Dynamically Reconfigurable FIR Filter Architectures with Fast Reconfiguration

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

***Abstract*—This work compares two finite impulse response (FIR) filter architectures for FPGAs for which the coefficients can be reconfigured during run-time. One is a recently proposed filter architecture based on distributed arithmetic (DA) and the other is based on a LUT multiplication scheme. Instead of using the common internal configuration access port (ICAP) for reconfiguration which is able to change the logic as well as the routing, it is sufficient to reconfigure only the logic in the regarded architectures. This is realized by using the configurable look-up table (CFGLUT) primitive of Xilinx that allows reconfiguration times which are orders of magnitudes faster than using ICAP. The resulting FIR filter architectures achieves reconfiguration times of typically less than 100 ns. They can be reconfigured with arbitrary coefficients that are only limited by their length and word size. As their resource consumptions depend on different parameters of the filter, a detailed comparison is done. It turned out that if the input word size is greater than approximately half the number of coefficients, the LUT based multiplication scheme needs less resources than the DA architecture and vice versa.**

I. Introduction

The finite impulse response (FIR) filter is one of the most fundamental components in digital signal processing. Its block schematic is shown in Fig. 1. Due to the high amount of multiply-accumulate (MAC) operations, the computational power of many real-time applications can only by realized by using the parallel nature of of application specific integrated circuits (ASICs) like field programmable gate arrays (FPGAs). To reduce the performance gap between ASICs and FPGAs, digital filters were one of the driving forces to push embedded multipliers or DSP blocks into the FPGA fabric. The price for those fixed coarse-grained blocks are their inflexibility and limited quantity. However, in many applications like for digital filters, the multiplications have to be performed only with constants that may be only reconfigured from time to time which can be used to reduce the complexity. Examples are multi-stage filters for decimation or interpolation like polyphase FIR filters [1] or frequency variable filters as needed in telecommunications, digital audio, medical, radar, sonar and instruments [2].

On ASICs, one has to realize the reconfiguration on a very low-level, e. g., by introducing multiplexers in the circuit to select between different configurations. Several methods for reconfigurable multiplier blocks (ReMB) for reconfigurable FIR filters have been proposed in the last decade [1], [3]–[6]. Of course, these circuits can be also mapped to FPGAs [4], [6], but they are very limited in the number of different filter

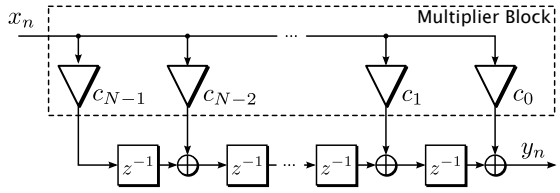


Fig. 1. FIR filter in transposed form

configurations. Furthermore, todays FPGAs provide standard interfaces for reconfiguration like the internal configuration access port (ICAP) of Xilinx FPGAs. With ICAP, a reconfigurable logic region can be defined whose functionality can be exchanged during run-time by any circuit, i. e., the logic functions (LUT content) as well as the routing can be completely exchanged. However, in many applications like in the FIR filter case, the functionality of different configurations is very similar and only a few parameters are exchanged.

A possibility to exchange only the logic without changing the complete routing on modern Xilinx FPGAs, namely the Virtex 5-7, Spartan 6, Kintex 7 and Artix 7, is given by the use of a configurable LUT (CFGLUT). It is very similar to the 16 bit shift register LUTs (SRL16) of older Xilinx FPGAs but has an input word size of five bit instead of four. One advantage of exchanging the logic only is, of course, the reduced reconfiguration memory. But much more important is the fact that there is no bandwidth limitation of a single reconfiguration port like ICAP, as the reconfiguration using CFGLUTs can be done in parallel. Each CFGLUT can be reconfigured in 32 clock cycles where the configuration clock is only limited by routing delays of the FPGA fabric. As each CFGLUT can be reconfigured in parallel, e. g., sourced from many block RAM resources, reconfiguration times in the order of 100 ns can be realized. Compared to ICAP, which has a single 32 bit port operating with up to 100 MHz [7], reconfiguration times are often in the order of microseconds or even milliseconds.

A very generic tool flow for mapping parametrizable circuit configurations to tunable LUTs (TLUTs) like the CFGLUT or SRL16, which is called dynamic circuit specialization, was proposed by Bruneel et al. [8], [9]. Here, the FPGA routing of the circuit is fixed but parameters of the circuit, like the constants of an FIR filter, can be exchanged at run-time by performing a dynamic reconfiguration of the LUT content. However, this work concentrates on hand optimized low-level

implementations.

A reconfigurable FIR filter that uses CFGLUTs was proposed in a recent work of our group [10]. In that work, the FIR filter was realized using distributed arithmetic (DA) [11], [12], as it is naturally realized using LUTs. To map these LUTs to the typically smaller CFGLUTs, a LUT reduction technique [13] was used. Surprisingly, the reconfigurable FIR filters were even more hardware efficient than the fixed coefficient (non-reconfigurable) DA filters generated by Xilinx Coregen (16% less slices on average). The achieved reconfiguration times were 80 ns on average and 106 ns in the worst case.

A totally different reconfigurable FIR filter can be built by replacing each constant multiplier of Fig. 1 with a reconfigurable one. A reconfigurable FPGA multiplier using distributed RAM or block RAM was proposed by Wiatr et al. [14]. It is based on a LUT based multiplication scheme which was introduced by Chapman [15], [16] and later extended by Wirthlin [17].

In both methods, *any* number of coefficient sets can be configured which are only constrained by the configuration memory. However, both methods are based on totally different architectures. Their resource consumptions depend on the input word size, the coefficient word size as well the number of coefficients in a different manner. Thus, the main contribution of this work is a detailed comparison of a reconfigurable FIR filter built from reconfigurable LUT multipliers [14] realized with state-of-the-art CFGLUTs, with the DA based reconfiguration scheme [10]. In addition to that, a detailed comparison with partial reconfiguration using the ICAP interface is done.

The remainder of the paper is organized as follows. In Section II, the CFGLUT is introduced as base for the discussed architectures. Then, the reconfigurable DA filter architecture [10] is introduced in Section III as needed for further comparisons. The reconfigurable LUT based multiplication scheme as well as the corresponding filter architecture are presented in Section IV. A detailed comparison of both architectures is done in Section V, followed by the experimental results and concluding remarks.

II. Run-time Reconfigurable LUTs

The interface of the run-time reconfigurable CFGLUT is shown in Fig. 2. Besides the LUT inputs I0…I4 and outputs O5 and O6, it provides a reconfiguration interface consisting of the signals *configuration data in* (CDI), *configuration data out* (CDO), *clock enable* (CE) and *configuration clock* (CCLK). It can be used as a single 5-input LUT using O6 only or as two 4-input LUTs with shared inputs by setting I4 = 1 and using O5 and O6. In order to perform a change of the output function(s), 32 bit of configuration data have to be clocked into CDI while CE is set to high. The previous configuration is shifted out at CDO which can be used to cascade a series of CFGLUTs. However, each CDI input can be programmed in parallel leading to a reconfiguration time of 32 clock cycles. The speed of the configuration clock is only limited by the FPGA routing fabric. The CFGLUT can be mapped into the FPGA by using the HDL primitive CFGLUT5 [18] which utilize the same slice resources as a standard 6-input LUT. Note that the clock of the output flip flop of that slice (if used) is in the same clock domain as CCLK.

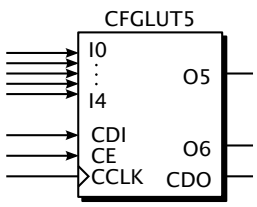


Fig. 2. Block diagramm of the CFGLUT5 primitive

Back portability to older Xilinx FPGA architectures can be achieved by using the SRL16 primitive (and adjusting the input word size). Alternatively, block RAMs which are available on nearly every modern FPGA can also be used as reconfigurable LUTs. Then, additional resources are needed for address counter and data multiplexers (if a single port RAM is used) [14].

III. Distributed Arithmetic FIR Architecture

The fundamental operation of an FIR filter is the inner product of two vectors,

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

where vector $\mathbf{x}$ consists of time shifted scalars of the filter input and vector $\mathbf{c}$ consists of the coefficients. When each input sample x_n is coded in two's complement format

$$x_n = \sum_{b=0}^{B_x-2} 2^b x_{n,b} - 2^{B_x-1} x_{n,B_x-1} \tag{2}$$

where $x_{n,b}$ denotes the b'th bit of x_n, we can insert (2) in (1) and exchange the two resulting sums:

$$y_n = \sum_{n=0}^{N-1} c_n \left(\sum_{b=0}^{B_x-2} 2^b x_{n,b} - 2^{B_x-1} x_{n,B_x-1} \right) \tag{3}$$

$$= \sum_{b=0}^{B_x-2} 2^b \underbrace{\sum_{n=0}^{N-1} c_n x_{n,b}}_{=f(\tilde{x}_b^N)} - 2^{B_x-1} \underbrace{\sum_{n=0}^{N-1} c_n x_{n,B_x-1}}_{=f(\tilde{x}_{B_x-1}^N)} \tag{4}$$

Now, the inner product can be represented as a sum of bit shifted results of the function

$$f(\tilde{x}_b^N) = \sum_{n=0}^{N-1} c_n x_{n,b} \ . \tag{5}$$

Function $f(\tilde{x}_b^N)$ only depends on the N-bit vector $\tilde{x}_b^N = (x_{0,b}, x_{1,b}, \ldots, x_{N-1,b})^T$ which contains the b'th bit of the (time shifted) elements of $\mathbf{x}$. It can be precomputed and stored in a single LUT with N inputs.

The storage requirement of the LUT is $B_f^N \cdot 2^N$ bit, where B_f^N denotes the output word size of the N-input LUT $f(\tilde{x}_b^N)$. This output word size depends on the coefficient word size

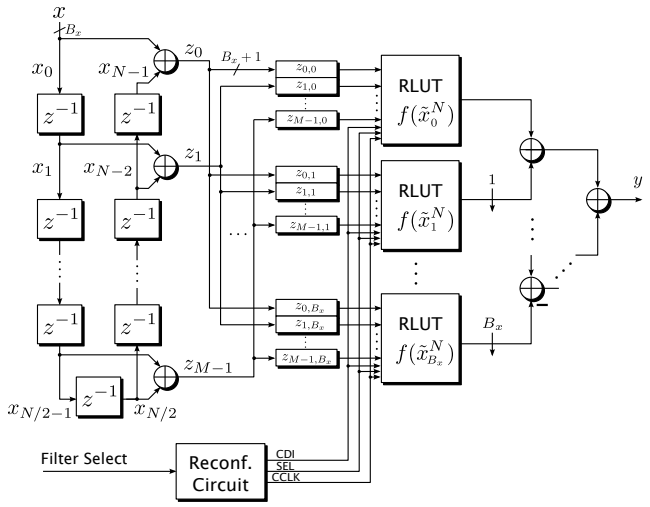


Fig. 3. Architecture of the reconfigurable distributed arithmetic FIR filter

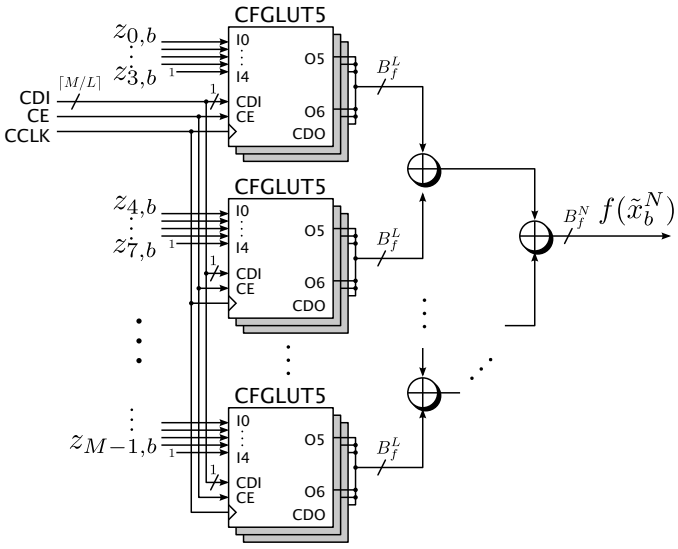


Fig. 4. A reconfigurable LUT realization of $f(\tilde{x}_b^N)$ using single CFGLUT5 primitives

of the given filter. For a maximum coefficient word size of B_c, the maximum LUT output word size can be obtained by evaluating (5) with $x_{n,b} = 1 \; \forall \; n$, leading to

$$B_f^N = B_c + \lceil \log_2(N) \rceil \quad . \tag{6}$$

A. Mapping the LUT into smaller partial LUTs

The input size and, therefore, the memory requirements of the LUT can be reduced by splitting the sum in (5) into several smaller sums

$$f(\tilde{x}_b^N) = \sum_{l=0}^{\lfloor N/L \rfloor - 1} \underbrace{\sum_{n=lL}^{(l+1)L-1} c_n x_{n,b}}_{=f_l(\tilde{x}_b^L)} + \underbrace{\sum_{n=N-L'}^{N-1} c_n x_{n,b}}_{=f_{\lfloor N/L \rfloor}(\tilde{x}_b^{L'})} \tag{7}$$

with $L < N$. Each of the functions

$$f_l(\tilde{x}_b^L) = \sum_{n=lL}^{(l+1)L-1} c_n x_{n,b} \tag{8}$$

can be realized as an L-input LUT. If N is not dividable by L, one additional LUT of input size $L' = N \bmod L$ is necessary, which is represented with the last term in (7).

Now, L can be chosen to fit into the input word size of the FPGA LUT. Choosing $L = 4$ or $L = 5$ allows a direct mapping to CFGLUTs. Furthermore, the LUT storage requirement for the N-input LUT $f(\tilde{x}_b^N)$ is reduced from $B_f^N \cdot 2^N$ bits to $\lfloor N/L \rfloor \cdot B_f^L \cdot 2^L + B_f^{L'} \cdot 2^{L'}$ bits. This memory reduction is paid by $\lfloor N/L \rfloor$ additional adders. Note that for a fixed L, this realization style grows linear with the number of filter taps N in contrast to (5) which grows exponentially. An analytic comparison of the LUT input size revealed that it is always advantageous to use a CFGLUT as two 4-input LUTs instead of a single 5-input LUT [10].

The resulting reconfigurable DA architecture [10] is shown in Fig. 3. Here, an additional pre-processing stage of registers and adders is used to exploit the inherent coefficient symmetry of common linear phase FIR filters. By doing that, the N LUT inputs for x_n are reduced to $M = \lceil \frac{N}{2} \rceil$ inputs of z_n, thus, the storage requirements are approximately halved by introducing $\lfloor N/2 \rfloor$ adder.

All adders in Fig. 3 are followed by pipeline registers (not shown) forming a pipelined adder tree. Furthermore, most of the left shift operations can be moved towards the output of the output adder tree for word size reduction. The reconfigurable LUTs (RLUTs) of Fig. 3 already contain CFGLUTs for reconfiguration which are shown in detail in Fig. 4. Each CFGLUT5 computes two bits of $f(\tilde{x}_b^L)$ which are further processed in a pipelined adder tree (pipeline registers not shown) according to (7).

IV. LUT Multiplier based FIR Architecture

A LUT based realization of (1) can also be achieved by simply mapping each product $c_n \cdot x_n$ to a B_x-input LUT. Then, the storage requirement would be $B_c \cdot 2^{B_x}$ for each LUT. Like for the DA, the LUT content can be represented as a sum of partial products. Hence, a similar LUT reduction method as used in (7) can be applied, leading to the constant multiplication scheme of Chapman [16].

Let us consider one of the $B_x \times B_c$ multiplications of (1). It can be divided into $K = \lceil \frac{B_x}{L} \rceil$ smaller products by rearranging the partial sum terms:

$$\begin{aligned} \underbrace{c_n \cdot x_n}_{B_c \times B_x \text{ mult.}} &= c_n \left(\sum_{b=0}^{B_x-2} 2^b x_{n,b} - 2^{B_x-1} x_{n,B_x-1} \right) \\ &= c_n \sum_{b=0}^{L-1} 2^b x_{n,b} + c_n \sum_{b=L}^{2L-1} 2^b x_{n,b} + \dots \\ &\quad + c_n \left(\sum_{b=(K-1)L}^{KL-2} 2^b x_{n,b} - 2^{KL-1} x_{n,KL-1} \right) \end{aligned}$$

$$= \underbrace{c_n \sum_{b=0}^{L-1} 2^b x_{n,b}}_{=g(\hat{x}^L_{n,l})} + 2^L \underbrace{c_n \sum_{b=0}^{L-1} 2^b x_{n,b+L}}_{=g(\hat{x}^L_{n,l})} + \dots$$

$$\dots + 2^{(K-1)L} \cdot \underbrace{c_n \left(\sum_{b=0}^{L-2} 2^b x_{n,b+(K-1)L} - 2^{L-1} x_{n,KL-1} \right)}_{=g'(\hat{x}^L_{n,l})}$$

$$= \sum_{l=0}^{K-2} 2^{lL} g(\hat{x}^L_{n,l}) + 2^{(K-1)L} g'(\hat{x}^L_{n,l}) \tag{9}$$

Each of the $K-1$ function terms $g(\hat{x}^L_l)$ represents a $B_c \times L$ unsigned multiplication with input vector $\hat{x}^L_{n,l} = (x_{n,lL}, x_{n,lL+1}, \dots, x_{n,lL+L-1})^T$:

$$g(\hat{x}^L_{n,l}) = c_n \cdot \hat{x}^L_{n,l} \tag{10}$$

The function term $g'(\hat{x}^L_l)$ for the MSB represents the same multiplication but with a signed interpretation of the the input vector $\hat{x}^L_{n,l}$ in case a singed multiplication is performed. Thus, $K = \lceil \frac{B_x}{L} \rceil$ multiplications of size $B_c \times L$ are necessary to perform the $B_c \times B_x$ multiplication. In case that B_x is not dividable by L, x_n has to be sign extended to KL bits with $K = \lceil B_x/L \rceil$. Now, each partial product can be realized by an L-input LUT with 2^L precomputed products. Hence, L can be chosen to fit the LUT input size of the specific FPGA. In the reconfigurable multiplier of Wiatr [14], L was chosen to fit the address word size of distributed RAM or block RAM, in this work, L is chosen to the input word size of the CFGLUT. Finally, the bit shifted partial products have to be added. The resulting reconfigurable constant LUT multiplier is shown in Fig. 5. A pipelined adder tree is used to improve the performance (pipeline registers not shown in Fig. 5). The applied pipelining does not lead to much extra resources, as most of the flip-flops (FFs) can be mapped to the otherwise unused FFs of slices realizing full adders. Note that the bit shifts after the LUTs can also be moved to later stages of the adder tree to reduce the word size.

The LUT storage requirements are now reduced from $B_c \cdot 2^{B_x}$ bits to $\lceil \frac{B_x}{L} \rceil B_g^L 2^L$ bits, where

$$B_g^L = B_c + L \ . \tag{11}$$

denotes the output word size of the partial LUTs. As for the reconfigurable DA architecture there are many CFGLUTs with identical inputs, so it is again better to configure the CFGLUT to two independent 4-input LUTs. The precomputed LUT contents are listed in Table I.

Now, several reconfigurable multipliers can be used to build a reconfigurable FIR filter. For that, all the multipliers in the dashed box in Fig. 1 are replaced by reconfigurable ones. Like for the reconfigurable DA architecture, coefficient symmetry is assumed for the proposed design, too. To realize that, only $M = \lceil N/2 \rceil$ of the multipliers are implemented which are shared between two of the so called structural adders shown below the multiplier block in Fig. 1. For negative symmetry, the corresponding structural adders are replaced by subtractors.

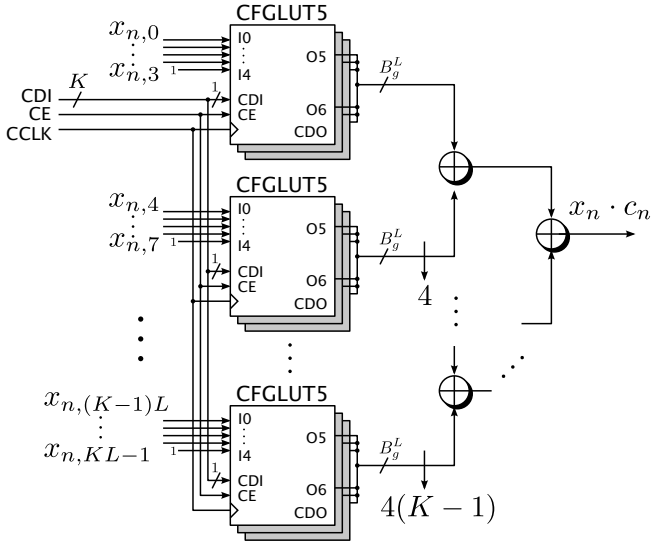


Fig. 5. LUT based constant multiplication with a pipelined adder graph

TABLE I
LUT CONTENTS OF THE CFGLUT MULTIPLIER USING $L = 4$

LUT address	$g'(\hat{x}^L_{n,l})$ (MSB LUT)	$g(\hat{x}^L_{n,l})$ (other LUTs)
0000	$0 \cdot c_n$	$0 \cdot c_n$
0001	$1 \cdot c_n$	$1 \cdot c_n$
...	...	...
0111	$7 \cdot c_n$	$7 \cdot c_n$
1000	$-8 \cdot c_n$	$8 \cdot c_n$
1001	$-7 \cdot c_n$	$9 \cdot c_n$
...	...	...
1111	$-1 \cdot c_n$	$15 \cdot c_n$

V. COMPARISON OF FIR ARCHITECTURES

A. Resource Consumption

To estimate the FPGA resource consumption the required CFGLUTs and adders are evaluated for both architectures. It can be taken as indicator for selecting the architecture for a given filter, which is specified with the number of taps (N), the input word size (B_x) and the coefficient word size (B_c). For the reconfigurable DA architecture $B_x + 1$ M-input LUTs are required, where each of them contains $\lceil M/L \rceil$ L-input LUTs with an output word size of B_f^L (see Fig. 3 and Fig. 4). Using $L = 4$, two output bits can be computed with a single CFGLUT, leading to a total number of CFGLUTs of

$$\begin{aligned} N_{\text{CFGLUT,DA}} &= (B_x + 1) \lceil M/4 \rceil \lceil B_f^4/2 \rceil \\ &= (B_x + 1) \lceil M/4 \rceil \lceil B_c/2 + 1 \rceil \ . \end{aligned} \tag{12}$$

In contrast to the reconfigurable DA filter, the LUT multiplier FIR architecture consists of M multipliers, realized as B_x-input LUT. Each of the B_x-input LUT is built from $\lceil B_x/L \rceil$ L-input LUTs with an output word size of B_g^L. When two output bits are computed with a single CFGLUT by setting

$L = 4$, the total number of CFGLUTs is

$$N_{\text{CFGLUT,LUTM}} = M \lceil B_x/4 \rceil \left\lceil B_g^4/2 \right\rceil \tag{13}$$
$$= M \lceil B_x/4 \rceil \lceil B_c/2 + 2 \rceil \ . \tag{14}$$

It can be seen that the required CFGLUTs are very similar for both architectures. To estimate the architecture with the least CFGLUTs, we assume that M and B_x are both dividable by four, and B_c is dividable by two. Then, the LUT multiplier architecture needs less CFGLUTs compared to the DA architecture when the following unequation is true:

$$N_{\text{CFGLUT,LUTM}} < N_{\text{CFGLUT,DA}}$$
$$MB_x/4(B_c/2 + 2) < (B_x + 1)M/4(B_c/2 + 1)$$
$$B_x < B_c/2 + 1 \tag{15}$$

Hence, if the input word size B_x is less than approximately half the coefficient word size B_c, the LUT multiplier architecture needs less CFGLUTs and vice versa.

Besides the used CFGLUTs a large amount of adders are necessary. For the DA FIR architecture M adders are necessary in the pre-processing stage as well as an adder tree with B_x adders in the post-processing (see Fig. 3). Each of the $B_x + 1$ RLUTs consist of $\lceil M/L \rceil$ adders, resulting in

$$N_{\text{ADD,DA}} = M + B_x + (B_x + 1) \lceil M/4 \rceil \ . \tag{16}$$

For the LUT multiplier architecture, $N-1$ structural adders are needed to compute the filter output from the multiplier results (see Fig. 1). Each of the M reconfigurable multipliers consists of $\lceil B_x/L \rceil$ adders, resulting in

$$N_{\text{ADD,LUT}} = N - 1 + M \lceil B_x/4 \rceil \ . \tag{17}$$

To give an estimate for the architecture with the least adders, we assume that M and B_x are both dividable by 4 and $N = 2M$. Then, the LUT multiplier architecture needs less adders when the following unequation is true:

$$N_{\text{ADD,LUT}} < N_{\text{ADD,DA}}$$
$$2M - 1 + MB_x/4 < M + B_x + (B_x + 1)M/4$$
$$3M - 4 < 4B_x \tag{18}$$

As each adder typically consists of several slices, where each slice contains up to four CFGLUTs, the adder estimation is much more significant than the CFGLUT estimation. Hence, as a rule of thumb, if the input word size is greater than approximately half the number of coefficients, the LUT multiplier architecture needs less resources and vice versa.

B. Configuration Memory

With configuration memory we mean the storage requirements for a single configuration. Thus, it has to be multiplied by the number of different configurations to get the total memory requirement. For the DA architecture all RLUTs shown in Fig. 3 have the same content. As each of the CFGLUT needs 32 bit of configuration data, the storage requirement is

$$S_{\text{DA}} = 32 \cdot \lceil M/4 \rceil \lceil B_c/2 + 1 \rceil \ \text{bit} \ . \tag{19}$$

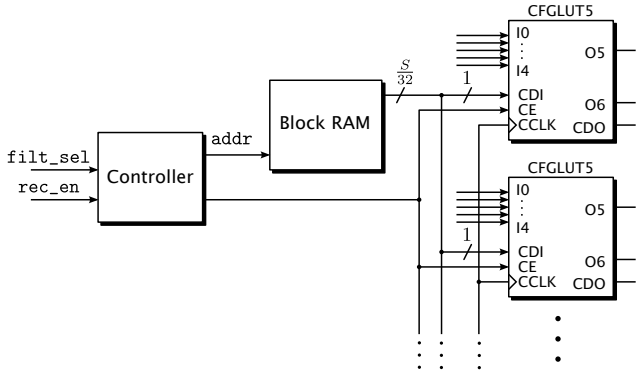


Fig. 6. Control architecture for the reconfiguration of CFGLUTs

For the LUT multiplier based architecture, all partial product LUTs except the MSB LUT can use the same configuration data, as they all represent the multiplication of one 4 bit input with the same coefficient. Only the MSB LUT has to be reconfigured with a different configuration because of the sign bit for signed multiplication. Each partial product has an output word size of B_g^L, so the storage requirement is

$$S_{\text{LUTM}} = 64 \cdot M \lceil B_c/2 + 2 \rceil \ \text{bit} \ . \tag{20}$$

As it can be seen, the required storage is independent of the input word size B_x and the influence of the coefficient word size is $B_c/2$ for both architectures. But for the LUT multiplier based architecture, the memory requirement is approximately eight times higher than for distributed arithmetic. Thus, the reconfigurable DA architecture is always the best in terms of memory requirements.

VI. Reconfiguration Circuit

The reconfiguration circuit can be built from a block RAM or distributed RAM and a simple controller. It is shown in Fig. 6, together with two CFGLUTs. An input vector `filt_sel` is used which can arbitrarily select a filter coefficient set and an enable signal `rec_en` is used to start the reconfiguration. Then, a 5 bit counter is used to address a sequence of 32 reconfiguration words. The RAM address can then be composed of the `filt_sel` which addresses the higher bits together with the five output bits of the counter to address the lower bits by a simple concatenation.

Note that during the reconfiguration process, the output of the multiplier is not valid. This problem can be fixed by a clock enable signal at the output register of the multiplier, which leads to a neglect of the input during reconfiguration. Alternatively each CFGLUT can be doubled as it was proposed in previous work [10]. It can be reconfigured with the new coefficient while the other CFGLUT is processing data. A multiplexer is used to switch between the two CFGLUTs to achieve a glitch free data processing.

VII. Results

Two synthesis experiments were made to evaluate the resource usage, speed and reconfiguration times of the different

architectures. In the first experiment, the LUT based multiplier architecture as well as the DA architecture are synthesized for different filter lengths and input word sizes. In the second experiment, we used ICAP for the partial reconfiguration of highly optimized filter instances to compare the resource overhead and configuration times. A VHDL code generator was developed in Matlab for the different architectures. All synthesis results were obtained after place & route using Xilinx ISE v13.4 with the design goal 'speed' for a Virtex 6 FPGA (XC6VLX75T-FF784).

A. Comparison of CFGLUT based Architectures

To compare the two discussed architectures, several different reconfigurable filters were synthesized. To ease the comparison with other work, we used a benchmark set of nine filters with $N = 6$ up to $N = 151$ coefficients which were already used in previous publications [10], [19]–[21]. The coefficients have a maximum word size of $B_c = 17$ bit and are available online as MIRZAEI10_N [22]. The input word size was varied from $B_x = 8 \ldots 32$ bit in steps of 8 bit. Of course, any filter of the same symmetry up to B_c, B_x and N can be reconfigured. Lower word sizes have to be sign extended while unused coefficients have to be set to zero. The same clock was used for the filter as well as for reconfiguration.

Detailed synthesis results are listed in Table II, which shows the configuration memory per instance (S), the number of Slices, the maximum clock frequency f_{clk} as well as the reconfiguration time $T_{\text{rec}} = 32/f_{\text{clk}}$ for both methods. On average, both architectures achieve nearly equally low reconfiguration times of 69.94 ns and 65.94 ns and fast clock frequencies of 472.4 MHz and 494.2 MHz, respectively. As expected from Section V-B, the reconfiguration memory of the LUT multiplier architecture is approximately eight times higher compared to the DA based architecture. A comparison of the resources is visualized in Fig. 7 which shows the percentage slice improvement of the LUT multiplier based architecture over the DA based architecture. Hence, in case positive values are shown, the LUT architecture is the better choice and vice versa. As expected from Section V-A, the LUT multiplier based architecture performs better for smaller filter instances when $\lceil N/2 \rceil < B_x$. Exceptions from that rule can be explained by the other parts of the filter, e. g., an unbalanced adder tree with $2^{n-1} < x < 2^n$ inputs may need as many slices as a balanced one with $x = 2^n$ inputs due to pipelining. Up to 35% slice reductions can be achieved by using the LUT multiplier architecture for small filters but the DA architecture is still advantageous for larger filter instances requiring up to 40% less slices.

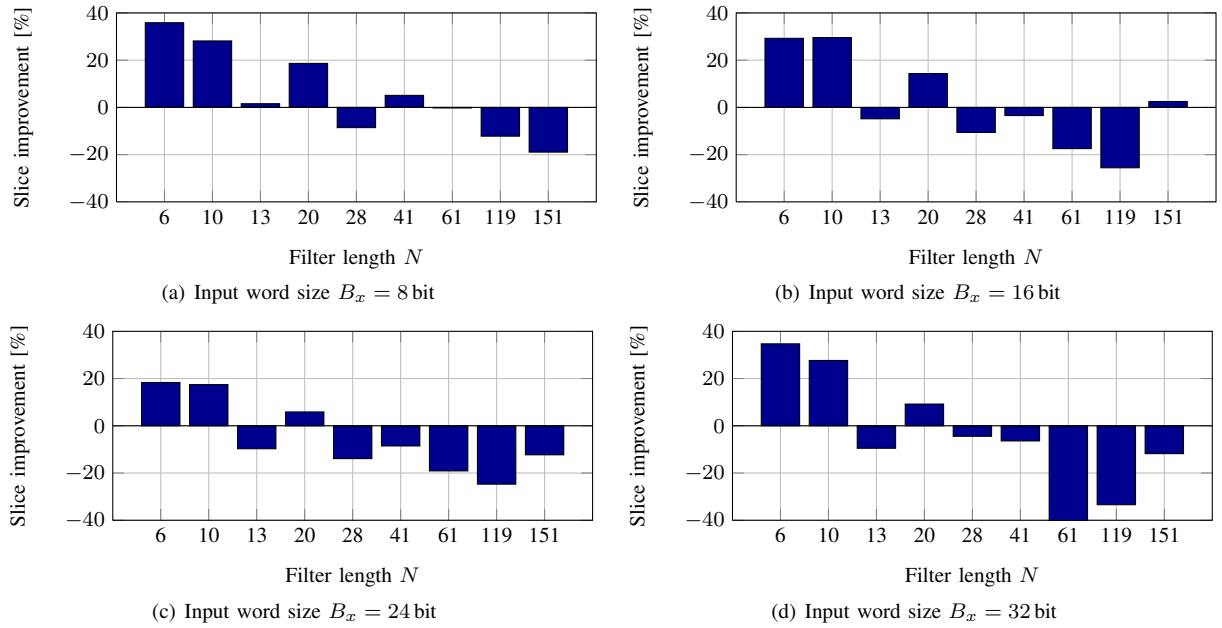
B. Comparison with ICAP

The standard partial reconfiguration scheme supported by the Xilinx ISE tools provides that a reconfigurable region of the FPGA is reserved which can be exchanged by any circuit in run-time using the ICAP interface. This implies that the reconfigurable region is large enough to include the largest configuration. In contrast to the discussed CFGLUT architectures, both logic and routing can be replaced. On the one hand, each configuration can be optimized like a static design which is very resource efficient. On the other hand, the storage requirement is much larger which also leads to longer reconfiguration times due to the sequential ICAP interface. The ICAP interface has a word size of 32 bit and can be operated with up to 100 MHz [7]. However, this speed is a theoretical maximum which can only be achieved with some effort [23]. Reconfiguration clocks of 94.5 MHz were reported by using a block RAM cache that was closely located at the ICAP [24].

To get the maximum performance out of the ICAP approach, we used the reduced pipelined adder graphs (RPAG) algorithm [25] which is a state-of-the-art optimization method for fixed coefficient multiplier blocks as needed for FIR filters. All multiplications by constants are mapped to add/subtract and bit shift operations in a pipelined way. It was shown in previous work [20], [21], [26], [27], that this method outperforms DA based filters as well as LUT based multipliers in the case that input word sizes of more than 12 bit are used. As

TABLE II
SYNTHESIS RESULTS FOR THE PROPOSED RECONFIGURABLE LUT MULTIPLIER FIR FILTER IN COMPARISON WITH RECONFIGURABLE DA FIR FILTER [10]

		Reconf. FIR DA [10]				Reconf. FIR LUT (proposed)			
B_x	N	S [bit]	Slices	f_{clk} [MHz]	T_{rec} [ns]	S [bit]	Slices	f_{clk} [MHz]	T_{rec} [ns]
8	6	320	145	570.5	56.1	2112	93	647.3	49.4
8	10	640	199	492.4	65.0	3520	143	575.0	55.6
8	13	640	191	601.3	53.2	4928	188	608.6	52.6
8	20	960	365	496.3	64.5	7040	297	534.8	59.8
8	28	1280	374	525.2	60.9	9856	406	551.9	58.0
8	41	1920	627	522.7	61.2	14784	595	573.4	55.8
8	61	2560	835	543.2	58.9	21824	837	499.5	64.1
8	119	4800	1487	499.5	64.1	42240	1668	504.0	63.5
8	151	6080	1813	395.4	80.9	53504	2156	415.3	77.1
16	6	320	253	622.7	51.4	2112	179	576.7	55.5
16	10	640	372	552.8	57.9	3520	262	572.4	55.9
16	13	640	353	545.3	58.7	4928	370	522.2	61.3
16	20	960	635	467.7	68.4	7040	544	518.4	61.7
16	28	1280	707	525.8	60.9	9856	782	540.3	59.2
16	41	1920	1071	521.9	61.3	14784	1108	487.8	65.6
16	61	2560	1341	413.7	77.3	21824	1575	463.8	69.0
16	119	4800	2594	348.3	91.9	42240	3257	480.3	66.6
16	151	6080	4256	410.5	78.0	53504	4149	428.8	74.6
24	6	320	371	550.7	58.1	2112	303	536.2	59.7
24	10	640	522	556.2	57.5	3520	431	498.3	64.2
24	13	640	535	470.8	68.0	4928	587	545.6	58.7
24	20	960	939	573.7	55.8	7040	884	531.6	60.2
24	28	1280	1029	416.5	76.8	9856	1172	482.9	66.3
24	41	1920	1558	383.6	83.4	14784	1691	467.1	68.5
24	61	2560	1958	372.0	86.0	21824	2332	435.0	73.6
24	119	4800	4047	355.4	90.0	42240	5047	393.1	81.4
24	151	6080	5802	385.2	83.1	53504	6515	380.7	84.1
32	6	320	490	524.9	61.0	2112	320	512.3	62.5
32	10	640	734	480.3	66.6	3520	531	485.4	65.9
32	13	640	704	507.1	63.1	4928	771	495.8	64.5
32	20	960	1130	517.9	61.8	7040	1026	473.9	67.5
32	28	1280	1419	430.7	74.3	9856	1482	442.9	72.3
32	41	1920	1965	376.5	85.0	14784	2091	446.0	71.7
32	61	2560	2570	358.6	89.2	21824	3601	401.6	79.7
32	119	4800	4889	302.9	105.6	42240	6523	377.2	84.8
32	151	6080	7170	391.1	81.8	53504	8016	386.7	82.8



(a) Input word size $B_x = 8$ bit

(b) Input word size $B_x = 16$ bit

(c) Input word size $B_x = 24$ bit

(d) Input word size $B_x = 32$ bit

Fig. 7. Slice improvement of the reconfigurable FIR filter based on LUT multipliers in comparison with the reconfigurable DA FIR filter

TABLE III
COMPARISON OF A SINGLE FILTER MIRZAEI10_41 WITH $B_x = 16$ BIT USING ICAP RECONFIGURATION AND THE CFGLUT METHODS

Method	S [bit]	Slices	f_{clk} [MHz]	T_{rec} [ns]
RPAG [25] with ICAP	746496	502...569	386.7...448.8	233280
Reconf. FIR DA [10]	1920	1071	521.9	61.3
Reconf. FIR LUT	14784	1108	487.8	65.6

the optimization heavily depends on the numeric coefficient values, ten different filters were designed with the same length as the mid size benchmark filter MIRZAEI10_41 and an input word size of 16 bit. These served as realistic configurations which can be reconfigured via ICAP.

The results are summarized in Table III. The number of slices using RPAG optimized FIR filters varied for the different filter instances from 502...569. Hence, a reconfiguration region with a capacity of 569 slices has to be reserved. The reconfiguration is organized in frames of 80 slices, thus, eight frames have to be reserved where each frame contributes with 93312 bit, leading to a reconfiguration memory requirement of $S_{\text{ICAP}} = 746496$ bit per filter instance. Compared to the CFGLUT-based methods, a factor of 388 and 50 more reconfiguration memory is necessary, respectively. Assuming that the full performance of ICAP can be used, the reconfiguration takes $T_{\text{rec}} = S_{\text{ICAP}}/32 \cdot 10\,\text{ns} = 233\,\mu s$. Thus, compared to the slowest CFGLUT methods with 65.5 ns, the ICAP reconfiguration is a factor of 3556 slower. The price for these fast reconfiguration times and low memory requirements is paid by a slice overhead of 88% and 95%, respectively.

VIII. CONCLUSION

We analyzed two reconfigurable FIR filter architectures based on the CFGLUT primitives which can be mapped to all modern FPGAs of Xilinx. The first one is based on a recently proposed method based on distributed arithmetic [10], the second one uses several instances of a reconfigurable LUT multiplier [14] to build a reconfigurable multiplier block as needed in the FIR filter. Similarities between the different approaches were derived as both methods uses similar arithmetic transformations to map large LUTs to several smaller LUTs by the use of additional adders. It turned out that less CFGLUTs, and, in most of the cases, less slices are needed for the LUT based multiplier architecture in the case that the input word size is greater than approximately half the number of coefficients and vice versa. Both methods have reconfiguration times and memory requirements which are about four orders of magnitudes faster than using partial reconfiguration via the ICAP interface which is paid by approximately twice the amount of slices.

REFERENCES

[1] M. Faust, O. Gustafsson, and C.-H. Chang, "Reconfigurable Multiple Constant Multiplication Using Minimum Adder Depth," in *Conference Record of the Forty Fourth Asilomar Conference on Signals, Systems and Computers (ASILOMAR)*, 2010, pp. 1297–1301.

[2] Lowenborg and Johansson, "Minimax Design of Adjustable-bandwidth Linear-phase FIR Filters," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 53, no. 2, pp. 431–439, 2006.

[3] S. S. Demirsoy, A. Dempster, and I. Kale, "Design Guidelines for Reconfigurable Multiplier Blocks," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2003.

[4] S. S. Demirsoy, I. Kale, and A. Dempster, "Efficient Implementation of Digital Filters Using Novel Reconfigurable Multiplier Blocks," in *Conference Record of the Thirty-Eighth Asilomar Conference on Signals, Systems and Computers*, 2004, pp. 461–464.

[5] P. Tummeltshammer, J. Hoe, and M. Puschel, "Time-Multiplexed Multiple-Constant Multiplication," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 9, pp. 1551–1563, Sep. 2007.

[6] R. Gutierrez, J. Valls, and A. Perez-Pascual, "FPGA-Implementation of Time-Multiplexed Multiple Constant Multiplication based on Carry-Save Arithmetic," in *International Conference on Field Programmable Logic and Applications (FPL)*, 2009, pp. 609–612.

[7] Xilinx, Inc., *Partial Reconfiguration User Guide*, Oct. 2010.

[8] K. Bruneel, W. Heirman, and D. Stroobandt, "Dynamic Data Folding with Parameterizable FPGA Configurations," *Transactions on Design Automation of Electronic Systems*, vol. 16, no. 4, pp. 43:1–43:29, Oct. 2011.

[9] K. Bruneel, "Efficient Circuit Specialization for Dynamic Reconfiguration of FPGAs," Ph.D. dissertation, Universiteit Gent, 2011.

[10] M. Kumm, K. Möller, and P. Zipf, "Reconfigurable FIR Filter Using Distributed Arithmetic on FPGAs," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2013, pp. 2058–2061.

[11] A. Crosisier, D. J. Esteban, M. E. Levilio, and V. Riso, "Digital Filter for PCM Encoded Signals," *United States Patent No. 3777130*, 1973.

[12] S. Zohar, "New Hardware Realizations of Nonrecursive Digital Filters," *IEEE Transactions on Computers*, vol. 22, no. 4, pp. 328–338, 1973.

[13] S. A. White, "Applications of Distributed Arithmetic to Digital Signal Processing: A Tutorial Review," *IEEE ASSP Mag.*, vol. 6, no. 3, 1989.

[14] K. Wiatr and E. Jamro, "Implementation of Multipliers in FPGA Structures," *International Symposium on Quality Electronic Design*, pp. 415–420, 2001.

[15] K. D. Chapman, "Fast Integer Multipliers Fit in FPGAs," *Electronic Design News*, 1994.

[16] K. Chapman, "Constant Coefficient Multipliers for the XC4000E," *Xilinx Application Note*, 1996.

[17] M. J. Wirthlin, "Constant Coefficient Multiplication Using Look-Up Tables," *Journal of VLSI Signal Processing*, vol. 36, no. 1, pp. 7–15, Jan. 2004.

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

[19] 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. 2010, pp. 1–17, 2010.

[20] M. Kumm and P. Zipf, "High Speed Low Complexity FPGA-based FIR Filters Using Pipelined Adder Graphs," in *International Conference on Field-Programmable Technology (FPT)*, 2011, pp. 1–4.

[21] U. Meyer-Baese, G. Botella, D. Romero, and M. Kumm, "Optimization of High Speed Pipelining in FPGA-based FIR Filter Design Using Genetic Algorithm," in *Proceedings of SPIE*, 2012.

[22] FIRsuite, "Suite of constant coefficient FIR filters," 2013. [Online]. Available: http://www.firsuite.net

[23] S. Liu, R. N. Pittman, A. Forin, and J. L. Gaudiot, "On Energy Efficiency of Reconfigurable Systems With Run-time Partial Reconfiguration," in *IEEE International Conference on Application-specific Systems Architectures and Processors (ASAP)*, 2010, pp. 265–272.

[24] M. Liu, W. Kuehn, Z. Lu, and A. Jantsch, "Run-time Partial Reconfiguration Speed Investigation and Architectural Design Space Exploration," in *International Conference on Field Programmable Logic and Applications (FPL)*, 2009, pp. 498–502.

[25] M. Kumm, P. Zipf, M. Faust, and C.-H. Chang, "Pipelined Adder Graph Optimization for High Speed Multiple Constant Multiplication," in *IEEE Int. Symposium on Circuits and Systems (ISCAS)*, 2012, pp. 49–52.

[26] U. Meyer-Baese, J. Chen, C. H. Chang, and A. G. Dempster, "A Comparison of Pipelined RAG-n and DA FPGA-based Multiplierless Filters," in *IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, 2006, pp. 1555–1558.

[27] M. Kumm, D. Fanghänel, K. Möller, P. Zipf, and U. Meyer-Baese, "FIR Filter Optimization for Video Processing on FPGAs," *EURASIP Journal on Advances in Signal Processing*, vol. 2013, no. 1, p. 111, 2013. [Online]. Available: http://asp.eurasipjournals.com/content/2013/1/111