Low-Latency FIR Filter Structures Targeting FPGA Platforms

PIERO RIVERA BENOIS, PATRICK NOWAK AND UDO ZÖLZER, Dept. of Signal Processing and Communications, Helmut-Schmidt University Hamburg

MARCEL ECKERT AND BERND KLAUER, Dept. of Computer Engineering, Helmut-Schmidt University Hamburg

ACM Reference Format:

Piero Rivera Benois, Patrick Nowak and Udo Zölzer and Marcel Eckert and Bernd Klauer. 2018. Low-Latency FIR Filter Structures Targeting FPGA Platforms. 1, 1 (July 2018), 7 pages. https://doi.org/10.1145/3241793.3241807

1 INTRODUCTION

Finite Impulse Response (FIR) filters are one of the basic building blocks in Digital Signal Processing. The computational complexity of their filtering process is determined by the length of their impulse responses. In high-order systems, the effective group-delay and phase response of the implemented filter may restrictively deviate from the designed one, due to the processing latency that adds on top of the group delay. This deviation may break causality or phase margin constraints within the system, decreasing the performance or correct functioning of it. A good example of systems under such constraints are feedforward and feedback active noise control digital implementations [2, 5].

Varied approaches can be used to decrease the overall complexity, if coefficients are known and have certain patterns [3, 6], a digit-serial architecture is used [4], or if frequency-domain filtering is applied [2].

Nevertheless, reducing the computational complexity (and with it probably increasing the implementation effort) is not the only way towards reducing the processing latency. In the present work, a time-domain sample-by-sample convolution based on precalculations of the output is proposed, which reduces the processing latency to the time required to calculate one multiplication and one addition. This is achieved by changing the chronological order of the calculations, under the constraint that the time to calculate the precalculations also have to fit within a sampling period. The implementation of it can be done based on a one-step or on an iterative precalculation process. To achieve higher order filters with less amount of calculation resources, the multiplications and additions needed for the precalculations are grouped into parallel lanes (in some extent similar to [7]) that sequentially solve partial results using the same calculation units. Although performing the same task, the one-step and iterative precalculations have different memory and logic requirements. Thus, an evaluation and comparison of both are presented in this work.

In the following section, the precalculation strategy is explained. In Section 3, the one-step and iterative precalculations are described together with the grouping and parallelization strategy for reducing the needed computational resources.

Authors' addresses: Piero Rivera Benois, Patrick Nowak and Udo Zölzer, Dept. of Signal Processing and Communications, Helmut-Schmidt University Hamburg, Holstenhofweg85, Hamburg; Marcel Eckert and Bernd Klauer, Dept. of Computer Engineering, Helmut-Schmidt University Hamburg, Holstenhofweg85, Hamburg.

Manuscript submitted to ACM

In Section 4, both strategies are compared in terms of memory and logic requirements. Finally, in Section 5, conclusions are drawn.

2 PRECALCULATION OF CONVOLUTION RESULTS

The discrete-time convolution of a causal FIR filter's impulse response h of order $K-1$ and its input signal $x(n)$ can be written as the well-known sum of products (SOP)

$$y(n) = \sum_{k=0}^{K-1} h_k \cdot x(n-k), \tag{1}$$

where h_k represents the k^{th} coefficient of the impulse response of length K. Depending on K, the computational power of the target platform, and the strategy of calculation, the time needed to calculate the SOP for the present sample $x(n)$, called the processing latency, may increase beyond the sampling period or decrease below it. If it is below, then the input and output samples involved in the convolution do not change during its calculation, and the first term of the SOP may be pulled out

$$y(n) = h_0 \cdot x(n) + \sum_{k=1}^{K-1} h_k \cdot x(n-k) \tag{2}$$

to show that besides $x(n)$, all other input samples are already known before starting with the calculation. If this fact is strategically used to provide the next convolution calculation with a precalculated output

$$\tilde{y}(n+1) = \sum_{k=1}^{K-1} h_k \cdot x(n+1-k), \tag{3}$$

then the processing latency is minimized to the time it takes to perform one multiplication and one addition

$$y(n+1) = h_0 \cdot x(n+1) + \tilde{y}(n+1), \tag{4}$$

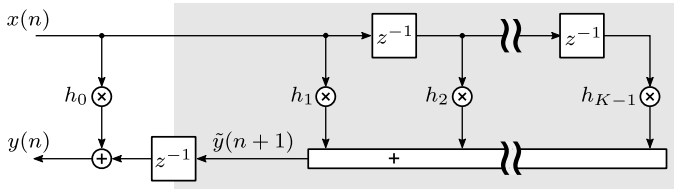
without having to allocate additional computational resources.

This progress in decreasing the processing latency may still not be fully exploited, if ADC and DAC units are clocked with the same signal, i.e. coupled together. In this case, the processing latency will have an effective value that is fixed to the sampling period. To get rid of this undesirable effect, ADC and DAC units should be decoupled and the digital-to-analog conversion triggered directly after $y(n)$ is calculated.

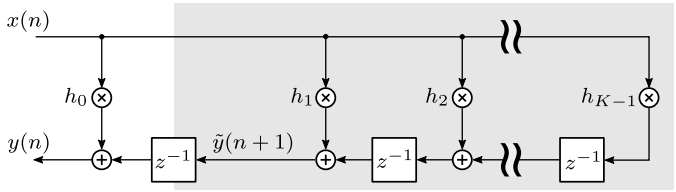
3 IMPLEMENTATION ALTERNATIVES

If the precalculation strategy presented in the previous section is done using a one-step approach (OSA), then the structure of the calculations matches the special case of an FIR Hybrid Form I [1] (see Fig. 1a), but if instead the precalculations are done using an iterative approach (ITA), then the behaviour matches the FIR Transposed Direct Form I [8] (see Fig. 1b).

These two alternatives for the calculation of $\tilde{y}(n+1)$ have the same computational complexity, but different memory requirements. On the one hand, the Hybrid Form I has to store $K-1$ samples of $x(n)$ and eventually, but not mandatory, the partial results in the adder-tree for pipelining. On the other hand, the Transposed Form I has to store all additions of individual multiplication results with the signal that comes from the stage before. For fixed-point implementations this means an advantage of the Hybrid Form over the Transposed one, because of the wider bit-width that the results have



(a) Special case of a Hybrid Form I



(b) Transposed Direct Form I

Fig. 1. Two alternative structures for precalculations in an FIR filter: (a) A special case of the Hybrid Form I and (b) the Transposed Direct Form I.

over the input samples. This advantage may be bigger, if the bit-width of the coefficients is increased, the adder-tree is small and no pipelining is required.

If all multiplications are done in parallel, as implied in Fig. 1a and Fig. 1b, the amount of computational resources used for the K multiplications may go beyond the ones available. If instead the calculations are sequentially performed by calculation units running in parallel, longer impulse responses can be reached with less resources. This idea is applied to the one-step precalculation to reconfigure the SOP in (3) into

$$\tilde{y}(n+1) = \sum_{q=0}^{Q-1} \sum_{k=1}^{L} h_{qL+k} \cdot x(n+1-qL-k), \tag{5}$$

where Q is the number of Multiply-Accumulate-Lanes (MAC-Lanes) that simultaneously calculate sums of length L

$$\tilde{y}_q(n+1) = \sum_{k=1}^{L} h_{qL+k} \cdot x(n+1-qL-k), \tag{6}$$

and a pipelined adder-tree of depth $\lceil \log_2(Q) \rceil$ sums up their outputs to yield the final result

$$\tilde{y}(n+1) = \sum_{q=0}^{Q-1} \tilde{y}_q(n+1). \tag{7}$$

All this with Q and L respecting the relationship $Q \cdot L = (K-1)$ and being powers of 2, for maintaining a balanced adder-tree.

In the case of the iterative calculation, there is no sequential accumulation of products, but instead a sequential update of registers. If the grouping is done under the same parameters Q and L, and similar indices k and q are used, an update at the position $qL+k$ can be expressed as

$$\Delta_{qL+k}(n) = h_{qL+k} \cdot x(n) + \Delta_{qL+k+1}(n-1), \tag{8}$$

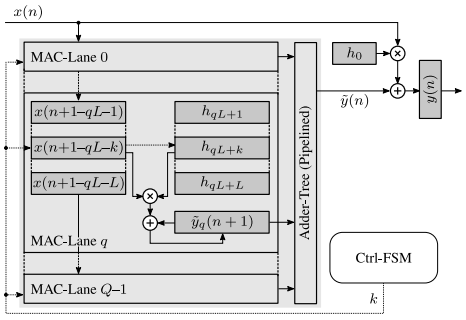
where the current register $\Delta_{qL+k}(n)$ is updated with the addition of the multiplication $h_{qL+k} \cdot x(n)$ and the last stored value in the preceding register $\Delta_{qL+k+1}(n-1)$. The sequence of updates produces that at any position the stored value yields

$$\Delta_{qL+k}(n) = \sum_{i=qL+k}^{QL} h_i \cdot x(n+qL+k-i), \tag{9}$$

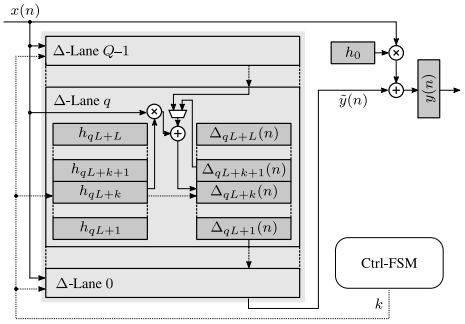
including the update at position 1

$$\tilde{y}(n+1) = \Delta_1(n) = \sum_{i=1}^{QL} h_i \cdot x(n+1-i). \tag{10}$$

The structograms for both calculation strategies are presented in Fig. 2. As previously described, the direct path of $x(n)$ through the multiplication with h_0 and addition with $\tilde{y}(n)$ is common for both structures and independent of the precalculation's parametrization. The calculation of $\tilde{y}(n)$ takes place within the light grey block under the control of the Finite State Machine (Ctrl-FSM), which not only synchronizes the parallel calculation in the MAC- and Δ-Lanes through the address k, but also the overall calculation cycle. Each dark grey block symbolizes a block of memory (RAM or ROM), where either the input samples $x(n+1-qL-k)$, the filter coefficients h_{qL+k}, the partial sums of each MAC-Lane $\tilde{y}_q(n+1)$, or the partial results $\Delta_{qL+k}(n)$ are stored.



(a) One-step approach: Special case of a Hybrid Form I using Q concurrent MAC-Lanes and a pipelined adder-tree of depth $\lceil \log_2(Q) \rceil$.



(b) Iterative approach: Transposed Direct Form I using Q concurrent Δ-Lanes.

Fig. 2. Two alternative FPGA implementation structures for the precalculation strategy: (a) One-step approach and (b) iterative approach.

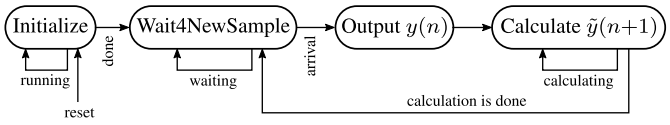


Fig. 3. Finite state machine used to coordinate the concurrent calculations of both FPGA implementations in Fig. 2.

The synchronization performed by the Ctrl-FSM follows the cycle depicted in Fig. 3. There, on reset, the state-machine goes into the *Initialize* state, where it waits until the ADC unit is ready with its initialization routine. After the ADC

signals the end of it, the state-machine transits to the *Wait4NewSample* state, where it is waiting for the ADC to deliver a new sample. Upon arrival, the sample is used for the calculation of $y(n)$ and stored for its future usage in the calculation of $\tilde{y}(n+1)$. In the used platform, the data is one clock cycle ahead of the control signal used to indicate its arrival. This clock cycle is enough time for the multiplication $x(n) \cdot h_0$ and its addition with $\tilde{y}(n)$, so that in state *Output y(n)* only the result has to be stored and the convert signal has to be sent to the DAC. After this is done, the state-machine transits to *Calculate* $\tilde{y}(n+1)$ and remains there until all L calculations are done in all Q lanes. When it is ready, a final transition to the *Wait4NewSample* state is done, where the filtering process starts all over again.

4 RESULTS AND EVALUATION

For comparing and evaluating the two implementation alternatives, the Kintex-7 xc7k325t-fbg900 from Xilinx clocked with a fixed processing rate of 100 MHz was used. A sampling frequency of 48 008 Hz was selected, which limits the clock cycles per sampling period to 2083. At the same time, this limitation provides a maximum depth of $L_{max} = 2048$ per MAC- or Δ-Lane. The ADC in the platform supplies samples with a bit-width of 16 bit, while 32 bit was chosen for the coefficients of the impulse response. The used FPGA offers 840 DSP slices, 445 random-access memory blocks (RAMBs), 203 800 Look-up tables (LUTs) and 407 600 flip-flops (FFs). Two DSP slices are needed to calculate one Multiply operation either in the direct path or in one of the parallel lanes for the precalculation. Therefore, the amount of required DSPs is related to the number of parallel MAC- or Δ-Lanes as $2Q + 2$, resulting in a maximum parallelization level of $Q_{max} = 256$ for the chosen FPGA. To determine the memory and logic resource requirements, the amount of RAMBs needed for storing purposes and the number of LUTs and FFs needed for the logic are measured under different length and parallelization setups.

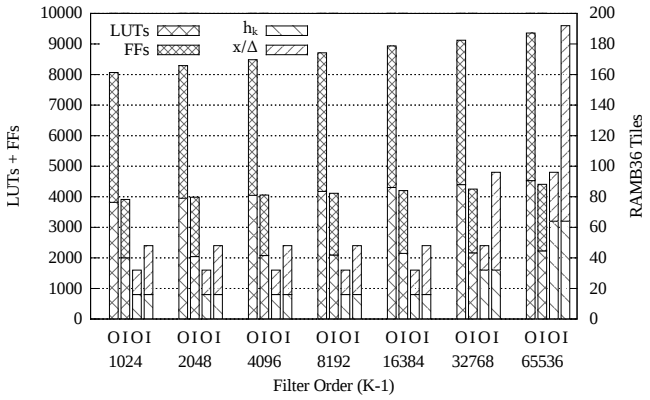


Fig. 4. Resource utilization for different filter orders with 32 parallel lanes. Repetitive x-axis index: O for OSA and I for ITA.

In Fig. 4, the resource utilization for different filter orders and a fixed parallelization level $Q = 32$ is shown. Four columns for each length are given, from which the first pair is relative to the left axis (logic requirements) and the second pair relative to the right axis (memory requirements). Within a pair, the left column belongs to the OSA and the right column to the ITA. It can be seen, that the OSA uses almost twice the LUTs and FFs than the ITA. This difference is mostly explained by the pipelined adder-tree needed in the OSA. In contrast to this, the ITA requires more RAMBs for its Δ-lanes, due to the difference in bit-width between the 16 bit of the samples stored by the OSA and its partial

results $\Delta_{qL+k}(n)$ stored using $16 + 32 + \lceil \log_2(K-1) \rceil$ bit. As expected, both structures use the same number of RAMBs for storing the coefficients h_{qL+k}. Additionally to this difference, it can be seen, that the amount of RAMBs needed in both structures remains constant until a filter order of 16 384 is reached. This fact is motivated by the ability of the RAMB36 tile to adjust his properties to the situation. Generally, a RAMB36 has a depth of 1024 and a width of 36 bit. If a different bit-width is required, it can be reconfigured to use different depths maintaining the same area (e.g. $512 \cdot 72$ bit). It has to be taken into account that the minimum depth is restricted to 512. For a parallelization of 32, this depth is reached by a filter of order 16 384, wherefore all shorter filters need the same number of RAMB36 tiles and waste some area of them. In addition to the reconfigurability, the RAMB36 tile can be split into two RAMB18 tiles, allowing two parallel lanes to use a common RAMB36 tile (e.g. $2 \cdot 512 \cdot 36$ bit). That is why the coefficients and the sample buffers require only half of the number of RAMBs compared to the partial results up to a filter order of 16 384. For a filter order of 32 768, a depth of 1024 is needed, therefore the amount of RAMBs used for coefficients and partial results $\Delta_{qL+k}(n)$ doubles, while the number of RAMBs for the sample buffers remains unchanged. Furthermore, the number of RAMBs needed in the implementation will duplicate for every doubling in the filter order, resulting in a constant factor of 4 between the amount of RAMBs used for x/Δ in the OSA and the ITA, respectively. In contrast to that progress of the RAMB36 tiles, the number of needed LUTs and FFs rises slowly with increasing filter order.

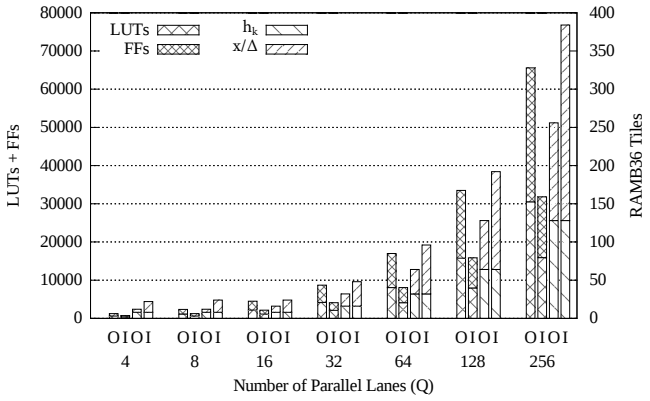


Fig. 5. Resource utilization for different number of parallel lanes with filter order 8192. Repetitive x-axis index: O for OSA and I for ITA.

The influence of different parallelization levels for a constant filter order is shown in Fig. 5. The filter order is fixed to 8192, while the parallelization level starts at $Q = 4$ and is incremented up to the maximum possible value of $Q_{max} = 256$. It is visible that the resource utilization linearly increases with the level of parallelization. In case of the RAMB36 tiles, the required number exactly doubles for a duplication of the parallelization level. This fact is due to the capabilities of the RAMB36 tile described before. The difference in resource requirements between the OSA and the ITA is similar to the one seen in Fig. 4. Where the former uses more LUTs and FFs and less RAMB36 tiles than the latter.

5 CONCLUSIONS

In the present work, two different low-latency FIR filter structures based on sample-by-sample one-step and iterative precalculations are proposed and evaluated in terms of resource requirements. As seen in the results of previous section, the one-step precalculation implementation utilizes between 33% and 50% less memory blocks than the iterative

precalculation scheme. Nevertheless, the logic behind it requires around twice the amount of LUTs and FFs, showing the validity of both approaches by their mutual contrasting resource requirements.

The parallelization of sequential calculations for the reutilization of DSP slices showed to be very effective in the studied scenario, by extending the maximum impulse response length by a factor of 2048. Nevertheless, the logic and memory requirements increase linearly with the number of parallel lanes, so if the available memory blocks, LUTs, or FFs are the limiting resources, a minimum number of parallel calculation lanes should be used.

The resource-efficient implementation of the one-step precalculation enables the chosen platform to reach a filter order of 262 144 (≈ 5.5 s at $f_s = 48$ kHz) using only 258 DSP slices, whereas in the case of the iterative precalculation only a half of the order can be reached, due to its higher memory requirements. Both results having a digital processing latency of only 20 ns (2 clock cycles at $f_{clk} = 100$ MHz).

REFERENCES

[1] K. Azadet and C. J. Nicole. 1998. Low-power equalizer architectures for high-speed modems. *IEEE Communications Magazine* 36, 10 (Oct 1998), 118–126. DOI: http://dx.doi.org/10.1109/35.722147

[2] Marco Fink, Sebastian Kraft, Martin Holters, and Udo Zölzer. 2013. Load-Balanced Implementation of a Delayless Partitioned Block Frequency-Domain Adaptive Filter. In *2. Workshop Audiosignal- und Sprachverarbeitung, 43. Jahrestagung der Gesellschaft für Informatik*.

[3] O. Gustafsson, J. O. Coleman, A. G. Dempster, and M. D. Macleod. 2004. Low-complexity hybrid form FIR filters using matrix multiple constant multiplication. In *Conference Record of the Thirty-Eighth Asilomar Conference on Signals, Systems and Computers, 2004.*, Vol. 1. 77–80 Vol.1. DOI: http://dx.doi.org/10.1109/ACSSC.2004.1399091

[4] Hwan-Rei Lee, Chein-Wei Jen, and Chi-Min Liu. 1996. A new hardware-efficient architecture for programmable FIR filters. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing* 43, 9 (Sep 1996), 637–644. DOI: http://dx.doi.org/10.1109/82.536760

[5] S. Liebich, C. Anemüller, P. Vary, P. Jax, D. Rüschen, and S. Leonhardt. 2016. Active Noise Cancellation in Headphones by Digital Robust Feedback Control. In *Proceedings of the 24th European Signal Processing Conference (EUSIPCO)*.

[6] Shyh-Feng Lin, Sheng-Chieh Huang, Feng-Sung Yang, Chung-Wei Ku, and Liang-Gee Chen. 2004. Power-efficient FIR filter architecture design for wireless embedded system. *IEEE Transactions on Circuits and Systems II: Express Briefs* 51, 1 (Jan 2004), 21–25. DOI: http://dx.doi.org/10.1109/TCSII.2003.821513

[7] P. K. Meher. 2007. Low-latency hardware-efficient memory-based design for large-order FIR digital filters. In *2007 6th International Conference on Information, Communications Signal Processing*. 1–4. DOI: http://dx.doi.org/10.1109/ICICS.2007.4449798

[8] Uwe Meyer-Baese. 2014. *Digital Signal Processing with Field Programmable Gate Arrays* (4th ed.). Springer Publishing Company, Incorporated.