FPGA Implementation of High Speed FIR Filters Using Add and Shift Method

Shahnam Mirzaei, Anup Hosangadi, Ryan Kastner
University Of California, Santa Barbara, CA 93106
E-mail: shahnam@umail.ucsb.edu, anup@ece.ucsb.edu, kastner@ece.ucsb.edu

***Abstract*-We present a method for implementing high speed Finite Impulse Response (FIR) filters using just registered adders and hardwired shifts. We extensively use a modified common subexpression elimination algorithm to reduce the number of adders. We target our optimizations to Xilinx Virtex II devices where we compare our implementations with those produced by Xilinx CoregenTM using Distributed Arithmetic. We observe up to 50% reduction in the number of slices and up to 75% reduction in the number of LUTs for fully parallel implementations. We also observed up to 50% reduction in the total dynamic power consumption of the filters. Our designs perform significantly faster than the MAC filters, which use embedded multipliers.**

I. INTRODUCTION

FPGAs are being increasingly used for a variety of computationally intensive applications, mainly in the realm of Digital Signal Processing (DSP) and communications [1-7]. Due to rapid increases in the technology, current generation of FPGAs contain a very high number of Configurable Logic Blocks (CLBs), and are becoming more feasible for implementing a wide range of applications. The high non-recurring engineering (NRE) costs and long development time for ASICs are making FPGAs more attractive for application specific DSP solutions. DSP functions such as FIR filters and transforms are used in a number of applications such as communication and multimedia. These functions are major determinants of the performance and power consumption of the whole system. Therefore it is important to have good tools for optimizing these functions.

Equation (I) represents the output of an L tap FIR filter, which is the convolution of the latest L input samples. L is the number of coefficients h(k) of the filter, and x(n) represents the input time series.

$$y[n] = \sum h[k]\, x[n-k] \qquad k = 0, 1, ..., L-1 \qquad (I)$$

The conventional tapped delay line realization of this inner product is shown in Figure 1. This implementation translates to L multiplications and L-1 additions per sample to compute the result. This can be implemented using a single Multiply Accumulate (MAC) engine, but it would require L MAC cycles, before the next input sample can be processed. Using a parallel implementation with L MACs can speed up the performance L times. A general purpose multiplier occupies a large area on FPGAs. Since all the multiplications are with constants, the full flexibility of a general purpose multiplier is not required, and the area can be vastly reduced using techniques developed for constant multiplication. Though most of the current generation FPGAs such as Virtex II™ have embedded multipliers to handle these multiplications, the number of these multipliers is typically limited. Furthermore, the size of these multipliers is limited to only 18 bits, which limits the precision of the computations for high speed requirements. The ideal implementation would involve a sharing of the Combinational Logic Blocks (CLBs) and these multipliers. In this paper, we present a technique that is better than conventional techniques for implementation on the CLBs.

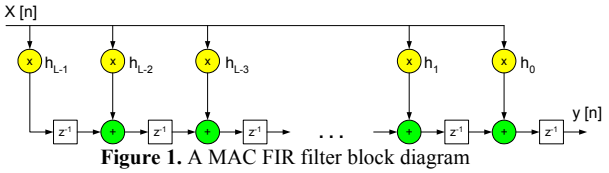


Figure 1. A MAC FIR filter block diagram

An alternative to the above approach is Distributed Arithmetic (DA) which is a well known method to save resources. Using DA method, the filter can be implemented either in bit serial or fully parallel mode to trade bandwidth for area utilization. Assuming coefficients c[n] are known constants, equation (I) can be rewritten as follows:

$$y[n] = \sum c[n] \cdot x[n] \qquad n = 0, 1, \ldots, N-1 \qquad (II)$$

Variable x[n] can be represented by:

$$x[n] = \sum x_b[n] \cdot 2^b \qquad b=0, 1, \ldots, B-1 \qquad (III)$$
$$x_b[n] \in [0, 1]$$

where $x_b[n]$ is the b^{th} bit of x[n] and B is the input width. Finally, the inner product can be rewritten as follows:

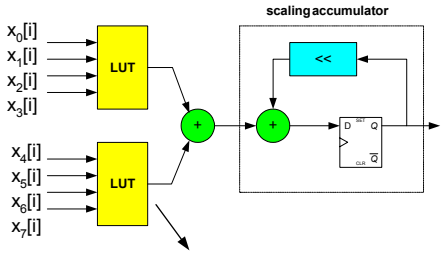
$$\begin{aligned} y &= \sum c[n] \sum x_b[k] \cdot 2^b \\ &= c[0]\,(x_{B-1}[0]2^{B-1} + x_{B-2}[0]2^{B-2} + \ldots + x_0[0]2^0) \\ &\quad + c[1]\,(x_{B-1}[1]2^{B-1} + x_{B-2}[1]2^{B-2} + \ldots + x_0[1]2^0) \\ &\quad + \ldots \\ &\quad + c[N-1]\,(x_{B-1}[N-1]2^{B-1} + x_{B-2}[0]2^{B-2} + \ldots + x_0[N-1]2^0) \\ &= (c[0]\,x_{B-1}[0] + c[1]\,x_{B-1}[1] + \ldots + c[N-1]\,x_{B-1}[N-1])2^{B-1} \\ &\quad + (c[0]\,x_{B-2}[0] + c[1]\,x_{B-2}[1] + \ldots + c[N-1]\,x_{B-2}[N-1])2^{B-2} \\ &\quad + \ldots \\ &\quad + (c[0]\,x_0[0] + c[1]\,x_0[1] + \ldots + c[N-1]\,x_0[N-1])2^0 \\ &= \sum 2^b \sum c[n] \cdot x_b[k] \end{aligned} \qquad (IV)$$

where $n=0, 1, \ldots, N-1$ and $b=0, 1, \ldots, B-1$

The coefficients in most of DSP applications for the multiply accumulate operation are constants. The partial products are obtained by multiplying the coefficients c_i by multiplying one bit of data x_i at a time in AND operation. These partial products should be added and the result depend only on the outputs of the input shift registers. The AND functions and adders can be replaced by Look Up Tables (LUTs) that gives the partial product. This is shown in Figure 2. Input sequence is fed into the shift register at the input sample rate. The serial output is presented to the RAM based shift registers (registers are not shown in Figure for simplicity) at the bit clock rate which is n+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 which is the filter output will be accumulated over the time. For an n bit input, n+1 clock cycles are needed for a symmetrical filter to generate the output.

In conventional MAC method with a limited number of MAC engines, as the filter length is increased, the system sample rate is decreased. 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 consumed.

Though 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 samples are processed. Each bit of the current input samples takes one clock cycle to process.



Address	Data
0000	0
0001	C_0
0010	C_0+C_1
…	…
1111	$C_0+C_1+C_2+C_3$

Figure 2. A serial DA FIR filter block diagram

Therefore, if the input bitwidth is 12, then a new input can be sampled every 12 clock cycles. The performance of the circuit can be improved by modifying the architecture to a parallel architecture which processes the data bits in groups. Figure 3 shows the block diagram of a 2 bit parallel DA FIR filter. The tradeoff here is performance for area since increasing the number of bits sampled has a significant effect on resource utilization on FPGA. For instance, doubling the number of bits sampled, doubles the throughput and results in the 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 implementation of the DA filter (PDA), the number of LUTs required would be enormous. In this work we show an alternative to the PDA method for implementing high speed FIR filters that consumes significantly lesser area and power.

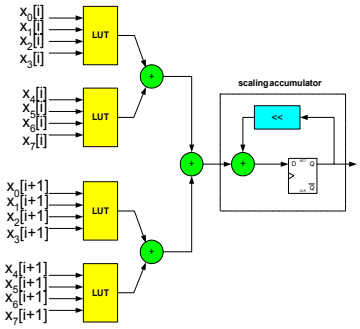


Figure 3. A 2 bit parallel DA FIR filter block diagram

A popular technique for implementing the transposed form of FIR filters is the use of a multiplier block, instead of using multipliers for each constant as shown in Figure 4. The multiplications with the set of constants $\{h_k\}$ are replaced by an optimized set of additions and shift operations, involving computation sharing. Further optimization can be done by factorizing the expression and finding common subexpressions. The performance of this filter architecture is limited by the latency of the biggest adder and is the same as that of the PDA.

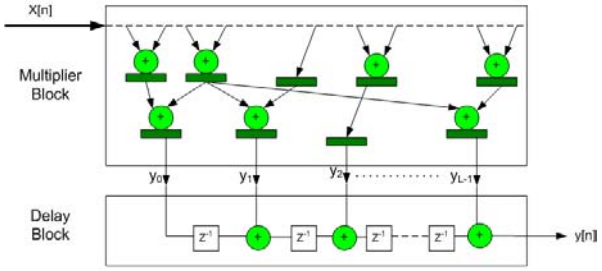


Figure 4. Replacing constant multiplication by multiplier block

The main contribution in this paper is the development of a novel algorithm for optimizing the multiplier block for FIR filters, using a modified algorithm for common subexpression elimination. The goal of the algorithm is to produce a filter that can provide the maximum sample rate with the least amount of hardware. Our algorithm takes into account the specific features of FPGA slices 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.

We compare our results with the industry standard Xilinx Coregen™, where we compare the total area and power consumption.

The rest of the paper is organized as follows: Section 2 presents some related work. In Section3, we describe our filter architecture. In Section 4, we present our optimization algorithm for reducing the total area of the design. In Section 5, we describe our experimental setup and present our results. Finally we conclude the paper in Section 6.

II. RELATED WORK

Multiplications with constants have to be performed in many signal processing and communication applications such as FIR filters, audio, video and image processing. Since implementing a general purpose multiplier is expensive on an FPGA and since we do not really need such a multiplier, when one of the operands is a constant, there has been a lot of work

on deriving efficient structures for constant multiplications [8-13]. All these techniques are based on computing constant multiplications using table lookups and additions. The method of Distributed Arithmetic [12, 14] which is the most popular method for implementing Multiplierless FIR filters, is also based on table lookup. The Xilinx™ CORE Generator has a highly parameterizable, optimized filter core for implementing digital FIR filters [12]. based on both Distributed Arithmetic as well as MAC (Multiply Accumulate) based architectures. It generates synthesized core that targeting a wide range of Xilinx devices. The MAC based implementations make use of the embedded DSP slices on the FPGA devices. In this work, we primarily compare our technique with the Coregen implementation of the Distributed Arithmetic, since that also is a Multiplierless technique. We show that our designs are much more area efficient than the DA based approach for fully parallel filters. We also compare our method with MAC based implementations, where we achieve significantly higher performance

Though there has been a lot of work on optimizing constant multiplications using adders and employing redundancy elimination [15-19] , they have not been effectively used for FIR filter design. The closest work to implementing filters with adders is in [20], FIR filters are implemented using the Add and Shift method. Canonical Signed Digit (CSD) encoding is used for the coefficients to minimize the number of additions. The paper discusses how high speed implementations can be achieved by registering each adder, due to which the critical path becomes equal to the delay of the adder. Registering an adder output comes at no extra cost on an FPGA because of the presence of a D flip flop at the output of each LUT. In comparison with [20], we extensively use common subexpression elimination for reducing the number of adders and therefore area. Furthermore, our designs can run with sample rates as high as 252 Msps (Million samples per second), whereas the designs in [20] can run only at 78.6 Msps.

In comparison with the other algorithms for common subexpression elimination [15, 16, 18, 19, 21], our method takes into account the structure of the FPGA slices (Figure 5) and takes into account both the cost of adders and registers when performing the optimization. Furthermore, we provide comprehensive evidence of the benefits of our technique through experimental results, where we compare our results with those produced by industry standard tools.

III. FILTER ARCHITECTURE

We base our filter architecture on the transposed form of the FIR filter as shown in Figure 1. The filter can be divided into two main parts, the multiplier block and the delay block, and is illustrated in Figure 4. In the multiplier block, the current input variable x[n] is multiplied by all the coefficients of the filter to produce the y_i outputs. These y_i outputs are then delayed and added in the delay block to produce the filter output y[n].

We perform all our optimizations in the multiplier block. The constant multiplications are decomposed into registered additions and hardwire shifts. The additions are performed using two input adders, which are arranged in the fastest tree structure. We use registered adders, so that the performance of the filter is only limited by the slowest adder. We use common subexpression elimination extensively, to reduce the number of adders, which leads to a reduction in the area. To synchronize all the intermediate values in the computation, we insert registers in the dataflow, wherever necessary.

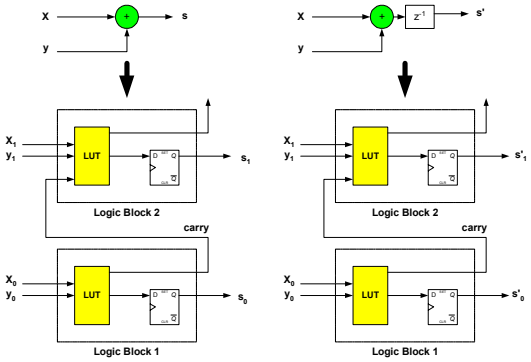


Figure 5. Registered adder at no additional cost

Performing 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.

$$F_1 = A + B + C + D$$
$$F_2 = A + B + C + E$$

Figure 6 shows the original unoptimized expression trees. Both the expressions have a minimum critical path of two addition cycles. These expressions require a total of six registered adders for the fastest implementation, and no extra registers are required. From the expressions we can see that the computation A + B + C is common to both the expressions. If we extract this subexpression, we get the structure shown in Figure 7. Since both D and E need to wait for two addition cycles to be added to (A + B + C), we need to use two registers each for D and E, such that new values for A,B,C,D and E can be read in at each clock cycle. Assuming that the cost of an adder and a register with the same bitwidth are the same, the structure shown in Figure 7 occupies more area than the one shown in Figure 6. A more careful subexpression elimination algorithm would only extract the common subexpression A + B (or A+C or B + C). The number of adders is decreased by one from the original, and no additional registers are added. This is illustrated in Figure 8. The algorithm for performing this kind of optimization is described in the next section.

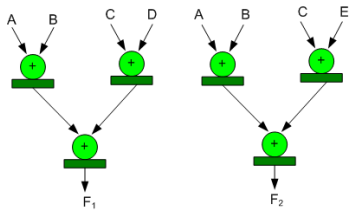


Figure 6. Unoptimized expression trees

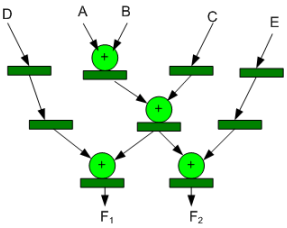


Figure 7. Extracting common expression (A + B + C)

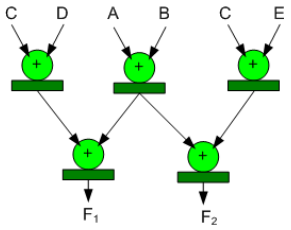


Figure 8. Extracting common subexpression (A+B)

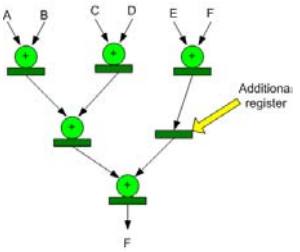


Figure 9. Calculating registers required for fastest evaluation

IV. OPTIMIZATION ALGORITHM

The goal of our optimization is to reduce the area of the multiplier block by reducing the number of adders and any additional registers required for the fastest implementation of the FIR filter. We first give a brief overview of the common subexpression elimination methods. A detailed description can be found in [22]. We then present the modified optimization algorithm to be used for our work.

A. Overview of common subexpression elimination

We use a polynomial transformation of 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

$$C*X = \sum_i \pm \mathbf{XL}^i \quad (V)$$

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

We use the **divisors** to represent all possible common subexpressions. 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 $F = XL^2 + XL^3 + XL^5$, consider the pair of terms $(+XL^2 + XL^3)$. The minimum exponent of L in the two terms is L^2. Dividing by L^2, we get the divisor $(X + XL)$. From the other two pairs of terms $(XL^2 + XL^5)$ and $(XL^3 + XL^5)$, we get the divisors $(X + XL^3)$ and $(X + XL^2)$ 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.

B. Optimization algorithm

We first calculate the minimum number of registers required for our design. We calculate this 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, we have the fastest tree structure with three addition steps, and we require one register to synchronize the intermediate values, such that new values for A,B,C,D,E,F can be read in every clock cycle. This is illustrated in Figure 9.

We first generate all the divisors for the set of expressions describing the multiplier block. We then use an iterative algorithm, where we extract the divisor that has the greatest value. To calculate the value of the divisor, we assume that the cost of a registered adder and a register is the same. We calculate the value of a divisor as the number of additions saved by extracting it minus the number of registers that have to be added. After selecting the best divisor, we rewrite the expressions using it. 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 iteration stops when there is no valuable divisor remaining in the set of divisors.

Consider the expressions shown in Figure 6. We need six registered adders and no additional registers for the fastest evaluation of F_1 and F_2. Now consider the selection of the divisor $d_1 = (A+B)$. This divisor saves one addition and does not increase the number of registers. Divisors (A + C) and (B + C) also have the same value, but (A+B) is selected randomly. The expressions are now rewritten as:

$$d_1 = (A + B)$$
$$F_1 = d_1 + C + D$$
$$F_2 = d_1 + C + E$$

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

Figure 10. Optimization algorithm to reduce area

After rewriting the expressions and forming new divisors, the divisor $d_2 = (d_1 + C)$ is considered. This divisor saves one adder, but introduces five additional registers, as can be seen in Figure 7. Therefore this divisor has a value of - 4. No other valuable divisors can be found and the iteration stops. We end up with the expressions shown in Figure 8.

V. EXPERIMENTS

The goal of our experiments was to compare the number of resources consumed by our add and shift method with that produced by the cores generated by the commercial Coregen™ tool, based on Distributed Arithmetic. Besides the resources, we also compared the power consumption of the two implementations, and also measured the performance. For our experiments, we considered 9 FIR filters of various sizes (6, 10, 13, 20, 28, 41, 61, 119 and 151 tap filters). We targeted the Xilinx Virtex II device for our experiments. 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, we decomposed all the constant multiplications into additions and shifts and optimized the expressions using the algorithm explained in Section 4.2. We used the Xilinx Integrated Software Environment (ISE) for performing synthesis and implementation of the designs. All the designs were synthesized for maximum performance.

Table 1a shows the resources utilized for the various filters and the performance in terms of Million samples per second (Msps) for the filters implemented using the add and shift method. Table 1b, shows the same numbers for the filters implemented using Xilinx Coregen, using the Parallel Distributed Arithmetic (PDA) method.

Table 1a. Filter Synthesis using Add Shift method

Filter (# taps)	Slices	LUTs	FFs	Performance (Msps)
6	264	213	509	251
10	474	406	916	222
13	386	334	749	252
20	856	705	1650	250
28	1294	1145	2508	227
41	2154	1719	4161	223
61	3264	2591	6303	192
119	6009	4821	11551	203
151	7579	6098	14611	180

Figure 11 plots the reduction in the number of resources, in terms of the number of Slices, Look Up Tables (LUTs) and the number of 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. Though our algorithm does not optimize for performance, the synthesis produces better performance in most of the cases, and for the 13 and 20 tap filters, we observe about 26% improvement in performance.

Table 1b. Filter Synthesis using Coregen (PDA method)

Filter (# taps)	Slices	LUTs	FFs	Performance (Msps)
6	524	774	1012	245
10	781	1103	1480	222
13	929	1311	1775	199
20	1191	1631	2288	199
28	1774	2544	3381	199
41	2475	3642	4748	222
61	3528	5335	6812	199
119	6484	9754	12539	205
151	8274	12525	15988	199

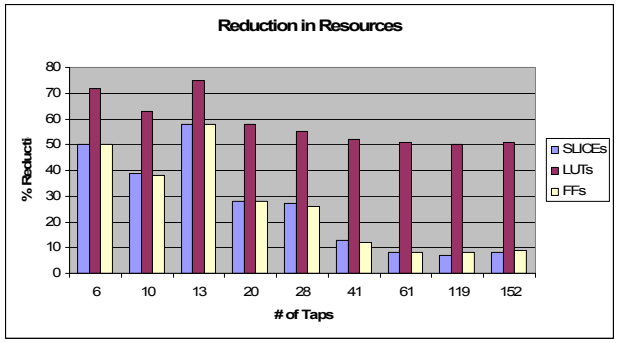


Figure 11. Reduction in resources

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. We did not include the quiescent power into our calculation 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 provided by Xilinx ISE software.

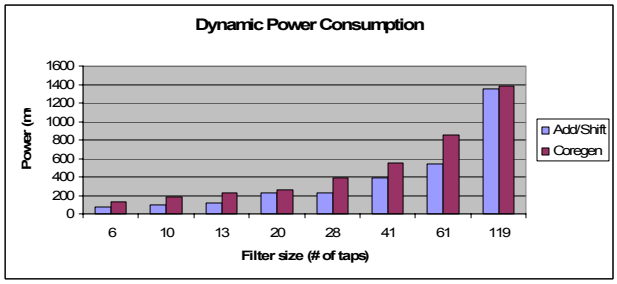


Figure 12. Power consumption

Comparison with MAC filters using embedded multipliers

Coregen™ can produce FIR filters based on the Multiply Accumulate (MAC) method, which makes use of the embedded multipliers and DSP blocks. We implemented the FIR filters using the MAC method to compare the resource usage and performance with our add and shift method. Due to tool limitations we had to do the experiments for Virtex IV device . We present the synthesis results in terms of number of slices on the Virtex IV device and the performance in Msps in Table 2.

Table 2. Comparing with MAC filter on Virtex IV

Filter (# taps)	Add Shift Method		MAC filter	
	Slices	Msps	Slices	Msps
6	264	296	219	262
10	475	296	418	253
13	387	296	462	253
20	851	271	790	251
28	1303	305	886	251
41	2178	296	1660	243
61	3284	247	1947	242
119	6025	294	3581	241
151	7623	294	7631	215

From the table, it can be seen that the MAC filter uses fewer number of slices compared to the add-shift method, but it also

uses the DSP blocks available on Virtex IV devices. The number of DSP blocks is equal to the number of taps of the filter. The results show that we achieve higher performance as the filter size increases. This is mainly because that critical path in our design consists of adders while in MAC method, critical path consists of multipliers and adders. Another limitation for MAC method is that Xilinx Coregen™ is limited to input width of 17 bits due to the embedded DSP block input limitation while our add and shift method can accept inputs of any width.

VI. CONCLUSION

In this paper we presented a multiplierless technique, based on the add and shift method and common subexpression elimination for low area, low power and high speed implementations of FIR filters. We validated our techniques on Virtex II™ devices where we observed significant area and power reductions over traditional Distributed Arithmetic based techniques. In future, we would like to modify our algorithm to make use of the limited number of embedded multipliers available on the FPGA devices.

VII. REFERENCES

[1] K.D.Underwood and K.S.Hemmert, "Closing the Gap: CPU and FPGA Trends in Sustainable Floating-Point BLAS Performance," presented at International Symposium on Field-Programmable Custom Computing Machines, California, USA, 2004.

[2] L.Zhuo and V.K.Prasanna, "Sparse Matrix-Vector Multiplication on FPGAs," presented at International Symposium on Field Programmable Gate Arrays (FPGA), Monterey, CA, 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," presented at Design Automation Conference (DAC), Anaheim, CA, 2005.

[4] B. L. Hutchings and B. E. Nelson, "Gigaop DSP on FPGA," presented at Acoustics, Speech, and Signal Processing, 2001. Proceedings. (ICASSP '01). 2001 IEEE International Conference on, 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," presented at International Symposium on Field Programmable Custom Computing Machines (FCCM), 2000.

[6] S.J.Melnikoff, S.F.Quigley, and M.J.Russell, "Implementing a Simple Continuous Speech Recognition System on an FPGA," presented at International Symposium on Field-Programmable Custom Computing Machines (FCCM), 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," presented at International Symposium on Field-Programmable Custom Computing Machines (FCCM), 2002.

[8] K.Chapman, "Constant Coefficient Multipliers for the XC4000E," Xilinx Technical Report 1996.

[9] K. Wiatr and E. Jamro, "Constant coefficient multiplication in FPGA structures," presented at Euromicro Conference, 2000. Proceedings of the 26th, 2000.

[10] M. J. Wirthlin and B. McMurtrey, "Efficient Constant Coefficient Multiplication Using Advanced FPGA Architectures," presented at International Conference on Field Programmable Logic and Applications (FPL), 2001.

[11] M.J.Wirthlin, "Constant Coefficient Multiplication Using Look-Up Tables," *Journal of VLSI Signal Processing*, vol. 36, pp. 7-15, 2004.

[12] "Distributed Arithmetic FIR Filter v9.0," Xilinx Product Specification 2004.

[13] T. Sasao, Y. Iguchi, and T. Suzuki, "On LUT Cascade Realizations of FIR Filters," presented at Euromicro Conference on Digital System Design (DSD), 2005.

[14] G.R.Goslin, "A Guide to Using Field Programmable Gate Arrays (FPGAs) for Application-Specific Digital Signal Processing Performance," Xilinx Application Note, San Jose 1995.

[15] 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*, 1996.

[16] R.I.Hartley, "Subexpression sharing in filters using canonic signed digit multipliers," *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on [see also Circuits and Systems II: Express Briefs, IEEE Transactions on]*, vol. 43, pp. 677-688, 1996.

[17] H.T.Nguyen and A.Chatterjee, "Number-splitting with shift-and-add decomposition for power and hardware optimization in linear DSP synthesis," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 8, pp. 419-424, 2000.

[18] H.-J. Kang, H. Kim, and I.-C. Park, "FIR filter synthesis algorithms for minimizing the delay and the number of adders," presented at Computer Aided Design, 2000. ICCAD-2000. IEEE/ACM International Conference on, 2000.

[19] A.Hosangadi, F.Fallah, and R.Kastner, "Reducing Hardware Compleity of Linear DSP Systems by Iteratively Eliminating Two Term Common Subexpressions," presented at Asia South Pacific Design Automation Conference, Shanghai, 2005.

[20] M. Yamada and A. Nishihara, "High-speed FIR digital filter with CSD coefficients implemented on FPGA," presented at Design Automation Conference, 2001. Proceedings of the ASP-DAC 2001. Asia and South Pacific, 2001.

[21] 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," presented at Application-Specific Systems, Architectures, and Processors, 2000. Proceedings. IEEE International Conference on, 2000.

[22] A.Hosangadi, F.Fallah, and R.Kastner, "Reducing Hardware complexity by iteratively eliminating two term common subexpressions," presented at Asia South Pacific Design Automation Conference (ASP-DAC), 2005.