High Performance IIR Filter FPGA Implementation Utilizing SOS Microcode Core

Ing. Ondřej Zoubek, Ph.D.[1)] and Ing. Tomáš Musil, Ph.D.[2)]
[1)] FEE CTU in Prague, Prague, Czech Republic, e-mail: *zoubeon1@fel.cvut.cz*
[2)] FTS CTU in Prague, Prague, Czech Republic, e-mail: *musil@fd.cvut.cz*

Abstract — **This paper discusses the methods of optimal IIR filter FPGA implementation. The methods are focused on the reduction of occupied resources and increasing data throughput. Higher demands on an internal controller complexity are successfully solved by utilizing programmable microcode controller. The novelty of SOS core and its capabilities are presented and different variants of SOS core are assessed. The workflow of IIR filter design using MATLAB considering rounded coefficient method is demonstrated.**

Keywords — digital filters, FPGA, high throughput, IIR filter, microcode, MSOSps, Second Order Section.

I. Introduction

The Infinite Impulse Response (IIR) digital filter design is well known for decades and a great variety of Field Programmable Gate Array (FPGA) and CMOS process implementations have been published. Nevertheless, the performance of published IIR filter [1], [2] implementations in proportion to occupied FPGA fabric resources do not reach optimal levels. This is particularly important for mid-range FPGAs where a non-optimal design of a simple IIR filter can easily deplete FPGA resources. One of the ways how to reduce the logic complexity and resources demands to implement an IIR filter is to fit arithmetical resources optimally to requirements of the given signal. Time sharing of the FPGA most valuable resources is the other way how to reach higher performance. Dividing of complex IIR filters into 2nd Order Sections (SOS) helps in both cases [3], [4].

II. SOS Computation Complexity

Referring to Fig. 1 it can be seen that for implementing of 2nd order IIR filter it is necessary to multiply with four different constants (b_1, b_2, a_1, a_2) and add five values ($x(n)$, $x(n-1)\cdot b_1$, $x(n-2)\cdot b_2$, $y(n-1)\cdot a_1$, $y(n-2)\cdot a_2$) and accumulate them.

The values used for computation are filter input values $x(n)$, $x(n-1)$, $x(n-2)$ and filter output values $y(n-1)$ and $y(n-2)$. The output value $y(n)$ have to be saved for next computation.

Assuming that the filter input value $x(n)$ must be also stored somewhere, the whole 2nd order IIR digital filter requires six memory elements (word bit width according to requested arithmetical accuracy).

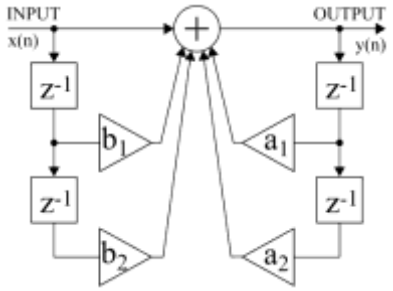

Fig. 1. General 2nd order IIR digital filter.

Mentioned general 2nd order IIR digital filter can be considered as a basic building SOS block of all other more complex IIR filters.

According to conventional design methods and fabric resources of mid-range FPGA (Spartan-6 family by Xilinx Inc. can be considered as an example of mid-range FPGA) one SOS consuming 4 multipliers and 4 adders in case of using recursion. That means 144 D-type flip-flops in rather common 24 bit arithmetic, not mentioning pipeline registers at this place. The computing of one value would take one system clock, specifically on Spartan-6, computing frequency can exceed 100 MHz, although this design speed is not necessary for many applications. On the other side, designs with more 2nd Order Sections for implementation of higher order IIR filters or more individual digital filters imply structure replication and spending multiple resources.

In applications where filter sample frequency versus design frequency ratio allows it the adders and the multipliers can be time multiplexed (see Fig. 2) and the memory places can be shared between 2nd Order Sections (see Fig. 3). For the purpose of researching, the amount of resources used for computation of one 2nd Order Section MSOSps (Mega SOS per second) was defined. One multiplier and one adder in mid-range FPGA possibly carry out 12.5 to 20 MSOSps at 100MHz frequency according to pipeline length and controller complexity. For higher MSOSps performance utilizing of more adders and multipliers is necessary.

TELEN2019008
DOI 10.14311/TEE.2019.2.022

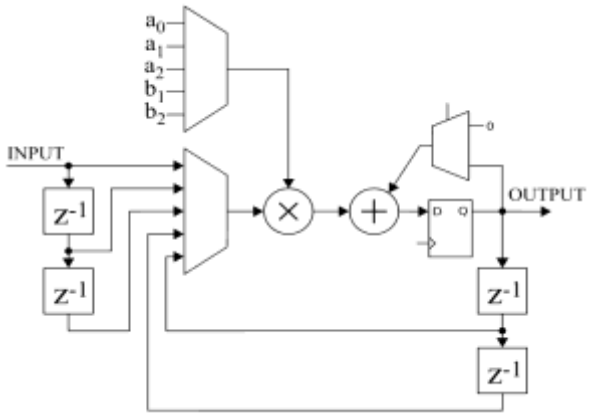

Fig. 2: Time multiplexed computation of SOS.

If the computation is processed in time multiplex with the adder (the accumulator) the length of accumulation determines the ratio between the amount of data stored and data fetched periodically in each clock cycle; the amount of data stored is constant and the amount of data fetched necessarily is even with adder and multiplier data flow i.e. one data word regardless of its particular bit length. The IIR filter consists of series of 2nd Order Sections. The internal 2nd Order Sections (not the first, not the last in the series) need to store five times less amount of data than read – one storage per five fetches and accumulate cycles. The first SOS input data and the last SOS output data have to be stored somewhere too, but we neglect it for now. Instead of using discrete D-type flip-flop in FPGA fabric it is more efficient to use addressable memory because the amount of data stored and fetched in each clock cycle is small (up to one word). Considering Spartan-6 FPGA there are two possibilities of addressable memory. The first is distributed memory (special mode of Configurable Logic Blocks configuration memory) with 64x1 bit granularity which can be used for filters with sample rates from ones to tens Msps or equivalently ones MSOSps at design frequency 100 MHz. The other is block RAM (BRAM) with the capacity of 18 kbit each and configurable organization of port width from 1 to 36 bits.

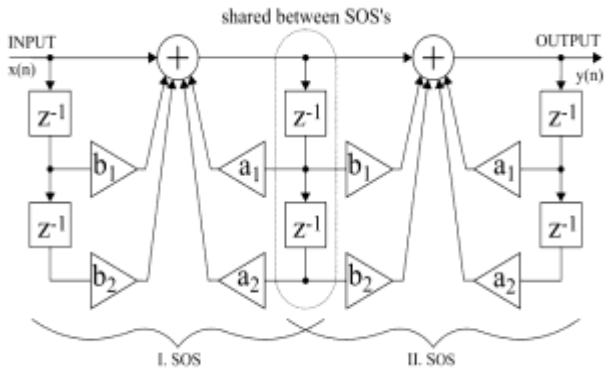

Fig. 3. Resource sharing between SOS's.

Considering the ideal fill, one BRAM can support up to 168 SOS's (in this extreme case, it is necessary to implement arithmetic in modulus 3 which is not particularly efficient for the FPGA implementation and all SOS's share the memory places with exception of the first one and the last one) what implies sample frequency in hundreds ksps.

With appropriate controller, this concept does not restrict implementation to just one filter but allows compute different filters for possibly different channels in one SOS cell based on one RAM, one multiplier and one adder, see Fig. 4. In a straightforward way, the sampling is simultaneous on all inputs so data comes synchronously but the controller can cover cases with different but commensurable sample rates or even with incommensurable sample rates.

TABLE I.
SOS CORE THROUGHPUT AND RESOURCE UTILIZATION

	Spartan-6 at 100 MHz
1 multiplier and 1 adder	12.5 to 20 MSOSps
distributed memory	suitable up to 10 SOS / sample
BRAM 512×36[1]	suitable up to 64 to 168 SOS / sample
BRAM 1024×18[2]	suitable up to 128 to 339 SOS / sample

III. CONTROLLER MICROCODE IMPLEMENTATION

Controller can be designed and implemented with common FSM design techniques (VHDL for example) but this way is complicated and gives unsatisfactory results in FPGA timing and space occupation; future modifications are complicated. As a more efficient way, the controller can be designed utilizing a program counter and RAM (BRAM in case of Spartan-6). A simple assembler language is suitable for writing controller microcode.

Table 2 displays example of computation two SOS's for one signal channel. The first section z-domain transfer function $H(z)$ is:

$$H(z) = \frac{\mathrm{Y(z)}}{\mathrm{X(z)}} = \frac{1-\frac{11}{8}z^{-1}+z^{-2}}{1-\frac{13}{8}z^{-1}+\frac{7}{8}z^{-2}} \quad (1)$$

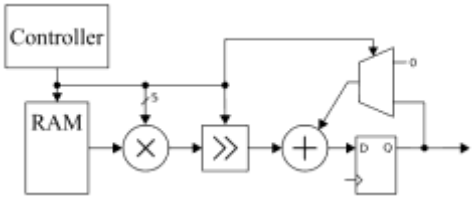

Fig. 4. SOS cell with programmable controller.

[1] Suitable for up to fixed-point int36 or single precision floating-point
[2] Suitable for up to fixed-point int18 or half precision floating-point

TELEN2019008
DOI 10.14311/TEE.2019.2.022

The second section z-domain transfer function $H(z)$ is:

$$H(z) = \frac{\mathrm{Y(z)}}{\mathrm{X(z)}} = \frac{1-\frac{5}{4}z^{-1}+z^{-2}}{1-\frac{5}{4}z^{-1}+\frac{1}{2}z^{-2}} \qquad (2)$$

TABLE II.
AN EXAMPLE OF CONTROLLER MICROCODE
IN SIMPLE ASSEMBLER LANGUAGE

```
input&store port(0), ram(0.0)
load ram(0.0), c=1/1
acc ram(0.2), c=−11/8
acc ram(0.1), c=1/1
acc ram(1.2), c=13/8
acc&store ram(1.1), c=−7/8, ram(1.0)
load ram(1.0), c=1/1
acc ram(1.2), c=-5/4
acc ram(1.1), c=1/1
acc ram(2.2), c=5/4
acc&store&output ram(2.1), c=−1/2, ram(2.0), port(0)
next3
```

IV. FILTER DESIGN AND OPTIMIZATION WORKFLOW

A. Allowed Poles and Zeros Locations

The most important part of the design of particular IIR filter is coefficient set. The width of multipliers increases demands for resources so the design aim is to minimize coefficient mantissa widths. The goal of the method of rounded coefficients is to obtain such coefficients that lead to multiplication and division by small integer numbers. Coefficient optimization is crucial otherwise unpredictable results can occur.

Fig. 5 displays possible pole locations for constants a_1, a_2 obtained as $n/8$ where n is integer. Because poles are complex conjugate, always symmetrical, the picture includes only the upper half of the complex plane.

For complex conjugate poles $p_{1,2} = c \pm d\mathrm{i}$ (and the same for zeros $n_{1,2}$) holds:

$$(z - p_1)(z - p_2) = z + a_1 z^{-1} + a_2 z^{-2}, \qquad (3)$$

where $a_1 = -2c = -2\Re(p_{1,2})$ and $a_2 = c^2 + d^2 = |p_{1,2}|^2$.

Because of these properties, allowed pole locations are grouped in columns with the same real part and in concentric circles with the same magnitude.

B. Poles and Zeros Location Optimization

For given filter poles and zeroes can be obtained using MATLAB with the command

```
[z,p,k]=cheby2(4,20,3/10);
```

Table 3 contains obtained and rounded coefficients and poles and zeros locations.

Fig. 6 displays the results of the original given filter and filter with rounded coefficients. According to the figure, the filter has too high attenuation in the region between 1/10 to 2/10 π·rad/sample.

Other parameters of the filter meet expectations. The attenuation in mentioned region is dominated by pole pairs $p_{1,2}$ from Table 3.

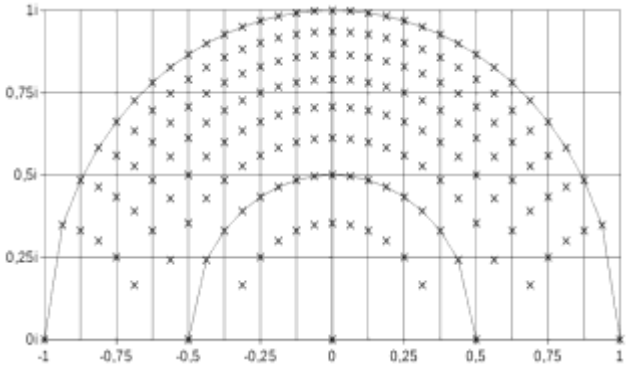

Fig. 5. Pole locations for constants $n/8$.

TABLE III.
POLES AND ZEROS LOCATIONS

	exact location	$n/8$ coefficients	rounded locations
		first iteration of rounding	
$p_{1,2}$	$0.3023 \pm 0.2725\mathrm{i}$	$a_1 = -5/8$; $a_2 = 1/8$	$0.3125 \pm 0.1654\mathrm{i}$
$z_{1,2}$	$-0.2787 \pm 0.9604\mathrm{i}$	$b_1 = 4/8$; $b_2 = 1$	$-0.25 \pm 0.9682\mathrm{i}$
$p_{3,4}$	$0.6015 \pm 0.5783\mathrm{i}$	$a_1 = -9/8$; $a_2 = 6/8$	$0.5625 \pm 0.6585\mathrm{i}$
$z_{3,4}$	$0.5336 \pm 0.8458\mathrm{i}$	$b_1 = -1$; $b_2 = 1$	$0.5 \pm 0.866\mathrm{i}$

TABLE IV.
POLE PAIR $P_{1,2}$ RELOCATION

	p	$\lvert p\rvert^2 = a_2$	arg p
original $p_{1,2}$	$0.3023 \pm 0.2725\mathrm{i}$	0.1656	±0.7337
relocated $p_{1,2}$	$0.3125 \pm 0.1654\mathrm{i}$	0.125	±0.4868

Poles cause peaks in the magnitude transfer characteristics of a filter. An argument (arg p) determines the frequency of a peak whereas an absolute value (abs p) determines the magnitude of the peak.

Rounding quoted in Table 4 moves the pole pair $p_{1,2}$ towards lower frequency and lower magnitude.

Fig. 7 displays other five possible relocations of the pole pair $p_{1,2}$ using constants n/8. Better results can be obtained with a pole pair with different absolute value and argument combination.

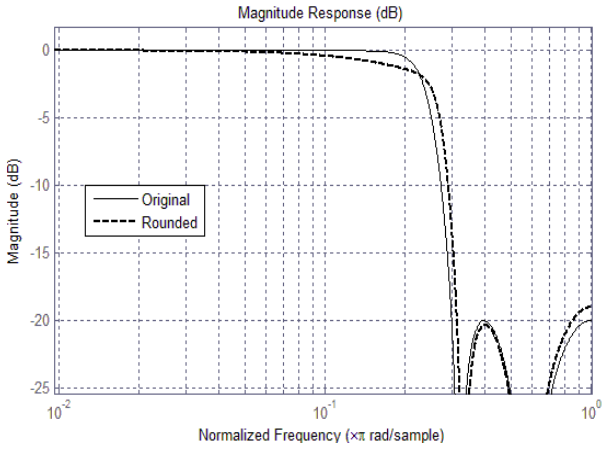

Fig. 6. Influence of coefficients rounding on filter.

TELEN2019008
DOI 10.14311/TEE.2019.2.022

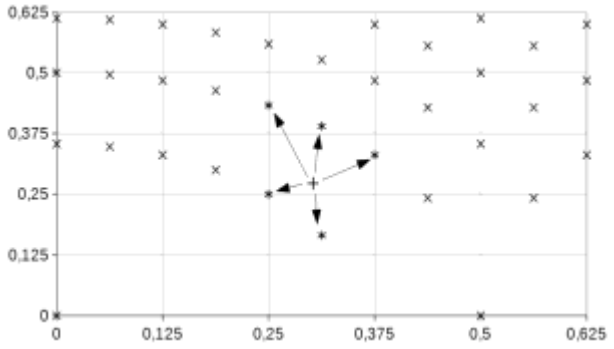

Fig. 7. Other possible pole pair $p_{1,2}$ relocations.

Fig. 8 shows frequency response of the original filter and two other filters with alternative pole pairs $p_{1,2}$ locations. The most suitable results yield the pole [1 –6/8 2/8].

In the matter of zeros, it is interesting to note that many IIR filter design methods place zeros on the complex unit circle ($b_2 = 1$), the only remaining parameter is then b_1. The smallest resolution occurs for filters with very low or very high[3] cutting frequencies. Therefore the method of rounded coefficients is not sufficient for filters with cut off frequency too much low or too much high in comparison to sample frequency. In the case of low frequencies, there might help to decimate the signal first or process the signal via a filter with the finite response (FIR), moving average filter or sinc^k filter can be a good candidate.

Implementation of the filter which operates on the decimated signal causes the poles and zeros are moved to higher values of arg *p*. Moreover, filter with poles close to the complex unity circle has higher demands for accumulation accuracy.

C. Overall Filter Gain

Every 2nd Order Section has its DC gain *g*:

$$g = \frac{b_0+b_1+b_2}{a_0+a_1+a_2} \tag{4}$$

Implementation of IIR filter with accumulator and microcode requires that $a_0 = 1$. Because b_0 is a parameter that can be chosen, the overall gain of all 2nd Order Section (whole filter) can be set to match the final filter realization. For example, the gain of all sections can be matched to 2^n so it is possible to divide the output of the filter just by shifting right.

Resolution of placement of zeros on the complex unit circle can be improved by replacing coefficient $b_0 = b_2$ to some other value than 1.

Regarding the fact that b_1/b_0 ratio need not be only $n/2^k$ but can reach other values n/k (for example 5/7, 6/7, 8/9) zero pairs can be selected from a finer set on complex unity circle.

[3] Near Nyquist frequency (sample frequency / 2)

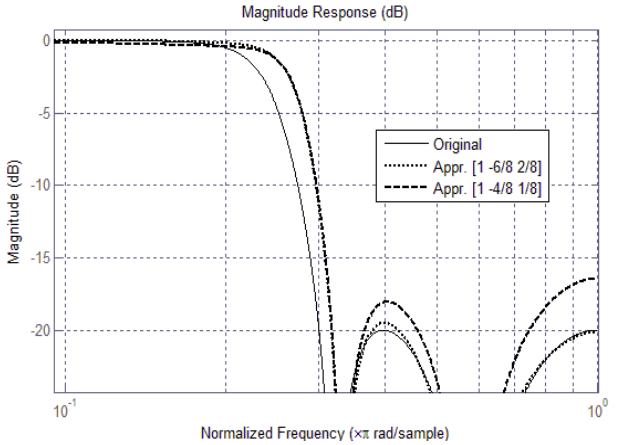

Fig. 8. Influence of relocation of the pole pair $p_{1,2}$.

D. Risk of Arithmetical Overflow

If it is guaranteed that the output of 2nd Order Section accumulator does not overflow (after the last accumulation has been done), there is no need to check overflow during the accumulation process assuming the accumulator uses two's complement representation. The maximal overshoot for each 2nd Order Section should be determined by simulation using step response. For the best results, SOS with the highest gain and the lowest overshoot should be placed first in series of SOS for reaching smaller rounding error (gain) and more arithmetical resources used by valid signal (overshoot).

V. Conclusion

Modern mid-range FPGAs can process IIR filters on very high throughput. IIR filter need not be designed with floating point arithmetic and with full fixed-point or floating-point multipliers. For a great number of filters with cutting frequencies in a reasonable ratio to the sampling frequency, the poles and zeros locations can be optimized so that IIR filter coefficients are round numbers whose multiplication can be realized without a complex multiplier. Utilizing the microcode controller and creating time multiplex allow recycling the FPGA fabric of IIR 2nd Order Section cell in a highly effective way for filtering more signals with different filter coefficients.

References

[1] R. Landry, V. Calmettes and E. Robin, “High speed IIR filter for XILINX FPGA,” in *1998 Midwest Symposium on Circuits and Systems (Cat. No. 98CB36268)*, Notre Dame, IN, 1998, pp.46-49.

[2] S. T. Pan, “Evolutionary Computation on Programmable Robust IIR Filter Pole-Placement Design,” *IEEE Transactions on Instrumentation and Measurement*, vol. 60, no.4, pp.1469-1479, April 2011. https://doi.org/10.1109/TIM.2010.2086850

[3] R. G. Lyons, “Infinite impulse response filters,” in *Understanding digital signal processing*, 2nd ed., New Jersey, USA: Pearson Education Inc. 2008, ch. 6, pp. 211–282.

[4] U. Meyer-Baese, “Infinite impulse response (IIR) digital filters,” in *Digital signal processing with Field Programmable Gate Arrays*, 4th ed., Heidelberg, Germany: Springer-Verlag Berlin Heidelberg. 2014, ch. 4, pp. 225–304. https://doi.org/10.1007/978-3-642-45309-0_4