

Hardware Acceleration of Digital Filters Using The STM32 FMAC Peripheral - a Performance Comparison

Marc Hochuli and Stefan Gorenflo

Abstract—Filters are a fundamental component of signal processing, and an increasing number of them are implemented digitally. While digital implementations offer increased flexibility they simultaneously set higher demands on microcontrollers. As a result selected generations of recent STM32 microcontrollers ship with a Filter Math Accelerator (FMAC) peripheral which allows to offload the filtering from the CPU. This letter provides a performance and accuracy comparison of the FMAC peripheral in single section and second order sections configuration to the widely adopted CMSIS-DSP software implementation. The results show that depending on configuration, filter order and data type the FMAC reduces CPU usage up to 99% - at the cost of accuracy - thus making it ideal for applications where concurrent tasks have to be executed.

Index Terms—Signalprocessing, hardware acceleration, Filtering, Filter Math Accelerator (FMAC), STM32.

I. INTRODUCTION

SELECTED generations of recent STMicroelectronics microcontrollers feature a Filter Math Accelerator (FMAC) peripheral. It allows offloading of filter tasks to hardware, freeing up vital CPU resources. This peripheral has already been used for various cases, such as type-II compensators in switched mode power supplies [1] or PLLs in three-phase converters [2]. Yet there is still no general evaluation regarding its behavior compared to the commonly used CMSIS-DSP implementation. In this letter we aim to fill this gap by implementing the FMAC peripheral on an STM32H563 MCU [3] and evaluate performance and accuracy compared to CMSIS-DSP version 1.17.0 [4]. While CMSIS always implements second order sections (SOS) [5, Ch. 6.8] the FMAC commonly implements single section (SS) [5, Ch. 6.1] filters. For this letter both configurations for the FMAC peripheral are examined to evaluate the differences.

II. BASICS OF FILTERING

In order to understand the difference between SS and SOS, the basic principles of digital filtering must be established first.

Equation 1 denotes the $\mathcal{Z}$ -transform of the transfer function of a digital filter. From this form the poles and zeros can easily be calculated and plotted in the $\mathcal{Z}$ -plane. [6, p. 251]

$$H(z) = \frac{\sum_{k=0}^N b_k z^{-k}}{1 + \sum_{k=1}^M a_k z^{-k}} \quad (1)$$

Figure 1 shows the locations of the poles and zeros in the $\mathcal{Z}$ -plane with different filter orders and cutoff frequencies. It can be seen that with higher filter order more poles need to fit into the same space, thus they move closer together. Furthermore, the greater the ratio between sampling frequency and cutoff frequency ($\delta = \frac{f_s}{f_c}$) the closer the poles move

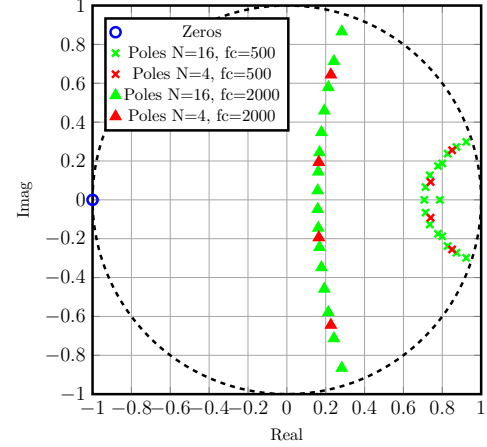


Fig. 1. Behavior of poles and zeros of a Bessel Lowpass with different orders (N) and cutoff frequencies (f_c). Sampling frequency is 10 kHz

towards $z = 1$, compressing the available space even more. Both of those effects result in increasingly tighter packed coefficients, leading to issues regarding numerical stability, becoming even more pronounced when quantizing the coefficients. The sensitivity of the poles p_i to quantization is described in (2) [7, p. 629].

$$\Delta p_i = - \sum_{k=1}^N \frac{p_i^{N-k}}{\prod_{l=1, l \neq i}^N (p_i - p_l)} \Delta a_k \quad (2)$$

It can be seen that each pole is influenced by all coefficients and its sensitivity increases with decreasing distance ($p_i - p_l$) between poles. So to reduce sensitivity, a small filter order and large pole-to-pole distance is desired. In order to reach this goal, a filter can be realized as a cascade of second order sections consisting of two complex-conjugated poles per section. Combining the poles appropriately leads to an increased distance of pole-pairs, while simultaneously reducing the pole count [5, Ch. 6.8]. Figure 2 shows the filter coefficients of a Single Stage filter compared to the complex conjugated poles from the Second Order Sections filter. It can be seen that the poles are now grouped into pairs which have a larger distance. The exception are the poles which are closest to the abscissa, but even in this case the sensitivity is reduced because the number of poles N is significantly reduced. In practice this reduction in sensitivity when using SOS will result in a better accuracy of the filter output.

III. FMAC ARCHITECTURE

This chapter provides an overview over the core principles and limitations of the FMAC peripheral, allowing to contex-

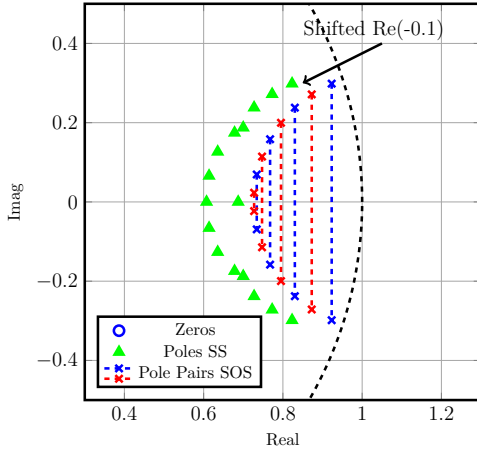


Fig. 2. Second Order Sections (SOS) vs. Single Section (SS) filter pole placement. For better readability SS poles are shifted by -0.1 on the x axis

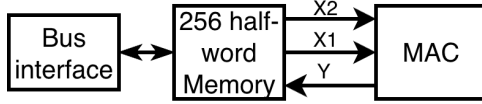


Fig. 3. FMAC-Peripheral overview

tualize the results better.

Figure 3 shows a brief overview over the peripheral. It communicates over the high-performance bus through which configuration and data can be written into its internal buffer of 256 half-words. The memory granularity reflects the peripheral's 16 bit architecture using the q1.15 signed fixed-point format, so all data including filter coefficients must be scaled to the range $[-1, 1 - 2^{-15}]$. The architecture of the multiply-accumulate (MAC) unit is shown in Figure 4, which is a Direct Form I structure with a programmable gain of 2^R . This gain can be used if the filter coefficients need to be scaled to the aforementioned range. R can be calculated using (3) and subsequently all the coefficients can be scaled as shown in (4).

$$R = \lceil \log_2(\max(\{a_i\} \cup \{b_j\})) \rceil \quad (3)$$

$$c_{i,s} = \frac{c_i}{2^R} \quad (4)$$

All the values used for the MAC unit are stored in its internal memory, whose layout can be seen in Figure 5. The $X2$ Vector contains all the filter coefficients a_k and b_k , leading to a length of $N + 1 + M$ elements. Vector $X1$ stores the input data x of the filter whose length is equal to the number of feedforward coefficients $N + 1$. The output data y is stored in vector Y with length M , which results in a total memory consumption of $2(N + 1) + 2M$. An IIR filter of order N usually has $N + 1$ b-coefficients and $M = N$ a-coefficients so the highest implementable filter order is $N = 63$. An FIR filter has no a-coefficients, so filter orders up to $N = 126$ can be realized [8].

As before-mentioned the implementation of filters with high oversampling or high order is often performed using cascaded SOS. Since there is only one FMAC peripheral it is not possible to directly cascade multiple of those to implement this structure. One possible solution is to simply

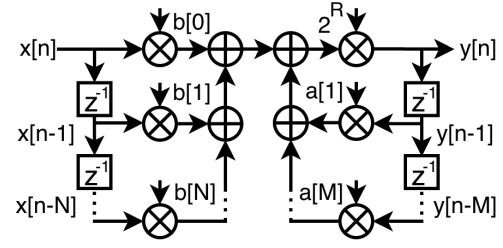


Fig. 4. Hardware implementation of Direct Form I

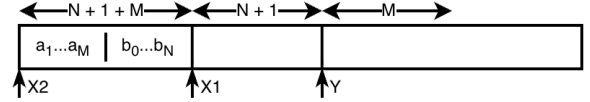


Fig. 5. Memory setup of FMAC peripheral for single stage filters

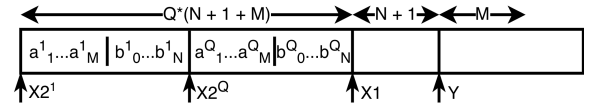


Fig. 6. Memory setup of FMAC peripheral for cascaded filters - with multiple filter coefficient packets (Q) preloaded

reconfigure the filter between different stages, however this would be detrimental to performance since a single chunk of data would need several reconfigurations. The better approach is to preload the filter coefficients for all stages into the FMAC memory and switch the $X2$ pointer between the cascades, eliminating the reconfigurations at the cost of space in the internal FMAC memory. The memory layout of this can be seen in Figure 6, where Q denotes the number of cascades. The total memory consumption amounts to $Q \cdot (N + 1 + M)$. Since for a second order section the coefficient counts are $N + 1 = 3$ and $M = 2$, the size of the vectors $X1$ and Y shrink allowing IIR filters with orders of up to 102 with minimal overhead for switching stages.

From a practical point of view one must consider that most filter design tools (i.e. Matlab Filter Designer and SciPy) normalize all the gain in the first stage of the Q second order sections. Consequently, for higher-order filter this approach results in numerically small b-coefficients in the first stage, which after conversion to q1.15 are detrimental to precision. In order to overcome this issue the gain should be redistributed among all the stages. The first section must have the total gain g removed (5) and subsequently all the stages need their individual gain applied (6).

$$b_i = \frac{b_{i,g}}{g}, \quad i = 1, \dots, 3 \quad (5)$$

$$b_{i,s} = b_i \cdot \sqrt[Q]{g}, \quad i = 1, \dots, 3Q \quad (6)$$

While the redistributed gain solves the quantization issue of the filter coefficients it introduces the possibility of internal overflow of the FMAC peripheral. This can easily be solved by dividing the input data before feeding the filter and compensating this by multiplying the same amount after filtering. For performance reasons this is ideally realized using left-

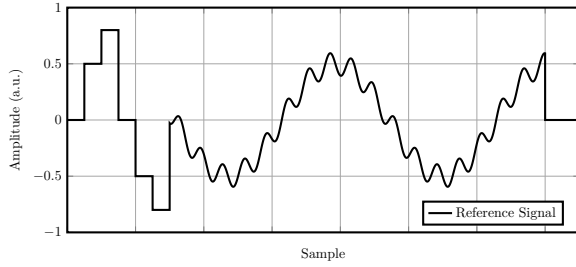


Fig. 7. Reference signal, used as an input for different filter implementations

and rightshifts. However, this comes at the cost of reduced resolution and a bias (truncation, Sec. IV).

IV. RESULTS

The FMAC peripheral was evaluated in relation to calculational accuracy and computational performance compared to the filter implementation of the CMSIS-DSP, an ARM-maintained library employing various optimizations such as SIMD to maximize performance, thus representing a near-ideal case of software implementation. For comparison of accuracy, a synthetic reference signal shown in Figure 7 was created and filtered using SciPy (v1.15.2) which calculates with 64-bit precision. It consists of multiple steps and two superimposed sine waves, allowing the evaluation of step response, saturation effects and dynamic behavior. To amplify the effect of the computational effort a large reference signal was used with a block size of 15×10^3 samples. We used a Bessel lowpass filter having a sampling frequency of 10 kHz and a cut-off frequency of 3 kHz for orders ranging from 2 to 80. The comparison is limited to the CMSIS-DSP floating point implementation, since the fixed point implementation will not outperform the hardware accelerated peripheral, while experiencing the same inaccuracies. The CPU effort was measured in cycles, using the CYCCNT register provided by the DWT (Data Watchpoint and Trace) unit. ICACHE was enabled and wherever possible data transfers were done using DMA. The compiler optimization level was set to -O0.

Figure 8 shows the needed CPU cycles for each filter implementation depending on the filter order. Since the FMAC calculates the filter in the background, a distinction has to be made between the actual computational effort (EFF), and the time it takes the FMAC to finish the calculation (TTF). The EFF determines how much CPU time is available for other tasks, whereas the TTF determines how long it takes until the result is ready. The EFF for all implementations with the FMAC is almost independent of the filter order since the only difference for the CPU is in the number of coefficients. The TTF scales approximately linearly for larger filter orders. It can be seen that for very low orders the FMAC unit does not have any advantage compared to the CMSIS implementation.

Table I shows the computational effort broken into different tasks. "Convert" includes the conversion of floating point to fixed point data which can be omitted if the data is available in an integer format as displayed by the *-I suffix. The "Scale" task handles the overflow preventing predivision, which is only

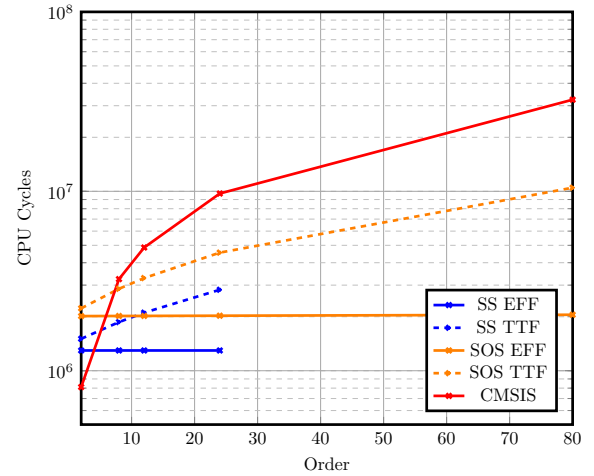


Fig. 8. Computational Effort vs. Filter Order with different implementations and floating point data

TABLE I
COMPUTATIONAL EFFORT OF DIFFERENT FILTER IMPLEMENTATIONS AND ORDERS IN CPU CYCLES
*-I = INT

Type	Convert	Scale	Append	Total EFF	TTF
Order 2					
SS	1.296 M	-	465	1.297 M	1.507 M
SS-I	-	-	454	454	211 k
SOS	1.296 M	720 k	459	2.016 M	2.228 M
SOS-I	-	720 k	459	721 k	932 k
CMSIS	-	-	-	811 k	811 k
Order 24					
SS	1.296 M	-	460	1.297 M	2.827 M
SS-I	-	-	454	454	1.531 M
SOS	1.296 M	720 k	10.1 k	2.026 M	4.553 M
SOS-I	-	720 k	10.1 k	730 k	3.257 M
CMSIS	-	-	-	9.723 M	9.723 M
Order 80					
SOS	1.296 M	720 k	37.8 k	2.051 M	10.473 M
SOS-I	-	720 k	37.8 k	758 k	9.177 M
CMSIS	-	-	-	32.409 M	32.409 M

applicable during the SOS implementation. "Append" is the effort it takes to hand over a new data block to the peripheral. In case of SOS implementation, this includes switching the coefficient pointers and the input and output buffers.

The table shows that the drawback of the FMAC at lower orders is caused by the effort needed to convert the floating point data to the fixed point format. This can be circumvented, by directly supplying the data in integer format, allowing the FMAC to show an immediate advantage. This works well for practical applications since the data is often taken from peripherals such as an ADC, which already delivers integer data. For higher order filters the FMAC peripheral provides improvements in EFF of multiple magnitudes, especially if the SS implementation can be realized, which reduces EFF by 99% (all orders) to a few hundred cycles. However, even when providing floating point data the FMAC in SS configuration reduces EFF by up to 87% (Ord. 24). From a pure performance point of view choosing SOS over SS is never worth it since the SOS implementation takes additional time to append and scale the data for overflow prevention. Still the SOS provides

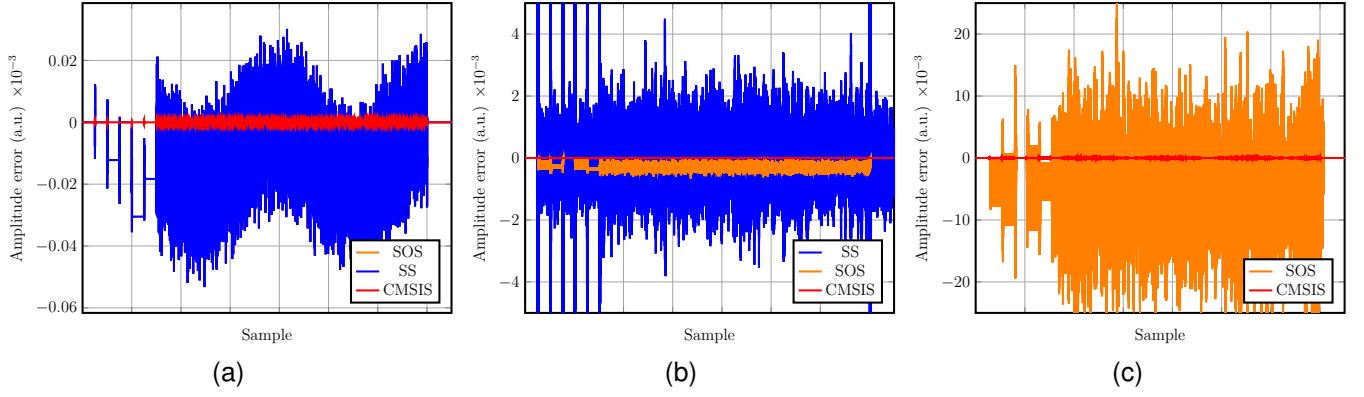


Fig. 9. Deviation of the output signals of filters with different orders from the reference signal. (9a) Ord. 2 (SS and SOS overlap), (9b) Ord. 24 and (9c) Ord. 80

EFF reductions of 80% (Ord. 24) or even up to 94% (Ord. 80) over CMSIS. The advantage over SS only becomes apparent in terms of accuracy, numerical stability and the ability to implement higher order filters (i.e. 80), as mentioned in the FMAC overview (memory consumption, Sec. III). Figure 9 shows the amplitude deviation of the different filter implementations at selected orders relative to the reference signal. It is evident, that the CMSIS implementation is the most accurate, due to its use of FP32. However, for order 2 (Fig. 9a), both the SS and the SOS implementations only deviate a few LSB (i.e. 0.03×10^{-3}) of the fixed point 1.15 format. For higher order filters (Fig. 9b) the SOS implementation exhibits improved accuracy ($< 0.05\%$) deviation compared to the SS ($< 0.4\%$) implementation, as predicted by the theory (reduced pole sensitivity, Sec. II). If the filter order increases substantially (Fig. 9c), even the SOS implementation has accuracy issues. This effect is not solely caused by coefficient quantization, but also results from the predivision (internal overflow, Sec. III) by right- and leftshifts required to avoid overflow. This effect could be mitigated by dividing in between each stage instead of only once at the start of the calculation, at the drawback of increased computational effort.

The error plots also show a consistent negative bias of the amplitude error, which increases for the SOS implementations with higher orders. This effect results from the shifts used for overflow prevention during the calculation. Specifically, arithmetic rightshifts perform truncation which always rounds down. While the expected value of a uniformly distributed random variable is $E[r] = 0$, after shifting one bit right then left results in $E[r_s] = -0.5$. For any shift i the expected value can be obtained from (7).

$$\epsilon = -\frac{2^i - 1}{2} \quad (7)$$

This could be improved by implementing more advanced truncation techniques, at the cost of more computational effort.

V. CONCLUSION

The investigations around the FMAC peripheral revealed different strengths depending on application constraints.

If precision is the primary concern the CMSIS implementation remains superior over the FMAC peripheral since the limitations by the data type introduce numerical inaccuracies.

However, if the application allows for some amount of deviation ($< 0.1\%$) and high data processing rates are required, the FMAC peripheral provides a way to greatly reduce the computational effort of filtering. If the order is low enough for using the single stage implementation and the data is available in a format which can directly be fed into the FMAC, the computational effort reduces by 99% (all orders) to only a few hundred CPU cycles per block to append new data. Even if data conversion is required and Second Order Stages need to be realized the peripheral provides up to 80% (Ord. 24) reduction, or in extreme cases up to 94% (Ord. 80).

These reductions in computational effort make the FMAC peripheral ideal for cases where concurrent workloads need to be executed, such as communication or operation of a control system, if high accuracy is not critical.

REFERENCES

- [1] W. Lin, G. Chu, M. Li, and Q. Bu, "Implementing a High-Frequency Digital SMPS with Microcontroller Hardware Accelerator," in *2024 13th International Conference on Renewable Energy Research and Applications (ICRERA)*, 2024, pp. 304–308.
- [2] A. A. Hussein, A. H. Ahmed, and N. M. Mohammed, "An enhanced implementation of SRF and DDSRF-PLL for three-phase converters in weak grid," *International Journal of Emerging Electric Power Systems*, vol. 24, no. 3, pp. 273–286, Jun. 2023. [Online]. Available: <https://www.degruyterbrill.com/document/doi/10.1515/ijeeps-2021-0367/html>
- [3] STMicroelectronics, "Reference Manual 0481," Apr. 2025.
- [4] "CMSIS-DSP: Overview." [Online]. Available: <https://arm-software.github.io/CMSIS-DSP/latest/index.html>
- [5] R. G. Lyons, *Understanding digital signal processing*, 3rd ed. Upper Saddle River, NJ: Pearson Education, 2010.
- [6] S. Palani, *Principles of Digital Signal Processing: 2nd Edition*. Cham: Springer International Publishing, 2022. [Online]. Available: <https://link.springer.com/10.1007/978-3-030-96322-4>
- [7] J. G. Proakis and D. K. Manolakis, *Digital signal processing*, fourth edition, pearson new international edition. ed. Harlow, England: Pearson, 2014, publication Title: Digital signal processing.
- [8] STMicroelectronics, "an5305-digital-filter-implementation-with-the-fmac-using-stm32cubeg4-mcu-package-stmicroelectronics.pdf." [Online]. Available: https://www.st.com/resource/en/application_note/an5305-digital-filter-implementation-with-the-fmac-using-stm32cubeg4-mcu-package-stmicroelectronics.pdf