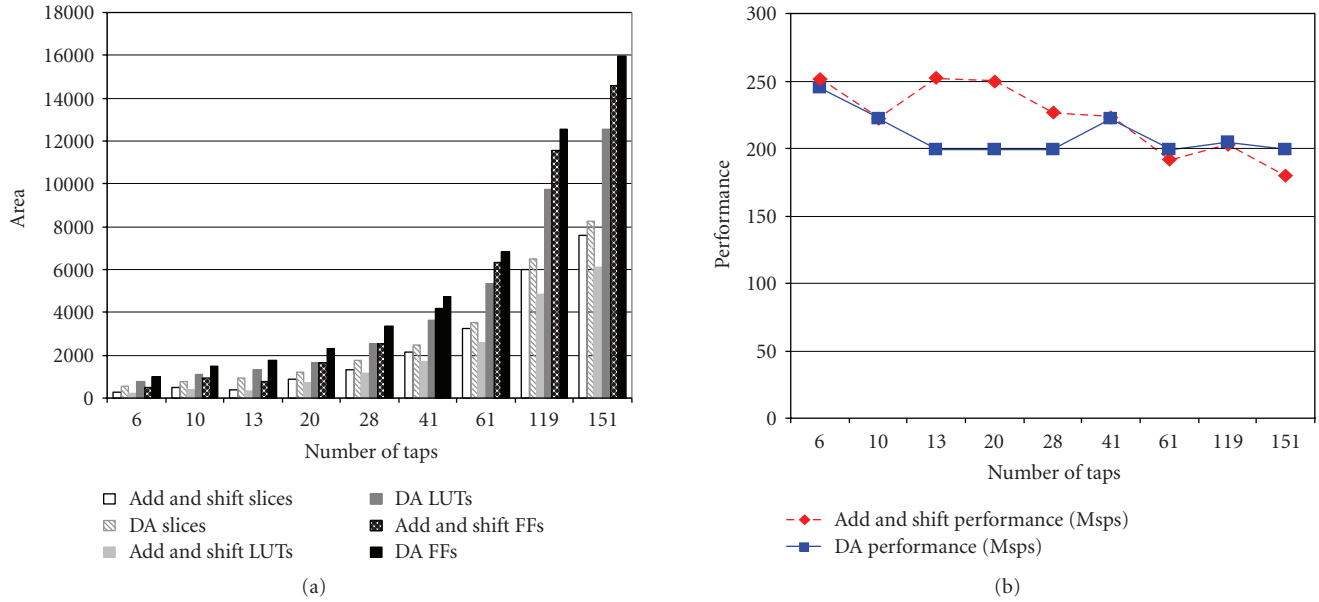

Figure 10: (a) Resource utilization in terms of # of slices, flip flops, and LUTs for various filters using add and shift method (this paper). (b) Performance implementation results (Msps) for various filters using add and shift method (this paper) versus parallel distributed arithmetic.

maximizing or minimizing the average mutual contraction (AMC):

(1) f_x: picks the divisor with maximum saving in number of addition. f_x is the area optimization algorithm presented in Algorithm 1 in Section 4.2 which is our reference modified CSE algorithm. The following algorithms will be compared against f_x.

(2) f_{xMax}: collects the divisors that save maximum number of additions and picks the divisor that produces the maximum AMC among all these divisors. This algorithm largely behaves like f_x when selecting among multiple divisors that all reduce the same number of adders; it picks the divisor that maximizes the AMC while f_x essentially picks a random divisor.

(3) f_{xMin}: collects all the divisors that save the maximum number of additions and picks the divisor that produces the minimum AMC among all these divisors. It is similar to f_{xmax}, but breaks the tie amongst divisors by selecting the divisor that minimizes the AMC.

(4) *Max:* selects the divisor that produces the maximum AMC among all the divisors. This algorithm picks the divisors that maximize the AMC regardless of saving number of additions.

(5) *Min:* selects the divisor that produces the minimum AMC among all the divisors. This algorithm picks the divisors that minimize the AMC regardless of saving number of additions.

5. Experiments

In the following we compare our results with other architectures for both area and performance. Add and shift method results are compared with the Coregen DA approach and SPIRAL software developed by Carnegie Mellon University in Sections 5.1 and 5.2 respectively. In Section 5.3, the results are discussed after applying our interconnect optimization algorithm to the add and shift method.

5.1. FIR Implementation Using Add and Shift Method versus Distributed Arithmetic Method. The main goal of our experiments is to compare the number of resources consumed by the add and shift method with that produced by the cores generated by the commercial Coregen tool based on DA. We compared the power consumption of the two implementations, and also measured the performance. The results use 9 FIR filters of various sizes (6, 10, 13, 20, 28, 41, 61, 119 and 151 tap filters). The target platform for experiments is Xilinx Virtex II device. The constants were normalized to 17 digit of precision and the input samples were assumed to be 12 bits wide. For the add and shift method, all the constant multiplications are decomposed into additions and shifts and further optimized using the modified CSE algorithm explained in Section 4.2. We used the Xilinx Integrated Software Environment (ISE) for synthesis and implementation of the designs. All the designs were synthesized for maximum performance.

Figure 10 shows the resource utilization in terms of the number of slices, flip flops, and LUTs and performance in millions of samples per second (Msps) for the various filters implemented using the add and shift method versus parallel distributed arithmetic (PDA) method implemented by Xilinx Coregen. DA performs computation based on lookup table. Therefore, for a set of fixed size and number of coefficients the area/delay of DA will always be the same (even if the values of the coefficients differ). Our method exploits similarities between the coefficients. This allows us to reduce the area by finding redundant computations.

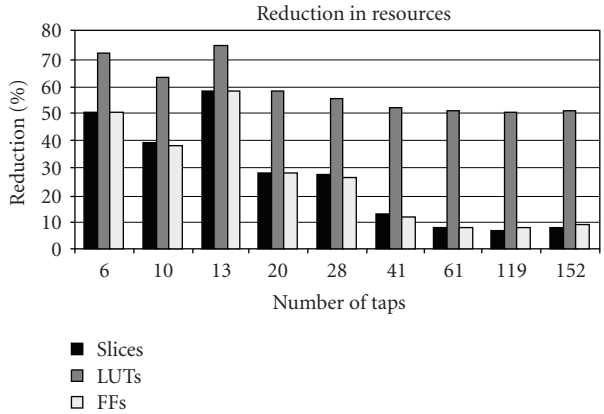

Figure 11: Reduction in resources for add and shift method (this paper) relative to that for DA showing an average reduction of 58.7% in the number of LUTs, and 25% reduction in the number of slices and FFs.

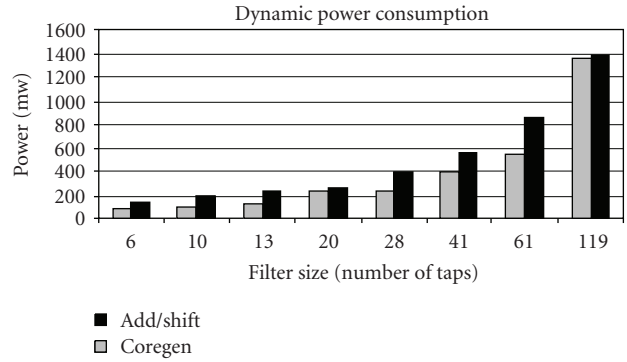

Figure 12: Comparison of power consumption for add and shift (this paper) relative to that for the DA showing up to 50% reduction in dynamic power consumption.

In Figure 10(b), it can be seen that for the cases with roughly the same area, the performance is almost the same. This is shown for filter sizes of 6, 10, 41, 61, and 119. There is a DA performance is 20% less for 13 and 20 tap filter and 10% more for 151 tap filter. In general, performance is inversely proportional to the area. Larger size filters show less performance due to the increase in adder sizes on critical path delay. This is also a consequence of the fact that routing delay dominates in FPGAs. This argument is strengthened by our results which show that smaller areas have smaller delays.

Figure 11 plots the reduction in the number of resources, in terms of the number of slices, LUTs and flip flops (FFs). From the results, we can observe an average reduction of 58.7% in the number of LUTs, and about 25% reduction in the number of slices and FFs. As it can be seen from the figure, the percentage of slices and FFs saved is roughly equal while the saving amount for LUTs is substantially higher. This is due to the fact that Xilinx synthesis tool does not report the slice as a used slice if the corresponding register element is not used.

In DA full parallel implementation, LUT usage is high. Therefore the percentage of saving amount is also high. Though our modified CSE algorithm does not optimize for performance, the synthesis produces better performance in most of the cases, and for the 13 and 20 tap filters, an improvement of about 26% can be seen in performance (see Figure 10).

Figure 12 compares power consumption for our add/shift method versus Coregen. From the results we can observe up to 50% reduction in dynamic power consumption. The quiescent power is not included in calculations since that value is the same for both methods. The power consumption is the result of applying the same test stimulus to both designs and measuring the power using XPower tools. Coregen can produce FIR filters based on the MAC method, which makes use of the embedded multipliers and DSP blocks. We have implemented the FIR filters using the Coregen MAC method to compare the resource usage and performance to the add and shift method. Due to tool limitations (MAC filters can not be targeted Virtex II devices using Xilinx ISE software), experiments are done for Virtex IV devices. Synthesis results are presented in terms of the number of slices on the Virtex IV device and the performance in Msps in Figure 13.

In Figure 13(a), add and shift method shows higher area compared to MAC implementation. MAC implementation uses DSP blocks to implement the MAC operation (shown in logarithmic scale). For instance a 151 tap FIR filter uses 151 DSP blocks and the rest of the logic is implemented using slice LUTs. There was no pipelining in the MAC implementation. Also the input width is the same as add and shift or DA method. In all cases, the input width was assumed to be 12 bits.

Figure 13(b) shows higher performance for add and shift method compared to MAC implementation. Routing delay dominates in FPGAs. The MAC implementation uses embedded DSP blocks and it adds to the routing delay due to the fact that signals have to travel outside the CLBs. Another limitation for MAC method is that Xilinx Coregen is limited to input width of 18 bits due to the embedded DSP block input limitation while our add and shift method can accept inputs of any width.

In this work, a comparison is made primarily with the Coregen implementation of DA, which is also a multiplierless technique. Based on the implementation results, our designs are much more area efficient than the DA based approach for fully parallel FIR filters. We also compare our method with MAC based implementations, where significantly higher performance is achieved (see Figure 13). The DA technique used by Xilinx Coregen stores the coefficients in LUTs. This makes the coefficient values relatively easy to change, if necessary. Our method uses a series of add and shifts to produce coefficients. In the case where the coefficients change, a recompile is needed to reproduce a new add and shift block specifically for the new coefficients. So in applications such as adaptive filters where this happens frequently, DA is the method of choice. However in applications with constant coefficients, our method is superior.

5.2. FIR Implementation Using Add and Shift Method versus Competing Methods. In the following, the add and shift

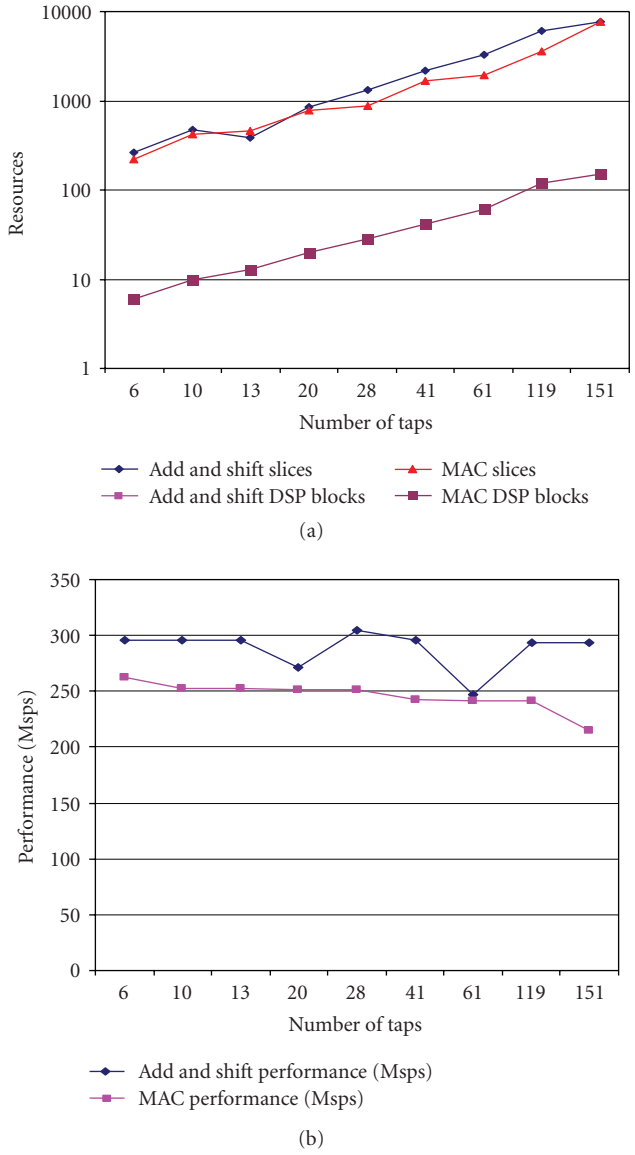

Figure 13: Resource utilization and performance implementation results for various filters using add and shift method (this paper) versus MAC method on Virtex IV. (a) Resource utilization in terms of no. of slices and DSP blocks presented in logarithmic scale. (b) Performance (Msps).

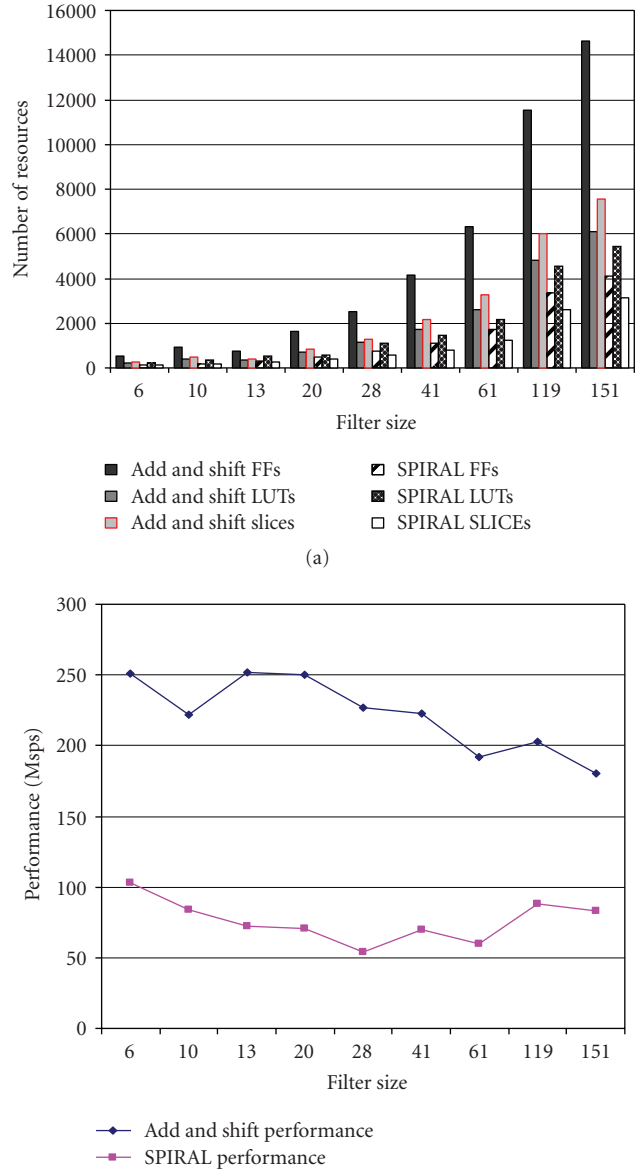

Figure 14: Resource utilization and performance implementation results for various filters using add and shift method (this paper) relative to that of SPIRAL automatic software. SPIRAL shows a saving of 72% in FFs, 11% in LUTs, and 59% in slices at the cost of 68% drop in performance. (a) Resource utilization in terms of # of FFs, LUTs, and SLICEs. (b) Performance (Msps).

method experimental results are compared against two competing methods: SPIRAL automatic software and RAG-*n*.

SPIRAL is a system that automatically generates platform-adapted libraries for DSP transforms. The system uses a high level algebraic notation to represent, generate, and manipulate various algorithms for a user specified transform. SPIRAL optimizes the designs in terms of number of additions and it tunes the implementation to the platform by intelligently searching in the space of different algorithms and their implementation options for the fastest on the given platform. The SPIRAL software is available for download. SPIRAL generates the C code (not the HDL code) for multiplier block of the FIR filter. In order to have a complete comparison, the C code for the multiplier block was generated for each filter using SPIRAL software and then converted to HDL code with the addition of the delay line. The resulting code was run by Xilinx ISE software and the implementation results are shown in Figure 14 for both area and performance.

In order to have a fair comparison, all inputs and outputs were registered. We implemented all experiments with the HDL codes (converted C code that was generated by SPIRAL software) and the results are shown in Figure 14. Figure 14(a) shows the FPGA area in terms of number of FFs, LUTs, and SLICEs and Figure 14(b) shows the performance.

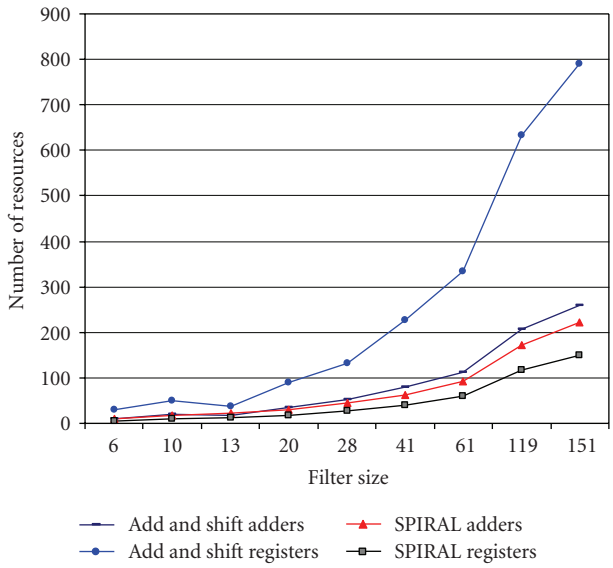

Figure 15: High level resource utilization in terms of # adders and registers for various filters using add and shift method (this paper) versus SPIRAL automatic software. SPIRAL shows a saving of 15% in number of adders and 81% in number of registers at the cost of 68% drop in performance.

The reason for the reduction in performance is the depth of the adder tree in multiplier block since this block is not pipelined by SPIRAL. The depth of the adder tree in multiplier block is dependent on the coefficients used and in some cases is as high as 7 levels of cascaded adders. The average performance for SPIRAL implementation is 73 Mhz as opposed to 231 Mhz for our add and shift method. There is a trade-off between performance and FPGA area in this case. Implementation results show that the drop in performance comes at an improvement to the FPGA area. The average FPGA area for various size filters is 2400 FFs, 1016 LUTs, and 1242 slices for add and shift method versus 679 FFs, 909 LUTs, and 512 slices for SPIRAL. There is a saving of 72% in FFs, 11% in LUTs, and 59% in slices at the cost of 68% drop in performance. Another interesting fact that can be seen in Figure 14(a) is that the number of LUTs used is very close in both methods. This means that both methods behave very closely when it comes to synthesizing adders. Our add and shift method takes advantage of registered adders depicted in Figure 5 and inserts registers whenever possible (without adding to area) to improve performance. Due to this fact, we show better performance for all size filters comparable with SPIRAL even though we are not optimizing our designs for performance.

The SPIRAL implementation is an optimum solution for software oriented platforms since it focuses on minimizing number of additions. However, this is not necessarily the best method for FPGA implementation. An important factor in FPGA implementation is to use the slice architecture in an efficient way and have a balanced usage of LUTs and registers.

Figure 15 provides the high level cost measure of the add and shift method versus SPIRAL. Both number of adders and registers that are synthesized are shown using each method. SPIRAL uses 16% less number of adders and 81% less number of registers compared to add and shift at the cost of 68% drop in performance.

It is impossible to compare our implementation results with RAG-n presented in [30] directly due to several reasons such as targeting a different Altera FPGA versus Xilinx, coefficients magnitude, filter size, and so forth. However, these numbers can be compared indirectly assuming Xilinx logic cells (LCs) are equivalent to Altera logic elements considering a conversion factor. In fact, each Xilinx LC is 1.125 Altera LE (This number is reported on manufacturer's websites [41]). Since we donot know the RAG-n method filter sizes, we can find the same size filters using FPGA area reported. Taking all these into account the implementation results for our add and shift method show size reduction of 59%, performance of +11% and cost improvement of 82% expressed as LCs/Fmax compared to DA. This shows our method is advantageous regardless of the coefficients. The authors in [30] specifically mention that RAG-n works best when many small coefficients are available, while DA offers greater advantage when there are many large coefficients.

5.3. FIR Implementation Using Mutual Contraction Concept. Mutual contraction defines a new edge weight for nets and then computes the relative weight of a connection. It can be used to estimate the relative length of interconnect. This concept can be extended to measure the contraction of a node group. Our CSE based cost function considers only area reduction as a constraint. It is based on extracting the divisors in a polynomial that minimizes the number of operations needed but constraints have been modified to incorporate mutual contraction concept. Section 5.3.1 describes the design flow and Section 5.3.2 discusses the experimental results.

5.3.1. Design Flow Using Mutual Contraction Metric. Figure 16 summarizes the steps taken towards our goals. Our experiments are based on implementation of different size FIR filters with fixed coefficients. We performed two term CSE for three cases trying to maximize and minimize the mutual contraction (according to the criteria explained in Section 4.2) and also with no consideration of interconnect mutual contraction effect. Thereafter, HDL RTL code for each case was generated. There are five RTL HDL codes for each size filter. For all cases, RTL code was synthesized and run through VPR Place and Route tool to compare the results.

For placement and routing we have followed VPR design flow summarized in [42]. High level language files (HDL) are read by the synthesis tool. In our experiment, Altera and QUIP toolsets are used to generate.BLIF (Berkeley Logic Interchange Format) file. The goal of BLIF file is to describe a logic level hierarchical circuit in textual form. A circuit can be viewed as a directed graph of combinational logic nodes and sequential logic elements. TVPACK and VPR tools do not support Xilinx ISE software. Furthermore, Xilinx ISE toolset does not provide any interconnect information for a placed and routed design). T-VPack is a packing program which can

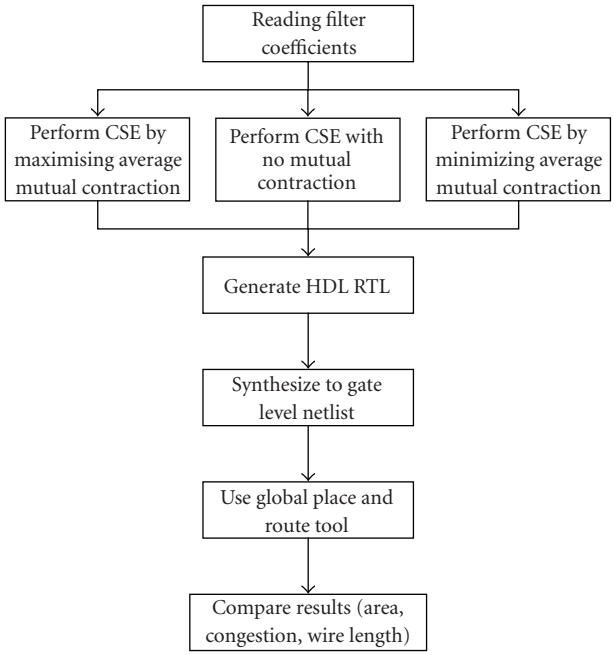

Figure 16: Implementation flow using mutual contraction concept.

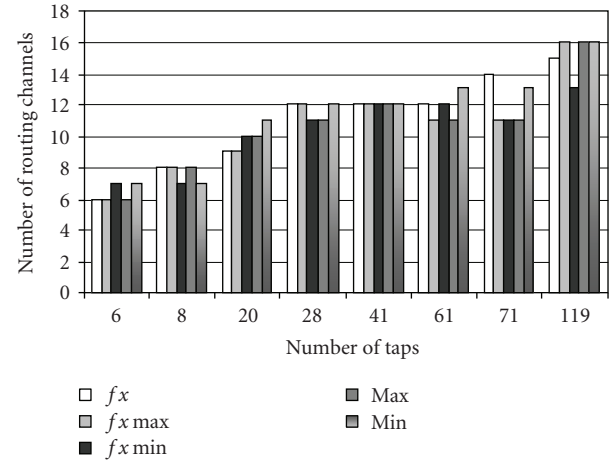

Figure 17: Number of routing channels versus filter size for various cost functions discussed in Section 4.3 with f_x being the modified CSE algorithm presented in Algorithm 1 and others based on maximizing or minimizing AMC. f_{xmin} is the best scenario that results in the minimum number of routing channels.

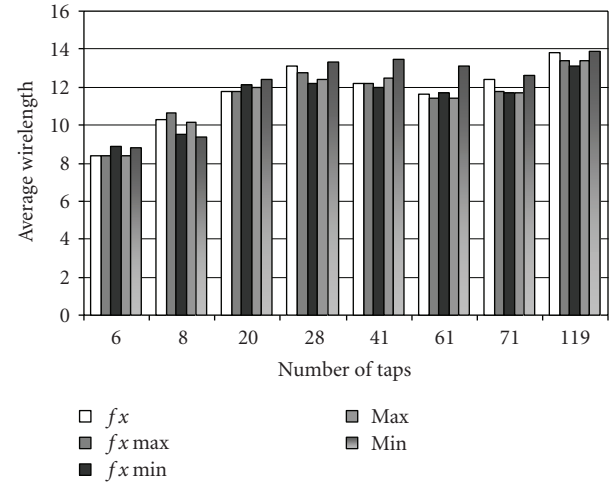

Figure 18: Average wirelength versus filter size. for various cost functions discussed in Section 4.3 with f_x being the modified CSE algorithm presented in Algorithm 1 and others based on maximizing or minimizing AMC. f_{xmin} is the best scenario that results in the minimum number of routing channels.

be used with or without VPR. It takes a technology-mapped netlist (in.BLIF format) consisting of LUTs and flip flops (FFs) and packs the LUTs and FFs together to form more coarse-grained logic blocks and outputs a netlist in the.NET format that VPR uses. VPR then reads.NET file along with the architecture file (.ARCH) and generates PAR files. VPR is an FPGA PAR tool. The output of VPR consists of a file describing the circuit placement (.P) and circuit's routing (.ROUTING). The.ARCH is another input to the VPR tool that defines the FPGA architecture for the VPR tool. VPR tool lets the user define the FPGA architecture and reads that as an input file.

5.3.2. Implementation Results Using Mutual Contraction Metric. We have implemented various size FIR filters taking mutual contraction into account. We have embedded four additional constraints introduced in Section 4.3 (f_{xMin}, f_{xMax}, f_{min}, f_{max}) into our cost function and regenerated the HDL codes and implemented all FIR designs. The place and route information can be obtained after implementing the designs. Figures 17 and 18 represent the data obtained after the implementation for both placement and routing of different size filters.

Figure 17 shows the number of routing channels versus number of taps for different size filters. Here Fx is the modified CSE algorithm presented in Algorithm 1 which is based on CSE. f_{xmin} is the best approach in terms of reduction in number of routing channels. Figure 18 shows the average wirelength versus filter size. f_{xmin} still shows maximum reduction in wirelength especially for large size filters.

For placement, as Figure 17 shows, there is a saving of up to 20% in the number of routing channels. This results in lower congestion. There is up to 8% saving in average wirelength for Fxmin as depicted in Figure 18. There is a trivial 2-3% saving in number of logic blocks for Fxmin. There are two factors here that can be affected by changing parameters: number of wires and wirelength. Saving number of adders reduces number of wires, and wirelength can be reduced by manipulating mutual contraction. As it can be seen from the figures, Max and Min are the worst cases since these two methods focus on maximizing or minimizing mutual contraction among the divisors regardless of saving number of additions. F_x was the modified CSE algorithm presented in Algorithm 1 with no mutual contraction incorporated and it only concentrates on saving number of additions. In general

maximizing mutual contraction minimizes the wirelength which means F_{xmax} should give the best results. However, this is not always the case. F_{xmin} scenario results in maximum saving. There seems to be a complex interplay between these two factors (wirelength and number of wires). Consequently, we see sporadic results even though most of the cases offer some saving in both wirelength and number of wires.

In comparison with [27], Common subexpression elimination is extensively used to reduce the number of adders and therefore area. Furthermore, our designs can run with sample rates as high as 252 Msps, whereas the designs in [27] can run only at 78.6 Msps. A summary of the above results is posted on [43].

6. Conclusions

In this paper, we presented a multiplierless technique, based on add and shift method and common subexpression elimination for low area, low power and high speed implementations of FIR filters. Our techniques are validated on Virtex II and Virtex 4 devices where significant area and power reductions are observed over traditional DA based techniques. In future, we would like to improve our modified CSE algorithm to make use of the limited number of embedded multipliers available on the FPGA devices. Also, the new cost function can be embedded into other optimization algorithms such as RAG-n or Hcub (embedded in SPIRAL) as future work.

We have extended our add and shift method to reduce the FPGA resource utilization by incorporating mutual contraction metric that estimates pre-layout wirelength. The original cost function in add and shift method is modified using mutual contraction concept to introduce five different constraints, two of which maximize and two others minimize the average mutual contraction. As a result, an improvement is expected in routing and total wirelength in routed design. Based on the overall results f_{xmin} scenario seems to be better in terms of placement and routing. In f_{xmin}, AMC is minimized among the divisors that save maximum number of additions.

For routing, there is up to 8% saving in average wirelength and up to 20% in number of routing channels for f_{xmin} compared to f_x algorithm (modified CSE algorithm). There is also a trivial 2-3% saving in number of logic blocks for this scenario. The obtained results related to routing could be a significant factor for high density designs since routing issues start to appear.

In comparison with SPIRAL, our method shows better performance. SPIRAL shows a saving of 72% in FFs, 11% in LUTs, and 59% in slices at the cost of 68% drop in performance. SPIRAL multiplier block is not pipelined and depending on the coefficients used, the cascaded adder tree could synthesize to several levels of logic and consequently result into low performance. This is a good solution for software implementation but not necessarily for FPGA implementation. An important factor in FPGA implementation is to use the slice architecture in an efficient way. Each FPGA slice includes a combinatorial part (LUT) and a storage element (register). Multiplier block generated by SPIRAL uses only the LUTs and registers that are left can not be used for other logic and consequently they are wasted.

References

[1] K. D. Underwood and K. S. Hemmert, "Closing the gap: CPU and FPGA trends in sustainable floating-point BLAS performance," in *Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '04)*, pp. 219–228, Napa, Calif, USA, April 2004.

[2] L. Zhuo and V. K. Prasanna, "Sparse matrix-vector multiplication on FPGAs," in *Proceedings of the International Symposium on Field Programmable Gate Arrays (FPGA '05)*, pp. 63–74, Monterey, Calif, USA, 2005.

[3] Y. Meng, A. P. Brown, R. A. Iltis, T. Sherwood, H. Lee, and R. Kastner, "MP core: algorithm and design techniques for efficient channel estimation in wireless applications," in *Proceedings of the Design Automation Conference (DAC '05)*, pp. 297–302, Anaheim, Calif, USA, 2005.

[4] B. L. Hutchings and B. E. Nelson, "GIGAOP DSP on FPGA," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '01)*, vol. 2, pp. 885–888, Salt Lake, Utah, USA, 2001.

[5] A. Alsolaim, J. Becker, M. Glesner, and J. Starzyk, "Architecture and application of a dynamically reconfigurable hardware array for future mobile communication systems," in *Proceedings of the International Symposium on Field Programmable Custom Computing Machines (FCCM '00)*, Napa, Calif, USA, 2000.

[6] S. J. Melnikoff, S. F. Quigley, and M. J. Russell, "Implementing a simple continuous speech recognition system on an FPG," in *Proceedings of the International Symposium on Field-Programmable Custom Computing Machines (FCCM '02)*, Napa, Calif, USA, 2002.

[7] T. Yokota, M. Nagafuchi, Y. Mekada, T. Yoshinaga, K. Ootsu, and T. Baba, "A scalable FPGA-based custom computing machine for medical image processing," in *Proceedings of the International Symposium on Field-Programmable Custom Computing Machines (FCCM '02)*, Napa, Calif, USA, 2002.

[8] H.-J. Kang, H. Kim, and I.-C. Park, "FIR filter synthesis algorithms for minimizing the delay and the number of adders," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD '00)*, pp. 51–54, San Jose, Calif, USA, 2000.

[9] A. Hosangadi, F. Fallah, and R. Kastner, "Reducing hardware compleity of linear DSP systems by iteratively eliminating two term common subexpressions," in *Proceedings of the Asia South Pacific Design Automation Conference (ASP-DAC '05)*, Shanghai, China, 2005.

[10] A. G. Dempster and M. D. Macleod, "Use of minimum-adder multiplier blocks in FIR digital filters," *IEEE Transactions on Circuits and Systems II*, vol. 42, no. 9, pp. 569–577, 1995.

[11] O. Gustafsson, A. G. Dempster, and L. Wanhammar, "Extended results for minimum-adder constant integer multipliers," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS '02)*, vol. 1, pp. I73–I76, Scottsdale, Ariz, USA, 2002.

[12] Y. Voronenko and M. Puschel, "Multiplierless multiple constant multiplication," *ACM Transactions on Algorithms*, vol. 3, no. 2, 2007.

[13] N. Al-Dhahir, A. H. H. Sayed, and J. M. Cioffi, "Stable pole-zero modeling of long FIR filters with application to the MMSE-DFE," *IEEE Transactions on Communications*, vol. 45, no. 5, pp. 508–513, 1997.

[14] A. Hosangadi, F. Fallah, and R. Kastner, "Optimizing polynomial expressions by algebraic factorization and common subexpression elimination," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 10, pp. 2012–2021, 2006.

[15] B. Hu and M. Marek-Sadowska, "Wire-length prediction based clustering and its application in placement," in *Proceedings of the Design Automation Conference (DAC '03)*, pp. 800–805, Anaheim, Calif, USA, 2003.

[16] K. Chapman, "Constant coefficient multipliers for the XC4000E," Xilinx Application Note, 1996, http://www.xilinx.com/.

[17] K. Wiatr and E. Jamro, "Constant coefficient multiplication in FPGA structures," in *Proceedings of the 26th EUROMICRO Conference*, Maastricht, The Netherlands, 2000.

[18] M. J. Wirthlin and B. Mcmurtrey, "Efficient constant coefficient multiplication using advanced FPGA architectures," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL '01)*, Belfast, UK, 2001.

[19] M. J. Wirthlin, "Constant coefficient multiplication using look-up tables," *Journal of VLSI Signal Processing*, vol. 36, no. 1, pp. 7–15, 2004.

[20] "Distributed Arithmetic FIR Filter v9.0," Xilinx Product Specification, 2005, http://www.xilinx.com/.

[21] T. Sasao, Y. Iguchi, and T. Suzuki, "On LUT cascade realizations of FIR filters," in *Proceedings of the 8th Euromicro Conference on Digital System Design (DSD '05)*, pp. 467–474, Porto, Portugal, 2005.

[22] G. R. Goslin, "A guide to using field programmable gate arrays (FPGAs) for application-specific digital signal processing performance," Xilinx Application Note, 1995, http://www.xilinx.com/.

[23] A. Peled and B. Liu, "A new hardware realization of digital filters," *IEEE Transactions on Acoustics, Speech, Signal Processing*, vol. 22, no. 6, pp. 456–462, 1974.

[24] M. Potkonjak, M. B. Srivastava, and A. P. Chandrakasan, "Multiple constant multiplications: efficient and versatile framework and algorithms for exploring common subexpression elimination," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 2, pp. 151–165, 1996.

[25] R. T. Hartley, "Subexpression sharing in filters using canonic signed digit multipliers," *IEEE Transactions on Circuits and Systems II*, vol. 43, no. 10, pp. 677–688, 1996.

[26] H. T. Nguyen and A. Chatterjee, "Number-splitting with shift-and-add decomposition for power and hardware optimization in linear DSP synthesis," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 8, no. 4, pp. 419–424, 2000.

[27] M. Yamada and A. Nishihara, "High-speed FIR digital filter with CSD coefficients implemented on FPGA," in *Proceedings of the Asia South Pacific Design Automation Conference (ASP-DAC '01)*, Yokohama, Japan, 2001.

[28] O. Gustafsson, A. G. Dempster, K. Johansson, M. D. Macleod, and L. Wanhammar, "Simplified design of constant coefficient multipliers," *Circuits, Systems, and Signal Processing*, vol. 25, no. 2, pp. 225–251, 2006.

[29] H. Safiri, M. Ahmadi, G. A. Jullien, and W. C. Miller, "A new algorithm for the elimination of common subexpressions in hardware implementation of digital filters by using genetic programming," *Journal of VLSI Signal Processing*, vol. 31, no. 2, pp. 91–100, 2002.

[30] U. Meyer-Baese, J. Chen, C. Chang, and A. Dempster, "A comparison of pipelined RAG-n and DA FPGA-based multiplierless filters," in *Proceedings of the IEEE Asia Pacific Conference on Circuits and Systems (APCCAS '06)*, pp. 1557–1560, Singapore, December 2006.

[31] K. N. Macpherson and R. W. Stewart, "RAPID PROTOTYPING—area efficient FIR filters for high speed FPGA implementation," *IEE Proceedings on Vision, Image and Signal Processing*, vol. 153, no. 6, pp. 711–720, 2006.

[32] Uwe Meyer-Baese, *Digital Signal Processing With Field Programmable Gate Arrays*, Springer, Berlin, Germany, 2004.

[33] Multiplier V10.1, "Xilinx Product Specification," http://www.xilinx.com/, April 2008.

[34] A. Croisier, D. J. Esteban, M. E. Levilion, and V. Rizo, "Digital filter for PCM encoded signals," US patent 3, 777, 130, December 1973.

[35] S. Zohar, "The counting recursive digital filter," *IEEE Transactions on Computers*, vol. C22, no. 4, pp. 338–347, 1973.

[36] A. M. Al-Haj, "Fast discrete wavelet transformation using FPGAs and distributed arithmetic," *International Journal of Applied Science and Engineering*, vol. 1, no. 2, pp. 160–171, 2003.

[37] A. Hosangadi, F. Fallah, and R. Kastner, "Common subexpression elimination involving multiple variables for linear DSP synthesis," in *Proceedings of the International Conference on Application-Specific Systems, Architectures and Processors*, pp. 202–212, September 2004.

[38] P. Flores, J. Monteiro, and E. Costa, "An exact algorithm for the maximal sharing of partial terms in multiple constant multiplications," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD '05)*, pp. 13–16, San Jose, Calif, USA, 2005.

[39] S. Hauck and G. Borriello, "An evaluation of bipartitioning techniques," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, no. 8, pp. 849–866, 1997.

[40] J. Cong and S. K. Lim, "Edge separability based circuit clustering with application to circuit partitioning," in *Proceedings of the Asia South Pacific Design Automation Conference (ASP-DAC '00)*, pp. 429–434, Yokohama, Japan, 2000.

[41] http://www.altera.com/cgi-bin/device_compare.pl.

[42] V. Betz and J. Rose, "VPR: a new packing, placement and routing tool for FPGA research," in *Proceedings of the 7th International workshop on Field Programmable Logic and Applications (FPLA '97)*, pp. 213–222, London, UK, 1997.

[43] http://cse.ucsd.edu/~kastner/research/fir_benchmarks.