FIR FILTERS IMPLEMENTATION APPROACHES

Ciprian Romeo Comşa[1], Georgian Grigore[1]

Abstract – This paper discusses different approaches of FIR filters implementation. It presents some basics of digital filters theory, followed by FIR filters design methodology in conventional approach and genetic, evolutionary approach. Finally, it describes the FPGA implementation methodology, highlighting the pipelined architecture of a multiplier accumulator (MAC). A multiplier-less filter approach is also considered.

1. Theoretical background

Digital filters can normally be placed into two categories: Finite Impulse Response (FIR) filters and Infinite Impulse Response (IIR) filters. Generally, the IIR filters are realized by recursive structures, but such structures may also appear in FIR filters too. IIR filters are components of many practical applications because of their better than FIR's magnitude characteristics. However the use of recursive structures is restricted by the stability problem. In applications sensitive to phase distortions, FIR filters are preferred because they may be realized with linear-phase response. FIR filters have uses in a number of applications including noise cancellation, linear prediction and adaptive signal enhancement.

The difference equation for the design of an $(N+1)$-tap causal *FIR filter* with constant coefficients is expressed by (1), where N is the order (number of delay elements) [4]. The output $y[n]$ is the discrete convolution of $x[n]$ with the (finite) impulse response $h[n]$ of the filter.

$$y[n] = \sum_{k=0}^{N} w_k \cdot x[n-k] \tag{1}$$

$$h[k] = \begin{cases} w_k, & k = 0,1,...,N \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

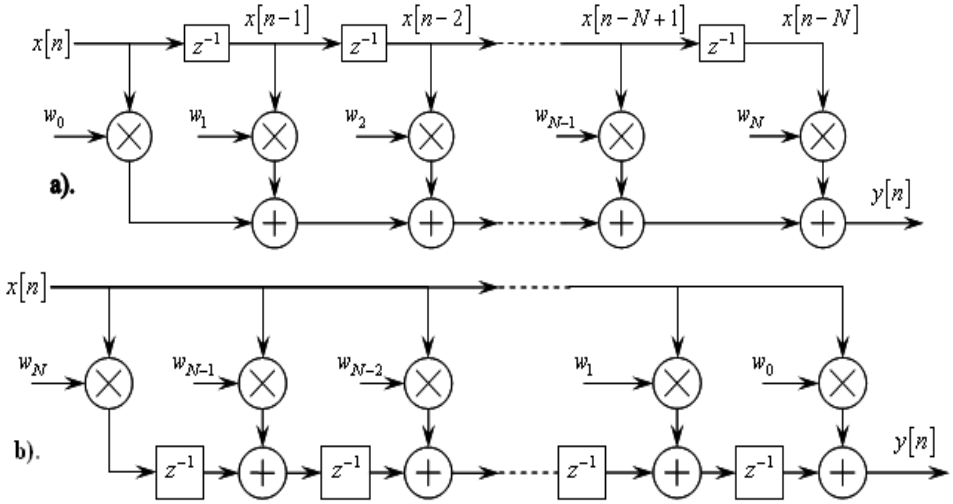

Figure 1. FIR filter structures: *a).* canonical form; *b).* inverted form

From equation (1), the direct form, transversal structure or canonical form (*Figure 1.a*) can be generated, while the transposed direct form structure or inverted form (*Figure 1.b*) is obtained by applying the Transposition Theorem [10].

[1] Faculty of Electronics and Telecommunications, Telecommunications Department, Bd. Carol I No. 11 Iaşi 700506, e-mail: ccomsa@etc.tuiasi.ro, ggrigore@etc.tuiasi.ro

The system has a *linear phase function* (constant group delay: $\tau_g(\omega) = -d\varphi(\omega)/d\omega$) if $h[n]$ satisfy the symmetry (3) or antisymmetry (4) condition [4]. Symmetric (folded) FIR topologies for both canonical and inverted forms are shown in *Figure 2*.

$$h[N-n] = h[n],\ n = 0,1,...,N \tag{3}$$

$$h[N-n] = -h[n],\ n = 0,1,...,N \tag{4}$$

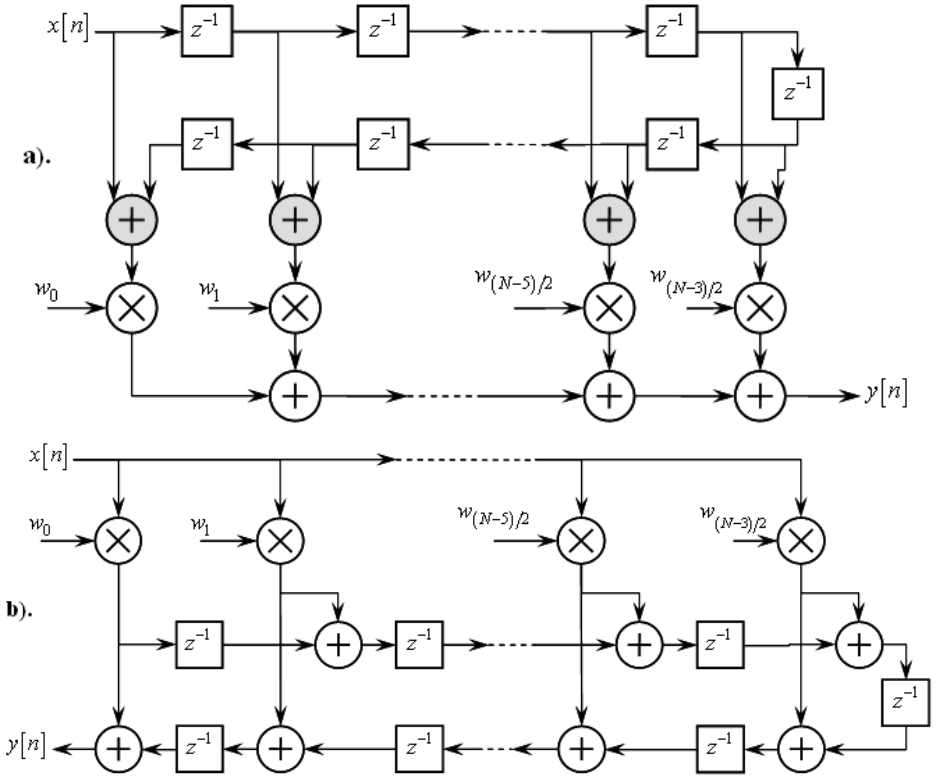

Figure 2. Symmetric FIR filter structures: *a).* canonical form; *b).* inverted form

The input $x[n]$ and the output $y[n]$ of a causal *IIR filter* satisfy the N^{th}-order linear constant coefficients difference equation of the form (5) [4]. Often the coefficient a_0 is assumed to be 1 and we can rewrite the difference equation as (6).

$$\sum_{k=0}^{N} a_k \cdot y[n-k] = \sum_{k=0}^{M} b_k \cdot x[n-k] \tag{5}$$

$$y[n] = \sum_{k=0}^{M} b_k \cdot x[n-k] - \sum_{k=1}^{N} a_k \cdot y[n-k] \tag{6}$$

A filter in the *direct form* is the straightforward implementation of the above difference equation. The system transfer function has the form (7). If we factor the polynomials by finding their roots (roots of numerator polynomial c_k : zeros; roots of the denominator polynomial d_k : poles) we can rewrite the system transfer function in the *cascade form* (8). Combining pairs of real factors and complex conjugate pairs into second-order stages (so called biquads), yields the equation (9).

$$H(z) = \frac{\sum_{k=0}^{M} b_k \cdot z^{-1}}{1 + \sum_{k=1}^{N} a_k \cdot z^{-1}} = \frac{B(z)}{A(z)} \tag{7}$$

$$H(z) = b_0 \cdot \frac{\prod_{k=1}^{M}\left(1 - c_k \cdot z^{-1}\right)}{\prod_{k=1}^{N}\left(1 - d_k \cdot z^{-1}\right)} = b_0 \cdot \frac{z^{-M} \cdot \prod_{k=1}^{M}(z - c_k)}{z^{-N} \cdot \prod_{k=1}^{N}(z - d_k)} = b_0 \cdot \frac{z^{-M}}{z^{-N}} \cdot \frac{C(z)}{D(z)} \tag{8}$$

$$H(z) = b_0 \cdot \prod_{k=1}^{N_{biquad}} \frac{b_{0k} + b_{1k} \cdot z^{-1} + b_{2k} \cdot z^{-2}}{1 + a_{1k} \cdot z^{-1} + a_{2k} \cdot z^{-2}} \quad (9)$$

One advantage of the cascade form over the direct form is that a small change of a coefficient (e.g. quantization) moves only the pair of poles (or zeros) of the corresponding stage and not all others [4]. Furthermore, the amount of displacement is less than for the overall higher-order direct form filter. Another advantage is that we directly can check if the filter is stable by verifying the a_{2k} coefficients only. For a complex conjugate pair of poles d_k, d_k^* we get the equation (10).

$$1 + a_{1k} \cdot z^{-1} + a_{2k} \cdot z^{-2} = 1 - 2 \cdot \mathrm{Re}\{d_k\} \cdot z^{-1} + |d_k|^2 \cdot z^{-2} \quad (10)$$

In the IIR filters design a *stability problem* arises, which is expressed in the following. Let the polynomial $D(z)$ from the expression $1/D(z)$ be with positive exponents of z; the absolute value of this expression converges if and only if $D(z)$ has all the roots inside the unit circle $|z| = 1$ [8]. The recursive filter with the transfer function defined in (8) is stable if all the poles of $H(z)$ are situated inside of the circle with unit radix from the z plane. This condition can be expressed like (11).

$$D(z_k) = 0 \Rightarrow |z_k| \le 1, \forall\, k = 0,...,N-1 \quad (11)$$

If the absolute value of every pole is less than unity than the filter is strictly stable. If there is at least one pole outside the unit circle from the z plane, than the filter is unstable. The *Bairstow* method, recently implemented in C programming language [8], can be successfully used in finding the roots of the denominator of the transfer function $H(z)$.

2. FIR filters design: constraints and solutions

Modern digital FIR filters are designed using computer-aided design engineering tools. The most used tool is the Filter Design & Analysis Tool from Matlab Signal Processing toolbox. The filter is applied to a specific application with a specific magnitude and group delay specification. Because in Matlab codes is difficult to model the specified group delay response, usually there are two stages: first, design a filter that meets the specification in magnitude response and should be minimum-phase; than, consider trade-offs between the group delay specification and implementation costs [5].

Most often, we already know the transfer function (i.e., magnitude of the frequency response) of the desired filter. Such a lowpass specification consists of the passband $W_{pass} = [0...F_{pass}]$, the transition band $[F_{pass}...F_{stop}]$ and the stopband $W_{stop} = [F_{stop}...F_s/2]$ specification, where the sampling frequency is assumed to be F_s. To compute the filter coefficients, we may apply one of the methods implemented in Matlab (i.e. the direct frequency method or equiripple method) [6].

If we want to implement the designed filter in ASIC or FPGA technologies than scaling and quantization has to be applied to calculated coefficients, translating them from floating point values to fixed-point. So, if we like to calculate expression (12) using integer arithmetic, where b_k is the equivalent of w_k from the equation (1) and *Figure 1* and should have fractional value (e.g. $b_k = 0.125$), we must scale it up to get an useful integer before the multiplication and scale it down again either directly after multiplication or after the accumulation. Let Q denote this scaling factor and for simplicity, let's choose a power of 2 (e.g. $Q = 2^{15}$) which can be applied using bitwise shift. Therefore the finest fractional resolution (i.e., quantization step) is Q^{-1}. Now the expression has the form (13) or (14). The second one produces definitely less round-off noise (σ_n^2 instead of $(M+1) \cdot \sigma_n^2$) but it needs a wider accumulator (double-length or more) [4].

$$\sum_{k=0}^{M} b_k \cdot x[n-k] \quad (12)$$

$$\sum_{k=0}^{M} Q^{-1} \cdot \left(b_k \cdot Q \cdot x[n-k]\right) \quad (13)$$

$$Q^{-1} \cdot \sum_{k=0}^{M} b_k \cdot Q \cdot x[n-k] \quad (14)$$

The Discrete Fourier Transform (DFT) establishes a direct connection between the frequency response and the time domain behaviour. Since the frequency domain is the domain of filter definition,

the DFT can be used to calculate a set of FIR coefficients which produce a filter that approximates the frequency response of the target filter. A filter designed in this manner is called a direct FIR filter, which is defined by (15). In order to smooth the magnitude frequency response, a data windowing may be applied to the FIR, for instance a Kaiser window [5].

$$f[n] = \mathrm{IDFT}\left(F[k]\right) = \sum_{k} F[k] \cdot e^{j2\pi kn/L} \tag{15}$$

$$L = \frac{-10 \cdot \log_{10}\left(A_{stop} \cdot A_{pass}/2\right)}{2.324 \cdot 2\pi \cdot \left(F_{stop} - F_{pass}\right)} + 1 \tag{16}$$

A typical filter specification not only includes the specification of passband F_p and stopband F_s frequencies and ideal gains, but also the allowed deviation (or ripple) from the desired transfer function. The transition band is most often assumed to be arbitrary in terms of ripple. A special class of FIR filter that is particularly effective in meeting such specifications is called the equiripple FIR. An equiripple design protocol minimizes the maximal deviations (ripple error) from the ideal transfer function. The equiripple or minimum-maximum (minimax) algorithm is normally implemented using Parks-McClellan iterative method. The length of the polynomial, and therefore the filter, can be estimated for a lowpass with (16), where $A_{pass}/2$ is the passband and A_{stop} is the stopband ripple [6].

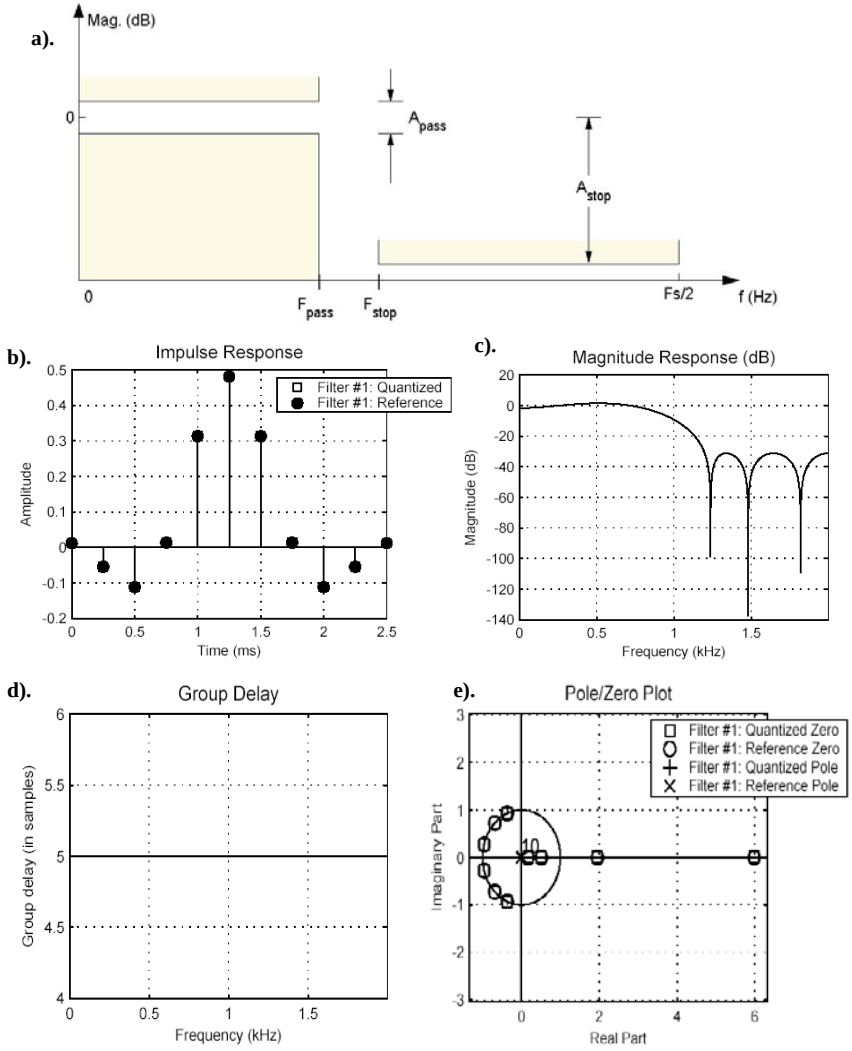

Figure 3. FIR filter analysis with FDAtool

Figure 3 illustrates the design by the equiripple method using the Matlab FDA tool for an quantized direct form symmetric FIR filter with minimum order and the specifications: $F_s = 4000Hz$, $F_{pass} = 800Hz$, $F_{stop} = 1200Hz$, $A_{pass} = 2dB$, $A_{stop} = 35dB$. The resulting filter has the order 10 and the coefficients are: 0.01187133789, -0.05444335938, -0.1121520996, 0.01327514648, 0.313293457, 0.4811096191, 0.313293457, 0.01327514648, -0.1121520996, -0.05444335938, 0.01187133789. The filter structure can be also automatically generated.

3. Genetic representation of FIR filters

In the design of the digital filters, a special approach is the genetic one [2]. Usually, a genetic algorithm is only applied to optimize the coefficients of a digital filter obtained in a conventional manner. A completely different design stile consists in having the whole design performed by an evolutionary algorithm. Evolutionary algorithms are a broad class of optimization methods, built on the key concept of Darwin evolution in biology. A digital filter can be represented as a sequence of elementary operations, which can be encoded to be handled by a genetic algorithm. The conventional approach, illustrated by the dashed line in *Figure 4a* consists in the design of an ideal specified filter (with infinite precision coefficients) by obtaining an approximation with finite word arithmetic. The genetic algorithm is used to produce from filter specifications directly the synthesizable RTL (register transfer logic) code, which is translated into structural and physical domains by means of other tools.

$$H(z) = \sum_{k=0}^{M} b_k \cdot z^{-k} \tag{17}$$

A description of digital filters can derived from their frequency response in the z-domain. The frequency response of a finite impulse response (FIR) digital filter is given by (17) and the canonical direct form of the filter is shown in *Figure 1a*. To save area and to reduce power consumption, generic multiplier blocks are often replaced with shifters and adders. As an example, multiplication by 13 can be implemented with two shifts and two additions, as illustrated in *Figure 4b*, where the block "<< n" means "left shift by n positions". The canonical signed digit (CSD) representation assigns a separate sign to each digit: 0, 1 and $\bar{1}$ (= –1). Its goal is to minimize the number of non-zero digits: by encoding the filter coefficients with CSD, the filter output can be computed using a reduced amount of hardware, since multiplications by zero are simply not implemented. As an example, consider the multiplication by 15: since $15 = 2^3 + 2^2 + 2^1 + 2^0 = (001111)_2$, this operation in binary arithmetic would require three shifts and three additions; while using CSD we can write $15 = 2^4 - 2^0 = (010001)_2$, and we implement the same operation using only one shifter and one subtractor (*Figure 4c*).

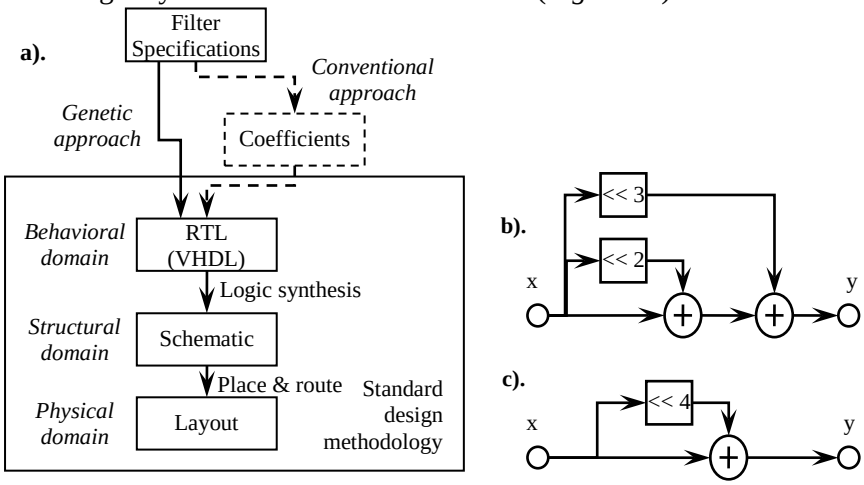

Figure 4. a). Digital design methodology; *b).* Multiplication by 13 implemented with shifters and adders; *c).* Multiplication by 15 implemented with one shifter and one adder

Name	*Code*	*Op 1*	*Op 2*	*Description*
Input	I	not used	not used	Copy input: $y_i = x$
Delay	D	n_1	not used	Store value: $y_i = y_{i-n_1} z^{-1}$
Left shift	L	n_1	p	Multiply by 2^p: $y_i = 2^p y_{i-n_1} z^{-1}$
Right shift	R	n_1	p	Divide by 2^p: $y_i = 2^{-p} y_{i-n_1} z^{-1}$
Adder	A	n_1	n_2	Sum: $y_i = \left(y_{i-n_1} + y_{i-n_2}\right) z^{-1}$
Subtractor	S	n_1	n_2	Difference: $y_i = \left(y_{i-n_1} - y_{i-n_2}\right) z^{-1}$
Complement	C	n_1	not used	Multiply by -1: $y_i = -y_{i-n_1} z^{-1}$

Table 1. Primitives of the genetic algorithm

Starting from these considerations, a digital filter can be described using a very small number of elementary operations. The primitives selected for digital filters are listed in *Table 1* [2]. Each elementary operation is encoded by its own code (one character) and by two integer numbers, which represent the relative offset (calculated from the current position) of the two operands. When all the offsets are positive (i.e. each block can receive data from previous blocks only), no feedback loop occurs and the resulting structure is a FIR filter. All primitives include a delay z^{-1}, to avoid possible problems due to timing violations during the synthesis process. Since the primitive operators requiring an adder block are more expensive in terms of power dissipation, a relative weight factor may be assigned to sum, deference and complement. As an example, the following sequence is made of 6 primitives (6 genes): (I 0 2) (D 1 3) (L 2 2) (A 2 1) (D 1 0) (S 1 5). It corresponds to the schematic diagram shown in *Figure 5*, and it is interpreted as in (18). The last value is the output of the filter. By merging the equations (18), we obtain the transfer function (19).

$$\begin{aligned} y_0 &= x \qquad y_2 = 2^2 y_0 z^{-1} \qquad y_4 = y_3 z^{-1} \\ y_1 &= y_0 z^{-1} \quad y_3 = (y_1 + y_2) z^{-1} \quad y = y_5 = (y_4 - y_0) z^{-1} \end{aligned} \tag{18}$$

$$H(z) = y/x = 5z^{-4} - z^{-1} \tag{19}$$

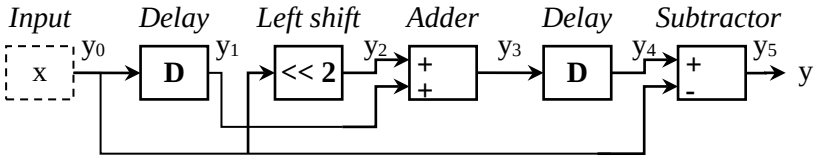

Figure 5. Schematic diagram corresponding to a sequence of 6 primitives

Such a representation has the same essence as a program in a simple programming language. Therefore, digital filter design can be automated through genetic programming. Fine granularity of primitives leads to a simple genetic encoding, and allows the evolutionary algorithm to perform a better search within the design space [2].

5. On the FPGA implementation of FIR filters

Most digital signal processing done today uses a specialized microprocessor, called a digital signal processor, capable of very high speed multiplication. This traditional method of signal processing is bandwidth limited. There is a fixed number of operations that the processor can perform on a sample before the next sample arrives. This limits either the applications that can be performed on a signal or it limits the maximum frequency signal that the application can handle. This limitation stems from the sequential nature of processors. DSPs using a single core can only perform one operation on one piece of data at a time. They can not perform operations in parallel. For example, in a 64 tap filter they can only calculate the value of one tap at a time, while the other 63 taps wait. Nor can they perform pipelined applications. In an application calling for a signal to be filtered and then correlated, the processor must first filter, then stop filtering, then correlate, then stop correlating, then filter, etc. If the applications could be pipelined, a filtered sample could be correlated while a new sample is simultaneously filtered. Digital Signal Processor manufacturers have tried to get around this problem by cramming additional processors on a chip. This helps, but it still remains true that in a digital signal processor most of the application is idle most of the time [1].

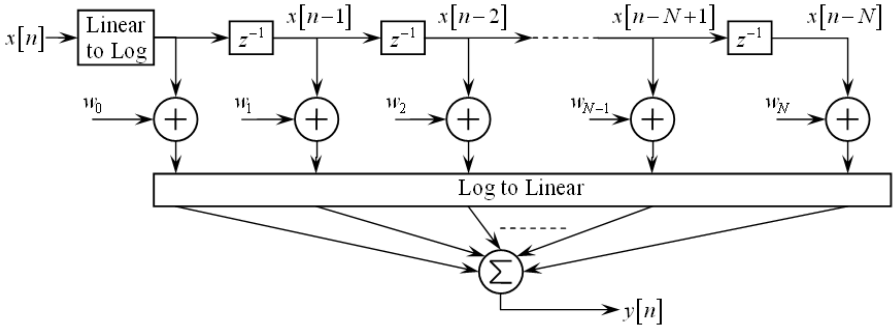

Figure 6. Multiplier-less transversal FIR filter

FPGA-based digital signal processing is based on hardware logic and does not suffer from any of the software based processor performance problems. FPGAs allow applications to run in parallel so that a 128 tap filter can run as fast as a 10 tap filter. Applications can also be pipelined in an FPGA, so

that filtering, correlation, and many other applications can all run simultaneously. In an FPGA, most of an application is working most of the time. An FPGA can offer 10 to 1000 times the performance of the most advanced digital signal processor at similar or even lower costs [1], [7].

Conventional transversal filters require a multiplying element for each tap. Multiplication is a resource and time consuming process. Of course, one approach is to use a pipelined architecture of a multiplier accumulator [9], [7] (MAC), equivalent to a transversal filter. But also, there is another fast way by carrying out the multiplication in the logarithmic domain to save time. The filter architecture is shown in *Figure 6* and it is based on the fact that any binary number N can be written as in (20) and for $0 \le x < 1$ the approximations from (21) are available [3].

$$N = 2^k \cdot (1 + x) \tag{20}$$

$$\log_2 N = k + \log_2 (1 + x) \approx k + x \tag{21}$$

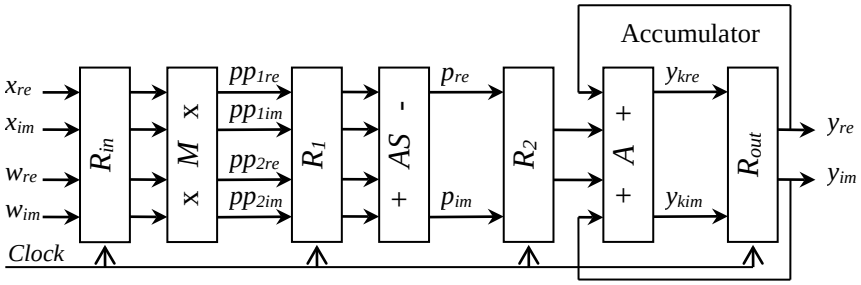

Figure 7. Pipelined FIR – MAC structure

$$\begin{aligned} pp_{1re} &= x_{re} \cdot w_{re} \\ pp_{1im} &= x_{re} \cdot w_{im} \qquad p_{re} = pp_{1re} - pp_{2re} \qquad y_{kre} = y_{re} + p_{re} \\ pp_{2re} &= x_{im} \cdot w_{im} \qquad p_{im} = pp_{1im} + pp_{2im} \qquad y_{kim} = y_{im} + p_{im} \\ pp_{2im} &= x_{im} \cdot w_{re} \end{aligned} \tag{22}$$

The pipelined architecture of a complex multiplier accumulator, which can be used in transversal filters implementation is presented in *Figure 7* [9]. It contains one input register R_{in}, one output register R_{out} and two pipeline registers R_1, R_2. The multiplication block *M* calculates the partial products pp_1 and pp_2 and the adder/subtractor block *AS* computes the final product *p* (22). The final product is accumulated using the accumulator block *A*. At each active front clock, the blocks *M*, *AS* and *A* do their functions and the results are stored in the pipeline R_1 and R_2 and output R_{out} registers and the next filter input samples are prepared in the input register R_{in}. The structure presented is sequential, so the samples of filter's input *x* and coefficients *w* have to be synchronized. Also, a *clear* signal to reset the accumulator is needed. For controlling this, a finite state machine is most adequate.

6. Conclusions

The FIR filters implementation may be approached mainly by two ways: the conventional one ant the evolutionary, genetic one. In the conventional approach, the filter's coefficients have to be calculated from the filter specifications, while in the evolutionary approach synthesizable RTL code is directly produced from the filter specifications using a genetic algorithm. For the conventional approach there are developed computer analysis tools like Matlab Filter Design and Analysis tool. Trough genetic programming, the digital filter design can be automated.

FPGA implementations of FIR filters run in parallel, which is done faster than the software implementations. Pipeline registers may be introduced in FPGA implementations and in this manner the time needed to calculate the value of one sample of the filter's output is reduced to the time period of the slowest register and not to the sum of time periods of all registers.

References

[1] Allaire B, Fischer B., "Block Adaptive Filter", Xilinx Application Note XAPP 055 1997

[2] Azzini A., Bettoni M., Liberali V., Rossi R., Tettamanzi A., "Evolutionary Design and FPGA Implementation of Digital Filters", University of Milano, 2003

[3] El-Eraki S.M., Batchelor J.C., Lee P., Langley R.J., "A Multiplier-less CMA Adaptive Equaliser", University of Kent at Canterbury

[4] Feldbauer Ch., "Digital Filter Implementation", http://spsc.inw.tugraz.at, 2002

[5] Guo Zhan, "Digital Filter Design: A Practical Prototyping Approach", Lund University, 2003

[6] Mayer-Baese U., "Digital Signal Processing with FPGA", Springer Verlag Berlin, 2001

[7] Parhi K.K., “Pipelining in Algorithms with Quantizer Loops”, IEEE Transactions on Circuits and Systems, vol. 38, No. 7, July 1991
[8] Rusu, I., „O Nouă Metodă de Determinare a Stabilităţii Filtrelor Numerice Recursive”, Telecomunicaţii, Nr.1/2001
[9] Stoica L., “A VHDL Pipeline Control Unit Model Approach of a Pipelined Multiplier Accumulator”, Buletinul Ştiinţific al Univ. „Politehnica” din Timişoara, Transactions on Electronics and Telecommunications, Tom 47(61), Fascicula 1-2, 2002
[10] Valls J., Peiro M., Sansaloni T., Boemo E., “A Study About FPGA-Based Digital Filtres”