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 turns 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].

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...
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.

### 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

\[
c_n = \pm c_{N-n-1}
\]

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

\[
y = \sum_{n=0}^{N/2-1} c_n \left( x_n \pm x_{N-n-1} \right) + c_{N/2} x_{N/2} = z_n
\]

and for odd \( N \), (1) results in

\[
y = \sum_{n=0}^{(N-1)/2-1} c_n \left( x_n \pm x_{N-n-1} \right) + c_{(N-1)/2} x_{(N-1)/2} = z_{n-1}
\]

Due to the reduction of \( N \), the sum terms in (1) are reduced to \( M = \left\lceil \frac{N+1}{2} \right\rceil \) terms in (6) and (7). Hence, the LUT storage requirement for a parallel DA filter is reduced from \( B_f B_f^N 2^N \) to \( (B_f + 1)B_f^N 2^\left\lceil \frac{N}{2} \right\rceil \) (the bit width \( B_f \) 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]

\[
f(\tilde{x}^N_b) = \sum_{l=0}^{[N/L]-1} \sum_{n=L}^{(l+1)L-1} c_n x_{n,b} \quad = f_l(\tilde{x}^N_l)
\]

with \( L < N \) where \( f_l(\tilde{x}^N_l) \) can be realized by partial \( L \)-input LUTs. If \( N \) is not dividable by \( L \), one additional partial LUT
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 \( |N/L| B_x B_f^T 2^L + B_x B_f^T 2^{L'} \) 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, \( B_f^T \) is typically less than \( B_f^N \) as fewer sum terms are involved in \( f(x_k^N) \) of (8). However, this large memory reduction is paid by \( |N/L| \) 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 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

![Optimized parallel realization of a distributed arithmetic FIR filter](image)

![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)](image)
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 $L_{\text{best}}$ (in terms of minimal slice resources)

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

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 $f_{\text{max}}$ 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 $L_{\text{best}}$ leading to a total cover of 119% and 118% for Virtex 4 and Virtex 6, respectively. Thus, there is much optimization potential in Coregen, especially for the latest FPGAs.

In conclusion, the partial LUT size $L$ 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 $L_{\text{best}}$ (in terms of minimal slice resources)

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

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 $f_{\text{max}}$ 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 $L_{\text{best}}$ leading to a total cover of 119% and 118% for Virtex 4 and Virtex 6, respectively. Thus, there is much optimization potential in Coregen, especially for the latest FPGAs.