Partial LUT Size Analysis in Distributed Arithmetic FIR Filters on FPGAs

Martin Kumm, Konrad Möller and Peter Zipf
Digital Technology Group
University of Kassel, Germany
Email: {kumm, konrad.moeller, zipf}@uni-kassel.de

***Abstract*—Distributed arithmetic is a popular method for implementing digital FIR filters on FPGAs. One essential optimization method is the division of large look-up tables (LUTs) into smaller partial LUTs by using additional adders. Previous work indicates, that the size of these partial LUTs should be chosen to the LUT input size of the FPGA which was 4 for a long time. Nowadays, modern FPGAs offer 6-input LUTs which can be configured to two 5-input LUTs with shared inputs. This paper investigates the optimal input size of partial LUTs on FPGAs with 4-input and 5/6-input LUTs. On FPGAs with 4-input LUTs, it turnes out that only in 62% of the cases (out of 220), a LUT input size of 4 leads to the best implementation. However, the slice overhead is 6.3% on average for the other cases. On FPGAs with 5/6-input LUTs, the least slice overhead (10% on average) is paid when the LUT input size is chosen to 6. However, it was shown that a resource reduction of up to 32% can be achieved when all input sizes in the range 4...7 are evaluated. Using the best partial LUT size, slice reductions of over 50% on average compared to Xilinx Coregen could be achieved for Virtex 6 FPGAs.**

I. Introduction

Field programmable gate arrays (FPGAs) play a key role in real-time digital signal processing (DSP) of signals with frequency components in range of several MHz. They provide the flexibility and fast reconfiguration like digital signal processors combined with a much higher computational power which can only be surpassed by application specific integrated circuits (ASICs). Finite impulse response (FIR) filters are a fundamental component in most DSP applications. The most complex part in an FIR filter is the multiplication with several filter coefficients. Different methods can be used to realize these multiplications on FPGAs:

a) direct implementation using multipliers
b) distributed arithmetic (DA) [1]–[7]
c) multiple constant multipliers (MCM) which are realized by additions, subtractions and bit shifts [8]–[13]

Method a) maps well to FPGAs which provide enough embedded multipliers. In Method b), the multiplications are realized using look-up tables (LUTs) which can be easily mapped to LUT based FPGAs. Originally, DA was proposed as sequential processing algorithm (sequential DA) but was later parallelized for higher throughput (parallel DA). Many general improvements of DA were proposed over the years [4], which have been adopted to FPGA realizations too [5], [6]. In the recent years, it was shown that multiplier blocks using add, subtract and shift operations (method c) consume less logic resources compared to parallel DA implementations [9]–[12]. Originally, these methods were designed for ASICs but also map well to the carry-chain arithmetic of FPGAs. To achieve the same speed as parallel DA they have to be pipelined when used on FPGAs [11]–[13].

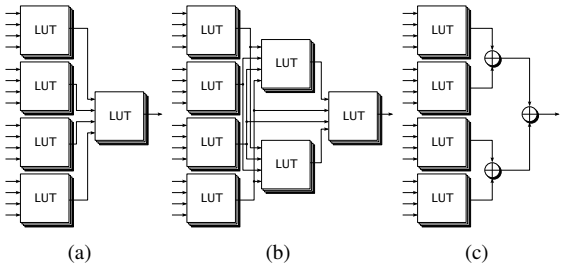


Fig. 1. Example for mapping large LUTs in distributed arithmetic to FPGAs (a-b) using logic only (c) using carry-chain arithmetic

However, DA is able to outperform the other methods for two important kinds of application: Very compact sequential FIR filters and reconfigurable FIR filters. Sequential FIR filters can be used wherever the sampling frequency is much smaller than the systems clock frequency, e. g., in later stages of decimation filters. Sequential DA filters are very compact and avoid large coefficient multiplexers which were necessary for multiplier-based implementations. Reconfigurable FIR filters allow to change the transfer function during runtime. With the introduction of dynamically reconfigurable LUTs in Xilinx Virtex 5...7 and Spartan 6 FPGAs (CFGLUT5 primitive [14]), a reconfigurable DA FIR filter is possible with low additional cost [7]. Compared to time-multiplexed MCM cores [15] which can be switched between a small set of coefficients, the reconfigurable DA allows *any* number of coefficient sets which are only constrained by the configuration memory.

As the LUT storage requirements grow exponentially with the number of filter taps N, a common storage reduction method is to divide the LUT into several smaller partial LUTs by using additional adders [4]. The input size L of these partial LUTs introduces an additional degree of freedom. Clearly, large L lead to large combinational blocks which require many FPGA LUTs while low L require additional carry-chain resources. As the implementation of combinatorial blocks to FPGA LUTs heavily depends on the logic optimization, no analytic estimation for the optimal size of L can be given. To illustrate this, examples of mapping a 16-input LUT to several 4-input LUTs are shown in Fig. 1. Fig. 1(a) shows the best

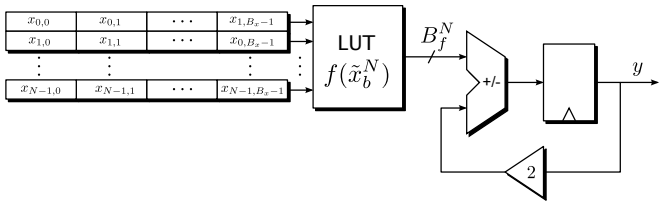


Fig. 2. Sequential realization of a distributed arithmetic FIR filter

case, where the 16-input LUT could be separated into five 4-input LUTs by logic optimization. Another 16-input LUT could require more FPGA LUTs as shown in Fig. 1(b). A DA mapping with $L = 4$ is shown in Fig. 1(c). Clearly, sometimes mapping to logic (FPGA LUTs) and sometimes mapping to the FPGA carry-chain arithmetic leads to the least resources.

Synthesis results using $L \in \{2, 4, 8\}$ of four filter instances were presented in the work of Meher et al. [6]. The results indicate that $L = 4$ is the best choice for FPGAs providing 4-input LUTs. As mentioned above, modern FPGAs (e. g., Altera Stratix II-V and Xilinx Virtex 5…7 and Spartan 6) provide 6-input LUTs which can be divided into two 5-input LUTs with (partly) shared inputs. This complicates the choice for the best value of L. Hence, the main contribution of this paper is a detailed analysis about the optimal input size of partial LUTs for FPGAs with 4-input and 5/6-input LUTs, respectively.

II. Distributed Arithmetic

The fundamental operation of a digital filter with N taps is the inner product of two vectors which can be represented as a sum-of-products of its components

$$y = \mathbf{c} \cdot \mathbf{x} = \sum_{n=0}^{N-1} c_n \, x_n \tag{1}$$

where c_n are usually constants and x_n are the time-shifted input samples. If each x_n is represented as a binary B_x bit 2'th complement number, where $x_{n,b}$ denotes the b'th bit of x_n, (1) can be rewritten to

$$y = \sum_{n=0}^{N-1} c_n \left(\sum_{b=0}^{B_x-2} 2^b x_{n,b} - 2^{B_x-1} x_{n,B_x-1} \right) \tag{2}$$

$$= \sum_{b=0}^{B_x-2} 2^b \underbrace{\sum_{n=0}^{N-1} c_n x_{n,b}}_{=f(\tilde{x}_b^N)} - 2^{B_x-1} \underbrace{\sum_{n=0}^{N-1} c_n x_{n,B_x-1}}_{=f(\tilde{x}_{B_x-1}^N)} \tag{3}$$

where $\tilde{x}_b^N = (x_{0,b}, \ldots, x_{N-1,b})^T$ is a bit vector of length N containing the b'th bit of each element of $\mathbf{x}$. The function

$$f(\tilde{x}_b^N) = \sum_{n=0}^{N-1} c_n x_{n,b} \tag{4}$$

can be precomputed and stored in a single N-input LUT. The storage requirement of the LUT is $B_f^N \, 2^N$ bit, where B_f^N denotes the output word size of $f(\tilde{x}_b^N)$. The inner product can now be obtained by accumulating the shifted outputs of the LUT according to (3). A sequential realization of (3) which computes a valid output every N samples is shown in Fig. 2. For higher throughput, a parallel implementation can be obtained by unfolding the structure as shown in Fig. 3.

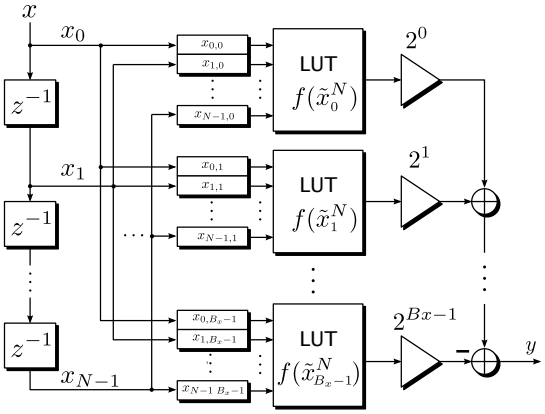


Fig. 3. Parallel realization of a distributed arithmetic FIR filter

III. Optimization Methods for DA

For our analysis, we used common DA optimization methods to get state-of-the-art filter implementations. These methods are described in the following.

A. Coefficient Symmetry

Linear phase FIR filters always have a symmetry of the form

$$c_n = \pm c_{N-n-1} \tag{5}$$

in their coefficients. This can be used in parallel DA to nearly halve the number of LUT inputs [16]. For even N, (1) can be rewritten to

$$y = \sum_{n=0}^{N/2-1} c_n \underbrace{(x_n \pm x_{N-n-1})}_{=z_n}, \tag{6}$$

and for odd N, (1) results in

$$y = \sum_{n=0}^{(N-1)/2-1} c_n \underbrace{(x_n \pm x_{N-n-1})}_{=z_n} + c_{(N-1)/2} \underbrace{x_{(N-1)/2}}_{=z_{M-1}} \tag{7}$$

Due to the reduction of N, the sum terms in (1) are reduced to $M = \lceil \frac{N}{2} \rceil$ terms in (6) and (7). Hence, the LUT storage requirement for a parallel DA filter is reduced from $B_x \, B_f^N \, 2^N$ to $(B_x + 1) B_f^N \, 2^{\lceil \frac{N}{2} \rceil}$ (the bit width B_x increases by one due to the addition/subtraction).

B. Dividing LUTs Into Smaller Partial LUTs

The size of the LUT can be further reduced by splitting the sum in (4) into several smaller sums [4]

$$f(\tilde{x}_b^N) = \sum_{l=0}^{\lfloor N/L \rfloor - 1} \underbrace{\sum_{n=lL}^{(l+1)L-1} c_n x_{n,b}}_{=f_l(\tilde{x}_b^L)} + \underbrace{\sum_{n=N-L'}^{N-1} c_n x_{n,b}}_{=f_{\lfloor N/L \rfloor}(\tilde{x}_b^{L'})} \tag{8}$$

with $L < N$ where $f_l(\tilde{x}_b^L)$ can be realized by partial L-input LUTs. If N is not dividable by L, one additional partial LUT

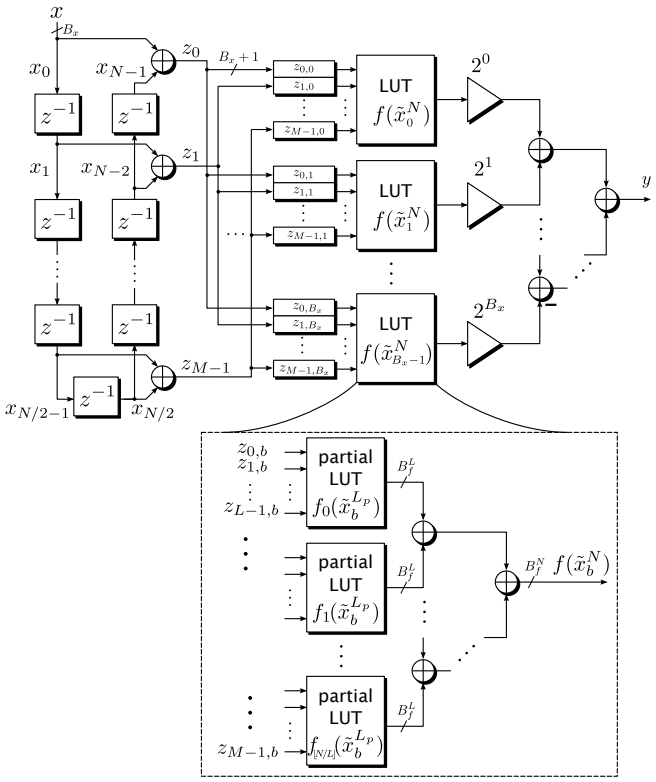


Fig. 4. Optimized parallel realization of a distributed arithmetic FIR filter

of size $L' = N \mod L$ is necessary, which is represented by the last term in (8). The storage requirement of that parallel DA implementation is $\lfloor N/L \rfloor B_x B_f^L 2^L + B_x B_f^{L'} 2^{L'}$ bit. Note that for a fixed L, this realization style grows linear with the number of filter taps N in contrast to (4) which grows exponentially. Furthermore, B_f^L is typically less than B_f^N as fewer sum terms are involved in $f_l(\tilde{x}_b^L)$ of (8). However, this large memory reduction is paid by $\lfloor N/L \rfloor$ additional adders.

The optimization methods described in the last and the current section can be independently combined. In addition, the adder chain can be realized as an adder tree. The resulting structure for even N is shown in Fig. 4. All adders are pipelined with a single register. Note that most shift operations can be moved towards the output to reduce the word size of the adder tree (not shown in Fig. 4). This structure is used as state-of-the-art base for our analysis.

IV. Experiments

We performed two synthesis experiments, both with an FPGA with 4-input LUTs (Xilinx Virtex 4, XC4VSX25-10FF668-10) and an FPGA with 5/6-input LUTs (Xilinx Virtex 6, XC6VLX75T-2FF484-2). All synthesis results were obtained after place & route using Xilinx ISE v13.4. For that, a VHDL code generator was written for the optimized DA structure described above. In the first experiment we used a benchmark set of nine filters which were already used in previous work [10]–[12]. This experiment was done to observe the resource usage for a wide range of input sizes of partial LUTs from $L = 2 \ldots 9$. Furthermore, it was used

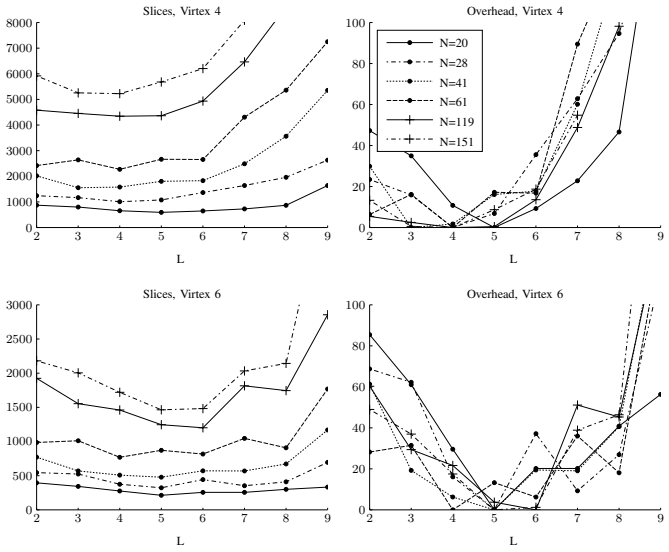


Fig. 5. Synthesis results for benchmark filter of [10]–[12] for Virtex 4 with 4-input LUTs (top) and Virtex 6 with 5/6-input LUTs (bottom) with absolute number of slices (left) and percentage slice overhead (right)

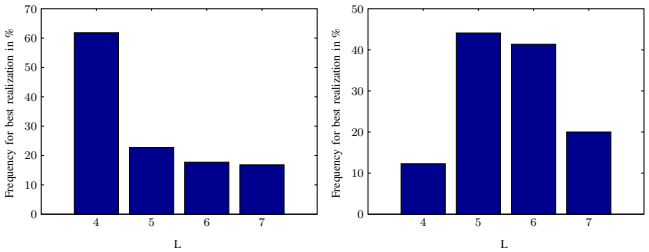
TABLE I
Virtex-4 Results (best results marked bold)

N	Optimized DA L_{best}	Slices	$f_{\max}$ [MHz]	$\frac{\text{Slices}}{f_{\max}}$	Coregen Slices	$f_{\max}$ [MHz]	$\frac{\text{Slices}}{f_{\max}}$
6	4-7	**310**	**332**	**0.94**	368	312	1.18
10	5-7	**420**	275	1.53	509	**411**	**1.24**
13	4	**477**	256	1.86	702	**392**	**1.79**
20	5	**592**	250	**2.37**	1032	**338**	3.05
28	4	**1007**	246	**4.09**	1572	**330**	4.76
41	4	**1580**	243	**6.50**	2199	**324**	6.79
61	4	**2273**	240	**9.47**	3247	**331**	9.81
119	4	**4344**	221	**19.66**	6182	**308**	20.07
151	4	**5225**	232	**22.52**	7873	**313**	25.15
avg.:		**1803**	255	**7.07**	2631	**340**	8.21

TABLE II
Virtex-6 Results (best results marked bold)

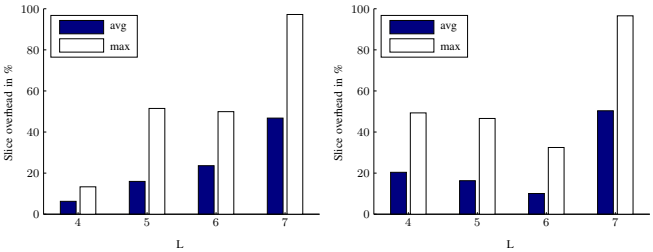
N	Optimized DA L_{best}	Slices	$f_{\max}$ [MHz]	$\frac{\text{Slices}}{f_{\max}}$	Coregen Slices	$f_{\max}$ [MHz]	$\frac{\text{Slices}}{f_{\max}}$
6	4-7	**114**	**599**	**0.19**	182	499	0.36
10	5-7	**164**	**585**	**0.28**	271	474	0.57
13	7	**170**	**560**	**0.30**	302	429	0.70
20	5	**213**	415	**0.51**	453	**456**	0.99
28	5	**323**	392	**0.82**	655	**413**	1.59
41	5	**478**	344	**1.39**	1004	**457**	2.20
61	4	**769**	393	**1.97**	1391	**411**	3.38
119	6	**1201**	334	**3.60**	2693	**352**	7.65
151	5	**1464**	**360**	**4.07**	3574	306	11.68
avg.:		**544**	**442**	**1.23**	1169	**422**	3.24

to demonstrate that resources can be reduced by choosing the best L. For comparison, DA cores were created using the FIR Compiler v5.0 tool (newer versions lead to faulty cores) of Xilinx Coregen [16]. As in previous work, the input bit width



(a) Virtex 4 (4-input LUT) (b) Virtex 6 (5/6-input LUT)

Fig. 6. Histogram of the partial LUT size L leading to the best solution out of 220 filters



(a) Virtex 4 (4-input LUT) (b) Virtex 6 (5/6-input LUT)

Fig. 7. Maximum (white) and average (blue) percentage slice overhead for different partial LUT sizes L

was chosen to $B_x = 12$. The synthesis results (number of slices) for different L are shown on the left side of Fig. 5 for Virtex 4 (top) and Virtex 6 (bottom). A clear minimum can be observed for L values which are close to the FPGA LUT input size. For a better comparison, the relative percentage overhead compared to the best input size L_{best} (in terms of minimal slice resources)

$$\text{overhead}(L) = 100 \cdot \text{slices}(L)/\text{slices}(L_{\text{best}}) \tag{9}$$

is plotted on the right side of Fig. 5. It shows that choosing the wrong L can quickly lead to large slice increases. Numeric results for the proposed structure for $L = L_{\text{best}}$ as well as Coregen results are listed in Table I and Table II. On average, lower slices and a lower slice per f_{max} could be achieved compared to Coregen.

In the second experiment, a large set of filters with $N = 7 \ldots 61$ taps was used. For each N, a low-pass, high-pass and band-pass impulse response, as well as a random sequence was computed, resulting in 220 different filters. Each filter was generated for $L = 4 \ldots 7$. The best L for each of the 220 filters are shown as histogram in Fig. 6. Note that an equal slice complexity for different L leads to multiple L_{best} leading to a total cover of 119% and 118% for Virtex 4 and Virtex 6, respectively. For Virtex 4, the best results are achieved with $L = 4$ in most of the cases (62%). This verifies the results of Meher et al. [6]. However, this also means that in 38% of the cases, a better result would be achieved using a different input size. For Virtex 6, the best LUT input size is either 5 or 6. In 85% of the cases, the $L = 5$ (44%) or $L = 6$ (41%) led to the best result. In most of the cases where the best choice of L was a smaller value than the FPGA LUT input size, also $L > L_{\text{best}}$ was the best choice. Setting L to a larger value than the FPGA LUT input size is sometimes advantageous when the LUTs can be reduced due to logical optimization.

To determine the influence of L to the resource consumption, the slice overhead as defined in (9) was evaluated. Fig. 7 shows the maximum and average slice overhead for Virtex 4 and Virtex 6. For Virtex 4, always choosing $L = 4$ leads to the least overhead which is 6.3% on average and maximally 13.3%. This is a clear indicator, that $L = 4$ is sufficient for most applications on FPGAs with single 4-input LUTs. For Virtex 6 the situation is slightly different. Here, always choosing $L = 6$ leads to the least overhead which is 10% on average and maximally 32.5%.

V. CONCLUSION

The influence of the partial LUT input size L in relation to the FPGA complexity was analyzed. It was shown that L should be chosen close to the input size of the FPGA LUTs. Choosing $L = 4$ and $L = 6$ leads to the least overhead of 6.3% and 10% on average for Virtex 4 and 6, respectively. However, the maximal overhead can be much higher and many resources can be saved by exploring the very small design space of $L = 4 \ldots 7$. By doing so, slice per f_{max} reductions of 14% and 62% could be achieved compared to Coregen for Virtex 4 and Virtex 6, respectively. Thus, there is much optimization potential in Coregen, especially for the latest FPGAs.

REFERENCES

[1] A. Crosisier, D. J. Esteban, M. E. Levilio, and V. Riso, "Digital Filter for PCM Encoded Signals," 1973.

[2] S. Zohar, "New hardware realizations of nonrecursive digital filters," *Computers, IEEE Transactions on*, vol. 22, no. 4, pp. 328–338, 1973.

[3] A. Peled and B. Liu, "A New Hardware Realization of Digital Filters," *IEEE Trans. on Acou., Sp. and Sig. Proc.*, vol. 22, no. 6, Dec. 1974.

[4] S. A. White, "Applications of distributed arithmetic to digital signal processing: a tutorial review," *IEEE ASSP Mag.*, vol. 6, no. 3, 1989.

[5] W. Sen, T. Bin, and Z. Jim, "Distributed Arithmetic for FIR Filter Design on FPGA," in *Communications, Circuits and Systems, 2007. ICCCAS 2007. International Conference on*, 2007, pp. 620–623.

[6] P. Meher, S. Chandrasekaran, and A. Amira, "FPGA Realization of FIR Filters by Efficient and Flexible Systolization Using Distributed Arithmetic," *Signal Processing, IEEE Transactions on*, vol. 56, no. 7, pp. 3009–3017, 2008.

[7] M. Kumm, K. Möller, and P. Zipf, "Reconfigurable FIR Filter Using Distributed Arithmetic on FPGAs," in *Circuits and Systems, IEEE Int. Sym. on (ISCAS)*, 2013.

[8] Y. Voronenko and M. Püschel, "Multiplierless multiple constant multiplication," *ACM Trans. Alg.*, vol. 3, no. 2, 2007.

[9] U. Meyer-Baese, J. Chen, C. H. Chang, and Dempster, "A Comparison of Pipelined RAG-n and DA FPGA-based Multiplierless Filters," in *IEEE APCCAS 2006*, 2006, pp. 1555 – 1558.

[10] S. Mirzaei, R. Kastner, and A. Hosangadi, "Layout Aware Optimization of High Speed Fixed Coefficient FIR Filters for FPGAs," *Int. Journal of Reconfigurable Computing*, vol. 3, pp. 1–17, Jan 2010.

[11] U. Meyer-Baese, G. Botella, D. Romeroa, and M. Kumm, "Optimization of High Speed Pipelining in FPGA-based FIR Filter Design using Genetic Algorithm," in *SPIE Defense Security+Sensing*, 2012.

[12] M. Kumm and P. Zipf, "High Speed Low Complexity FPGA-based FIR Filters Using Pipelined Adder Graphs," in *Field Programmable Technology, Int. Conf. on (ICFPT)*, 2011.

[13] M. Kumm, M. Faust, P. Zipf, and C.-H. Chang, "Pipelined Adder Graph Optimization for High Speed Multiple Constant Multiplication," in *Circuits and Systems, IEEE Int. Sym. on (ISCAS)*, 2012.

[14] Xilinx, Inc., *Xilinx Virtex-5 Libraries Guide for HDL Designs*, 2009.

[15] P. Tummeltshammer, J. Hoe, and M. Puschel, "Time-Multiplexed Multiple-Constant Multiplication," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Trans. on*, vol. 26, no. 9, Sep. 2007.

[16] Xilinx Inc., *IP LogiCORE FIR Compiler v5.0, DS534*, 2011.