International Journal of Technical Research and Applications e-ISSN: 2320-8163,
www.ijtra.com Special Issue 11 (Nov-Dec 2014), PP. 09-12

FIR FILTER IMPLEMENTATION BY USING BIT LEVEL TRANSFORMATION OF ADDER TREES FOR MCMS

R Srinivasa Rao[1], KSV Pavan Kumar[1], MD Javeed[2],
[1]Dept. of ECE, Khammam Institute of Technology and Science
Khammam, India
[2]Dept. of Electronics, Mewar University
Ghaziaba, India
[1]srinivasaug8@gmail.com, [2]pavan101088@gmail.com, [3]javeed@campuswave.in

***Abstract*---- Bit wise operations are used in many applications like digital signal processing and telecom etc. By the languages of high level programming in general purpose processors has become very costlier. In this paper we have shown Reverse bit level optimization for adder trees for multiple constant multiplications for the efficient implantation of Finite Impulse Response filters. Which decreases the cost as well the time is decreased by 21% as compared with the existing system. The code for the bit level optimization written in verilog and implemented in Xilinx Spartan 3e FPGA kit.**

***Index Terms*---- Multiple constant multiplications, Adder trees, Fir fiter, Bit level optimization.**

I. INTRODUCTION

Finite impulse response (FIR) digital filters are widely used as a basic tool in many digital signal processing (DSP) and communication applications. The complexity of a FIR filter is largely dominated by the multiplication of input samples with filter coefficients. But fortunately, the filter coefficients are constants for a given filter, so that multiplications are implemented by a network of adders, subtractors, and hardwired shifts, where the number of adders and subtractors are minimized by a constant multiplication scheme.

In case of a transposed direct-form FIR filter, the recent most input sample at any given clock period is multiplied with all the filter coefficients. A set of intermediate results are generated in this case, and shared across all the multiplications in order to minimize the total number of additions/subtractions using multiple constant multiplication (MCM) techniques as shown in Figure 1. Each such intermediate result in an MCM process corresponds to one of the common sub-expressions (CS) of the set of constants to be multiplied [1]. An finite impulse response (FIR) filter is a digital filter that works on digital inputs. A digital filter is a system that performs mathematical operations on sampled, discrete time signal to reduce or enhance certain aspect of that signal. There are two types of digital filter mainly used that are infinite response (IIR) filter and finite impulse response (FIR) filter [2].

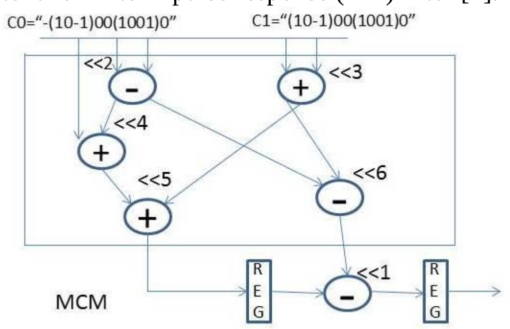


Figure 1 MCM composition block

There are two basic FIR structures, [3] direct form and transposed form, for a linear-phase even-order FIR filter. In the direct form, the multiple constant multiplication (MCM)/accumulation (MCMA) module performs the concurrent multiplications of individual delayed signals and respective filter coefficients, followed by accumulation of all the products. Thus, the operands of the multipliers in MCMA are delayed input signals $x[n - i]$ and coefficients ai. The results of individual constant multiplications go through structure adders (SAs) and delay elements. In order to avoid costly multipliers, most prior hardware implementations of digital FIR filters can be divided into two categories: multiplier less based and memory based [4].

II. BIT REVERSE FUNCTION

Bitwise operations are used extensively in many application domains, such as cryptography and telecommunications, etc. However, for applications written in high-level programming languages and executed on general-purpose processors, accessing and computing bit-values are relatively expensive, and bit level parallelism is not well exploited. This is mainly due to the lack of support in target machines, as well as high-level programming languages, such as C/C++. Most general-purpose processor architectures and high level programming languages do not support bitwise memory access and require a series of load/shift/mask/store instructions to implement simple bitwise operations, such as bit accessing and bit setting [7].

Customized hardware accelerators provide a promising approach to assisting general-purpose processors in exploiting performance of bitwise computation-intensive applications. Today, we can put more than one billion transistors in a single chip [5], and modern FPGAs allow users to exploit parallelism in applications by hundreds of thousands of logic cells and prefabricated IPs [6]. As RTL coding time is increasingly recognized as a significant component of the overall effort to solution, automated design processes and tools which compile higher-level abstraction into optimized hardware are gaining more and more popularity.

However, high-quality implementations are difficult to achieve automatically, especially when the description of the functionality is written in a high-level software programming language. For bitwise computation-intensive applications, one of the main difficulties is the lack of bit-accurate descriptions in high-level software programming languages. The wide use of bitwise operations in certain application domains calls for specific bit-level transformation and optimization to assist hardware synthesis of algorithmic descriptions [7].

Figure 1 and 2 shows example with a bit reversing function, where the bit reverse function takes a 32-bit integer as input and gets an output in the reverse bit order. The data-flow graph for the function is shown in Figure 1, while the optimal implementation is shown in Figure 2

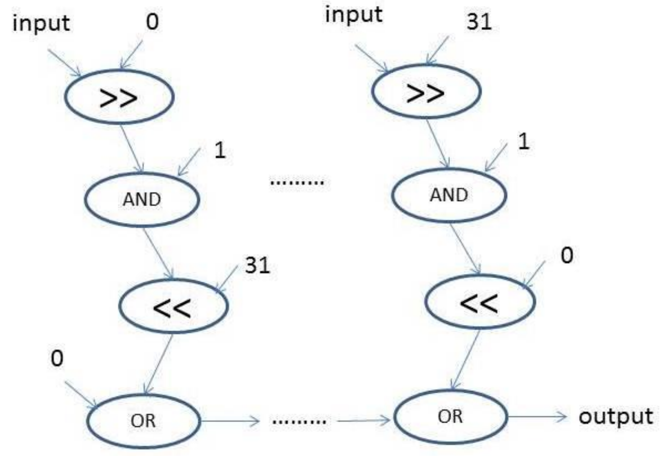


Fig.2 Bit reversible function data flow.

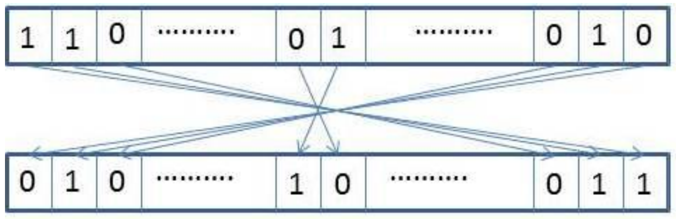


Fig.3 Implementation of bit reversible function

We can clearly see that the direct implementation based on the data-flow graph would use many more logical components and also have a longer latency compared to the optimal one, which only uses 32 wires to link the bits directly in the reverse order.

We can see from the example that efficient bit-level transformation and optimization for operations in algorithmic descriptions will lead to a much more direct and compact description. This will help the high level synthesis to generate better RTL designs for bitwise computation-intensive designs, and thus achieve better final implementations. Otherwise, in the absence of such optimization, the synthesis process can be misled by inaccurate area and timing estimation and thus generate suboptimal microarchitecture. It is often too late for the downstream RTL or logic synthesis and optimization techniques to make up for the QoR loss caused by the mistakes during compiler transformations.

III. PROPOSED SYSTEM

The common practice of handling the summation of CS terms of each coefficient is to use the tree-height minimization algorithm [1] to produce a height optimum adder-tree. Fig. 4 gives an example of the schedule for an adder-tree on the left with minimum delay.

Note that either a positive or negative sign is associated with each input term (see Fig. 2(a)), which denotes whether the corresponding term should be added to or subtracted from the summation. These signs also determine whether an addition operation or a subtraction operation should be used when the algorithm collapses a pair of terms in the adder-tree based on the following rules. (1) If two input edges are of the same sign, an ADD will be used; otherwise, it will be a SUB. (2) The sign of the output edge is always the same as that of the "left" input edge (i.e., theminuend edge in the subtraction case).Using these two rules, it is possible that the final term producing the summation result may carry a negative sign, such that a negation is needed after the adder-tree to correct the value. For an FIR filter, results from multiple adder-trees are accumulated by a structural

adder-register line. So the negation can be eliminated by replacing the structural adder with a subtractor (see coefficient in Fig. 1 for an example).

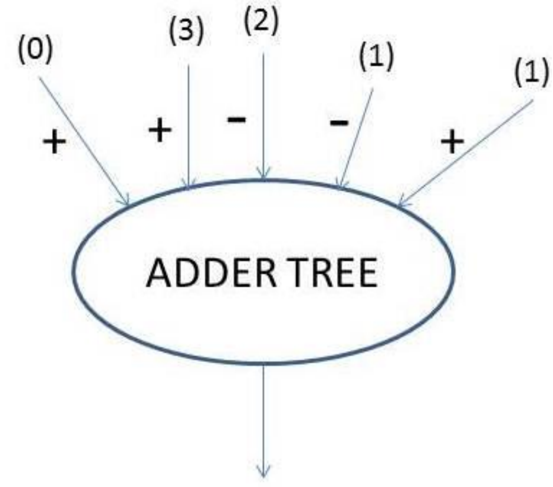


Fig. 4 Adder tree with delay and signs

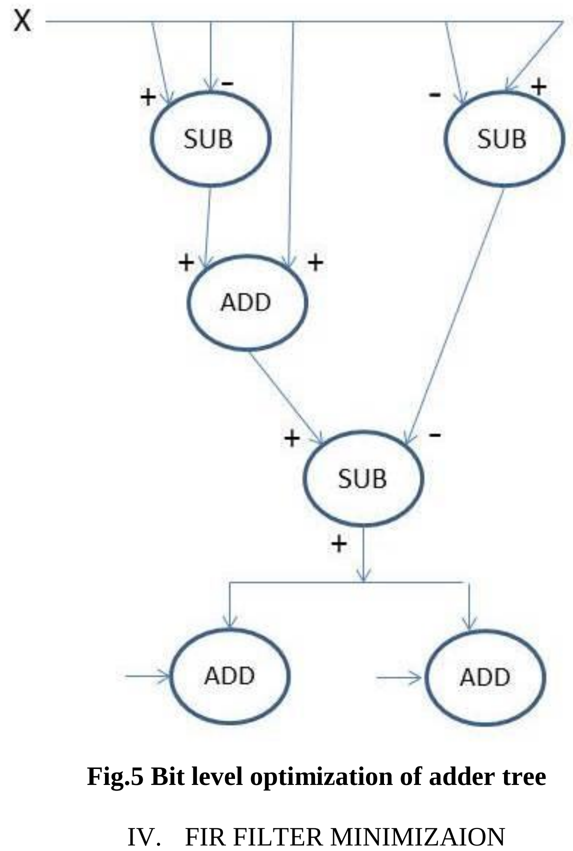


Fig.5 Bit level optimization of adder tree

IV. FIR FILTER MINIMIZAION

Finite impulse response (FIR) digital filter is one of the fundamental components in many digital signal processing (DSP) and communication systems. It is also widely used in many portable applications with limited area and power budget. A general FIR filter of order *M* can be expressed as

$Y[n]= a_i\Sigma^{m-1}_{i=0} \quad x[n-l]$

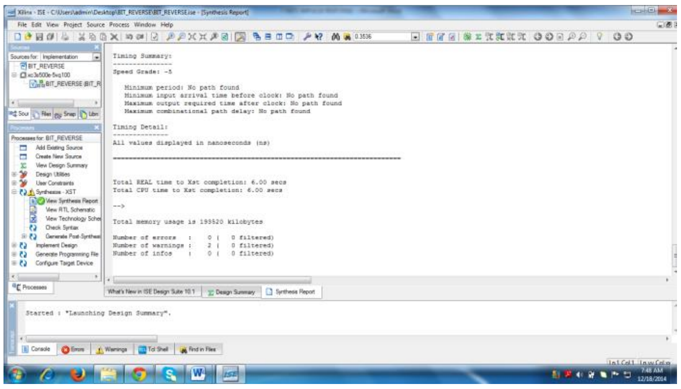
Firstly, the total number of ADD/SUB operators needed to sum up M input terms is M-1.

V. RESULTS AND SIMULATION

Implementation is done with Xilinx ISE tool and vertex Spartan 3e FPGA kit. The code is written in verilog. A simple logic is shown below for the reverse bit level function.

Op = (((Ip>>i)&1) << (31-i))

The results have shown the speed increased by 21% the figure 6 shows the timing results. Figure 7 shows the RTL view of the reverse bit level function.



```
Timing Summary:
---------------
Speed Grade: -5

   Minimum period: No path found
   Minimum input arrival time before clock: No path found
   Maximum output required time after clock: No path found
   Maximum combinational path delay: No path found

Timing Detail:
--------------
All values displayed in nanoseconds (ns)

=========================================================================


Total REAL time to Xst completion: 6.00 secs
Total CPU time to Xst completion: 6.00 secs

-->

Total memory usage is 193520 kilobytes

Number of errors   :    0 (   0 filtered)
Number of warnings :    2 (   0 filtered)
Number of infos    :    0 (   0 filtered)
```

```
Started : "Launching Design Summary".
```

Figure 6 Timing details

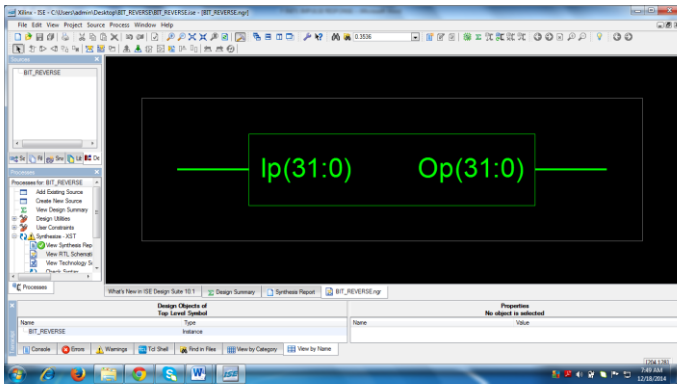


Figure 7 RTL view of reversible bit level function

VI. CONCLUSION

We have implemented the Finite impulse response filter by using the bit level transformation and optimization. Although the designs mostly based on the direct form of FIR filter, we have considered the transposed form of FIR filter at which the results have shown that speed is increased by 21% comparing to the other techniques.

REFERENCES

[1] Yu Pan and Pramod Kumar Meher, "Bit-Level Optimization of Adder-Trees for Multiple Constant Multiplications for Efficient FIR Filter Implementation" IEEE transactions on circuits and systems—i: regular papers, vol. 61, no. 2, february 2014

[2] Deepshikha Bharti #1, K. Anusudha*2, "High Speed FIR Filter Based on Truncated Multiplier and Parallel Adder" International Journal of Engineering Trends and Technology (IJETT) – Volume 5 Number 5 - Nov 2013.

[3] Hsiao S.F,Zhang Jian J.H, and Chen C.H, MAY 2013, Low-Cost FIR Filter Designs Based on Faithfully Rounded Truncated Multiple Constant Multiplication/Accumulation, VOL. 60, NO. 5 , pp no 287-292

[4] Senthilkumar, M.Ramani.S, " FPGA Implementation of Digital Fir Filter Based On Truncated Multiplier" IOSR Journal of Electronics and Communication Engineering(IOSR-JECE) e-ISSN: 2278-2834, p-ISSN: 2278-8735 PP 44-49

[5] "International Technology Roadmap for Semiconductors, 2007 Edition", Semiconductor Industry Association.

[6] http://www.xilinx.com, Xilinx Website.

[7] Jiyu Zhang*, Zhiru Zhang+, Sheng Zhou+, Mingxing Tan*, "Bit-Level Transformation and Optimization for Hardware Synthesis of Algorithmic Descriptions".