NATIONAL TECHNICAL UNIVERSITY OF UKRAINE
«IGOR SIKORSKY POLYTECHNICAL INSTITUTE»

Informatic and Computer Engineering Faculty
Computer Engineering Department

«On the rights of the manuscript»
УДК __004.942_____

«Defence is allowed»
Head of the Computer Science Dep-t

_____ S.G. Stirenko
(sign) (name)

“___”_____2018p.

Master's thesis

In speciality _____123 Computer Engineering_____

Specialization: _____123. Computer systems and networks _____

theme: _____Method of increasing the efficiency_____
_____ of the finite impulse response digital filters _____

Fulfilled: student of VI course, group _ IO 64м _____
(group sign)

_____Quadir Safwan Husein _____ _____
(Full Name) (signature)

Науковий керівник Ass.Prof., Dr.Sci, S.Sci. Sergiyenko A.M. _____ _____
(position, scientific degree, academic rank, surname and initials) (signature)

Reviewer Ass.Prof., Dr.Sci, Docent Romankevich V.O._____ _____
(position, scientific degree, academic rank, surname and initials) (signature)

I certify that in this master's thesis there are no borrowings from the works of other authors without the corresponding references.
Student _____
(signature)

Kyiv – 2018

РЕФЕРАТ

Метод підвищення ефективності нерекурсивних цифрових фільтрів.

Актуальність теми. Для швидкісного вирішення задач цифрової обробки сигналів, зокрема для обробки сигналів фільтрами зі скінченною імпульсною характеристикою (СІХ) широко використовуються програмовані логічні інтегральні схеми (ПЛІС). Реалізація СІХ-фільтрів у ПЛІС основана на широкому використанні апаратних блоків множення. При цьому решта програмованих ресурсів використовується нераціонально. Отже, для підвищення ефективності використання програмованих ресурсів ПЛІС та зменшення їх енергоспоживання необхідне впровадження більш досконалих структур СІХ-фільтрів.

Об'єктом дослідження є проектування процесорів для цифрової обробки сигналів.

Предметом дослідження є розробка високопродуктивних конвеєрних СІХ-фільтрів.

Мета роботи: створення методу проектування високопродуктивних СІХ-фільтрів, призначених для конфігурування в ПЛІС.

Наукова новизна полягає в наступному:

Розроблено метод підвищення ефективності нерекурсивних цифрових фільтрів, який оснований на тому, що блоки множення на коефіцієнти, що мають невелику амплітуду, заміняються на спеціалізовані блоки множення, за рахунок чого зменшуються апаратні витрати збільшується пропускна спроможність фільтрів.

Практична цінність отриманих в роботі результатів полягає в тому, що цифрові фільтри, які розроблені за новим методом, дозволяють підвищити ефективність систем обробки сигналів на базі ПЛІС за рахунок зменшення їхньої вартості або збільшення продуктивності.

Матеріали роботи використані у науково-дослідній роботі «Удосконалені методи та засоби проектування конфігурованих комп'ютерів на

основі відображення просторового графу синхронних потоків даних у структури на базі програмованих логічних інтегральних схем», № ДР.047U005087, шифр ФІОТ-30Т/2017, яка проводиться у НТУУ “КПІ ім. Ігоря Сікорського.

Апробація роботи. Основні положення і результати роботи були представлені та обговорювались на 20-тій Міжнародній конференції «Системний аналіз та інформаційні технології» SAIT-2018 21 – 24 травня 2018 року, Київ та міжнародній конференції "Безпека, Відмовостійкість, Інтелект" 10 – 12 травня 2018 року, Київ.

Структура та обсяг роботи. Магістерська дисертація складається зі вступу, трьох розділів та висновків.

У вступі подано загальну характеристику роботи, зроблено оцінку сучасного стану проблеми розробки систем цифрової обробки сигналів на ПЛІС, обґрунтовано актуальність напрямку досліджень, сформульовано мету і задачі досліджень, показано наукову новизну отриманих результатів і практичну цінність роботи, наведено відомості про апробацію результатів і їхнє впровадження.

У першому розділі досліджено особливості архітектури сучасних ПЛІС, розглянуто алгоритми цифрової фільтрації та їх відомі реалізації в ПЛІС.

У другому розділі розроблено метод підвищення ефективності нерекурсивних цифрових фільтрів, які реалізуються у ПЛІС.

У третьому розділі досліджено ефективність використання запропонованого методу та порівняння його з існуючими методами на основі ряду прикладів проектування СІХ-фільтрів.

У висновках представлені результати проведеної роботи.

Робота представлена на 93 аркушах, містить посилання на список використаних літературних джерел.

Ключові слова: ПЛІС, імпульсна характеристика, нерекурсивний фільтр, граф синхронних потоків даних, конвеєр.

ABSTRACT

Method of increasing the efficiency of the finite impulse response digital filters.

Relevance of the topic. Field programmable gate arrays (FPGAs) are widely used for the high-speed digital signal processing (DSP) in particular for processing by the finite impulse response (FIR) filters. The FIR filter implementation in FPGA is based on the widespread use of hardware multiplication blocks. At the same time, the rest of the FPGA programmable resources are used ineffectively. Therefore, in order to increase the efficiency of the use of the FPGA programmable resources and to reduce their energy consumption, it is necessary to introduce more perfect FIR filter structures.

The object of the research is designing of high-performance processors for the digital signal processing.

The subject of the research is development of the high-performance pipelined FIR filters.

The objective is the creation of a method for designing the high-performance FIR filters which are intended for configuring in FPGA.

The scientific novelty is as follows:

The method of increasing the efficiency of the FIR digital filters, which is based on the fact that the constant coefficient multiplication blocks for the small coefficients are replaced by the specialized blocks, due to which the hardware costs decreases, and the throughput of the filters increases.

The practical value of the obtained results is that the digital filters developed by the new method allows to improve the efficiency of the DSP systems on the basis of FPGA by reducing their cost or increase the speed.

The materials of the thesis were used in the research work "Advanced methods and tools of designing the configurable computers on the basis of

mapping the spatial synchronous data flow graphs into the structure for FPGA", № ДР.047U005087, ФІОТ-30Т / 2017, which is held at NTUU "Igor Sikorsky's KPI".

Approbation of the work. Substantive provisions and results of the work were presented and discussed at the 20-th International conference "System Analysis and Infirmational Technologies", SAIT-2018, May, 21 – 24, 2018, Kyiv, and at the International Conference on Security, Fault Tolerance, Intelligence (ICSFTI2018), May 10 – 12, 2018, Kyiv.

The structure and scope of the work. Master's thesis consists of an introduction, four sections and conclusions.

The introduction provides a general description of the work carried out to assess the current state of the problem, justified the relevance of research areas, formulate goals and objectives of research shows the scientific novelty of the obtained results and the practical value of the work, it provides the information about the testing results.

In the first section

In the second section a method for increasing the efficiency of non-recursive digital filters implemented in FPGA is developed.

The first section deals with the features of the modern FPGA architecture, algorithms of digital filtration and their known implementation in FPGA. Here, the basic methods of designing the pipelined application-specific processors, algorithms and structures for calculating the FIR filters are considered.

The second section describes a method for increasing the efficiency of FIR digital filters implemented in FPGA.

In the third section, the efficiency of using the proposed method is investigated and it is compared with the existing methods of the FIR filter design.

The conclusions describe the results of the work.

The work submitted 93 sheets, contains a list of references to the used literature.

Key words: FPGA, FIR filter, pipeline, synchronous dataflow graph.

РЕФЕРАТ

Метод повышения эффективности нерекурсивных цифровых фильтров

Актуальность темы. Для скоростного решения задач цифровой обработки сигналов, в частности для обработки сигналов фильтрами с конечной импульсной характеристикой (КИХ) широко используются программируемые логические интегральные схемы (ПЛИС). Реализация КИХ-фильтров в ПЛИС основана на широком использовании аппаратных блоков умножения. При этом остальные программируемые ресурсы используется нерационально. Следовательно, для повышения эффективности использования программируемых ресурсов ПЛИС и уменьшения их энергопотребления необходимо внедрение более совершенных структур КИХ-фильтров.

Объектом исследования является проектирование высокопроизводительных процесссоров для цифровой обработки сигналов.

Предметом исследования является разработка высокопроизводительных конвейерных КИХ-фильтров.

Цель работы: создание метода проектирования высокопроизводительных КИХ-фильтров, предназначенных для конфигурирования в ПЛИС.

Научная новизна заключается в следующем:

Разработан метод повышения эффективности нерекурсивных цифровых фильтров, который основан на том, что блоки умножения на коэффициенты, которые имеют небольшую амплитуду заменяются на специализированные блоки умножения, за счет чего уменьшаются аппаратные затраты увеличивается пропускная способность фильтров.

Практическая ценность полученных в работе результатов заключается в том, что цифровые фильтры, разработанные по новому методу, позволяют повысить эффективность систем обработки сигналов на базе ПЛИС за счет уменьшения их стоимости или увеличения производительности.

Материалы работы использованы в научно-исследовательской работе «Усовершенствованные методы и средства проектирования конфигурируемых компьютеров на основе отображения пространственного графа синхронных потоков данных в структуры на базе программируемых логических интегральных схем», № ДР.047U005087, шифр ФІОТ-30Т / 2017, которая проводится в НТУУ "КПИ им. Игоря Сикорского".

Апробация работы. Основные положения и результаты работы были представлены и обсуждались на 20-ой Международной конференции «Системный анализ и информационные технологии» SAIT-2018, 21 — 24 мая 2018, Киев и международной конференции "Безопасность, Отказоустойчивость, Интеллект" 10 — 12 мая 2018, Киев.

Структура и объем работы. Магистерская диссертация состоит из введения, трех глав и выводов.

Во введении представлена общая характеристика работы, произведена оценка современного состояния проблемы разработки систем цифровой обработки сигналов на ПЛИС, обоснована актуальность направления исследований, сформулированы цели и задачи исследований, показано научную новизну полученных результатов и практическую ценность работы, приведены сведения об апробации результатов и их внедрение.

В первом разделе исследованы особенности архитектуры современных ПЛИС, рассмотрены алгоритмы цифровой фильтрации и их известные реализации в ПЛИС.

Во втором разделе разработан метод повышения эффективности нерекурсивных цифровых фильтров, которые реализуются в ПЛИС.

В третьем разделе исследована эффективность использования предложенного метода и сравнение его с существующими методами на основе ряда примеров проектирования КИХ-фильтров.

В выводах представлены результаты проведенной работы.

Работа представлена на 93 листах, содержит ссылки на список использованных литературных источников.

Ключевые слова: ПЛИС, импульсная характеристика, а не рекурсивный фильтр, граф синхронных потоков данных, конвейер.

CONTENT

ABBREVIATIONS

ASIC	Application Specific Integrated Circuit
CCM	Constant Coefficient Multiplier
CSD	Canonic Signed Digit (presentation)
DSP	Digital Signal Processing
DFG	Data Flow Graph
FIR	Finite Impulse response
FPGA	field programmable gate array
GPU	Graphic Processing Unit
IC	Integrated Circuits
IIR	Infinite Impulse Response
IP core	Intellectual Property core
LUT	Look-Up Table
MSD	Minimum Signed Digit (presentation)
PU	Processing Unit
RAM	Random Access Memory
ROM	Read-Only Memory
RTL	Register Transfer Logic
SDF	Synchronous Data Flow graph
VHSIC	Very High Speed Integrated Circuits
VHDL	VHSIC Hardware Description Language
CPU	Central Processing Unit
VLSI	Very Large Scale Integration

INTRODUCTION

Finite impulse response (FIR) filter structure development is expanded since 60-ies [1,2]. But the most of FIR implementations are the program ones, for example, in the DSP processors [3]. Field programmable gate arrays (FPGAs) are widely used for the high-speed digital signal processing (DSP) in particular for processing by the FIR filters.

The leading FPGA companies provide the FIR IP core generators [4.5]. The generated FIR filter implementation in FPGA is based on the widespread use of hardware multiplication blocks. At the same time, the rest of the FPGA programmable resources are used ineffectively. Besides, the regulated parameters of the generated FIR filter structures, such as bit width, filter length, are limited. The user has none opportunity to infer the filter structure. Therefore, in order to increase the efficiency of the use of the FPGA programmable resources and to reduce their energy consumption, it is necessary to introduce more perfect FIR filter structures.

The object of the research is the high-performance processors for the digital signal processing.

The subject of the research is the structure of the high-performance pipelined FIR filters.

The objective is the creation of a method for designing the high-performance FIR filters which are intended for configuring in FPGA.

The scientific novelty is as follows:

The method of increasing the efficiency of the FIR digital filters, which is based on the fact that the constant coefficient multiplication blocks for the small coefficients are replaced by the specialized blocks, due to which the hardware costs decreases, and the throughput of the filters increases.

The practical value of the obtained results is that the digital filters developed by the new method allows to improve the efficiency of the DSP systems on the basis of FPGA by reducing their cost or increase the speed. The FIR filters designed using this method, can be introduced in any project where needed, because they are described in VHDL language. A tool box can be developed, which is able to generate the filter IP cores and compete with the generators described in [4.5].

The materials of the thesis were used in the research work "Advanced methods and tools of designing the configurable computers on the basis of mapping the spatial synchronous data flow graphs into the structure for FPGA", № ДР.047U005087, ФІОТ-30Т / 2017, which is held at NTUU “Igor Sikorsky’s KPI”.

Approbation of the work. Substantive provisions and results of the work were presented and discussed at the 20-th International conference “System Analysis and Infirmational Technologies”, SAIT-2018, May, 21 – 24, 2018, Kyiv, and at the International Conference on Security, Fault Tolerance, Intelligence (ICSFTI2018), May 10 – 12, 2018, Kyiv.

The structure and scope of the work. Master's thesis consists of an introduction, four sections and conclusions.

The introduction provides a general description of the work carried out to assess the current state of the problem, justified the relevance of research areas, formulate goals and objectives of research shows the scientific novelty of the obtained results and the practical value of the work, it provides the information about the testing results.

The first section deals with the features of the modern FPGA architecture, algorithms of digital filtering and their known implementation in FPGA. Here, the basic methods of designing the pipelined application-specific processors, algorithms and structures for calculating the FIR filters are considered.

The second section describes the development of a method for increasing the efficiency of FIR digital filters implemented in FPGA.

In the third section, the efficiency of using the proposed method is investigated and it is compared with the existing methods of the FIR filter design.

The conclusions describe the results of the work.

Publications of the work.

The main features of these investigations are published in two works. In the work [6] the author has proposed an approach, which provides the improveing the FIR filter performance. In the work [2] the author has proposed the way how to decrease the application specific multiplier hardware cost, and provides the experimental results.

The work submitted 93 sheets, contains a list of references to the used literature.

1 FINITE IMPULSE RESPONSE FILTER HARDWARE DESIGN

1.1 Basics of the FIR filters

1.1.1 Basic definitions

The linear systems such as filters are divided into FIR and Infinite Impulse response (IIR) filters. FIR filters have a number of following advantages:

— FIR filters are always stable, because they do not have poles and feedbacks, through which there may be excites;

— it is always guaranteed to construct a linear phase filter;

— rounding errors in FIR filters have much less effect on the result than in IIR filters;

— FIR filters are effectively implemented in microprocessors that have multiplication commands with the addition of the accumulator.

Amplitude-Frequency Characteristic (AFC) $|H(\omega)|$ of the filter specifies a specific gain at certain frequencies, and the Phase-Frequency Characteristic (PFC) $\varphi(\omega) = \arg(H(\omega))$ reflects the effect of the delay or phase shift of the signal at these frequencies. The group delay function is defined as

$$T_d(\omega) = -\frac{d\varphi(\omega)}{d\omega}. \quad (1.1)$$

The PFC of the linear phase filter is described by the formula:

$$\varphi(\omega) = -a\omega \text{ or } \varphi(\omega) = \pi - a\omega . \quad (1.2)$$

Therefore, according to (1.2), in a linear phase filter, the group delay of the signal is constant for all frequencies. Such a filter does not distort the phase of the signal and, accordingly, does not lose its shape (if its spectrum falls into the bandwidth), because all its frequency components are delayed by the same delay.

In the form of AFC, the filters are divided into low-pass filter (LPF), high-pass filters (HPF), bandpass filters (BPF) and rejector filters (RF). Because the AFC of a filter with the real coefficients is a symmetric function of ω, this AFC is set in the range $0 \le \omega \le \pi$.

The frequency response of the ideal LPF is shown in Fig. 1.1, a. The bands $0\ 0 \le \omega \le \omega_{зр}$ and $\omega > \omega_{зр}$ are called *passband* and *stopband*, respectively, and the frequency $\omega_{зр}$, which separates these bands, is the *cutoff frequency*. The ideal low-pass filter in the range $0 \le \omega \le \omega_{зр}$ has a frequency response $|H(\omega)| = 1$, and in the range $\omega > \omega_{зр}$ the response $|H(\omega)| = 0$ (see Fig. 1.1, a).

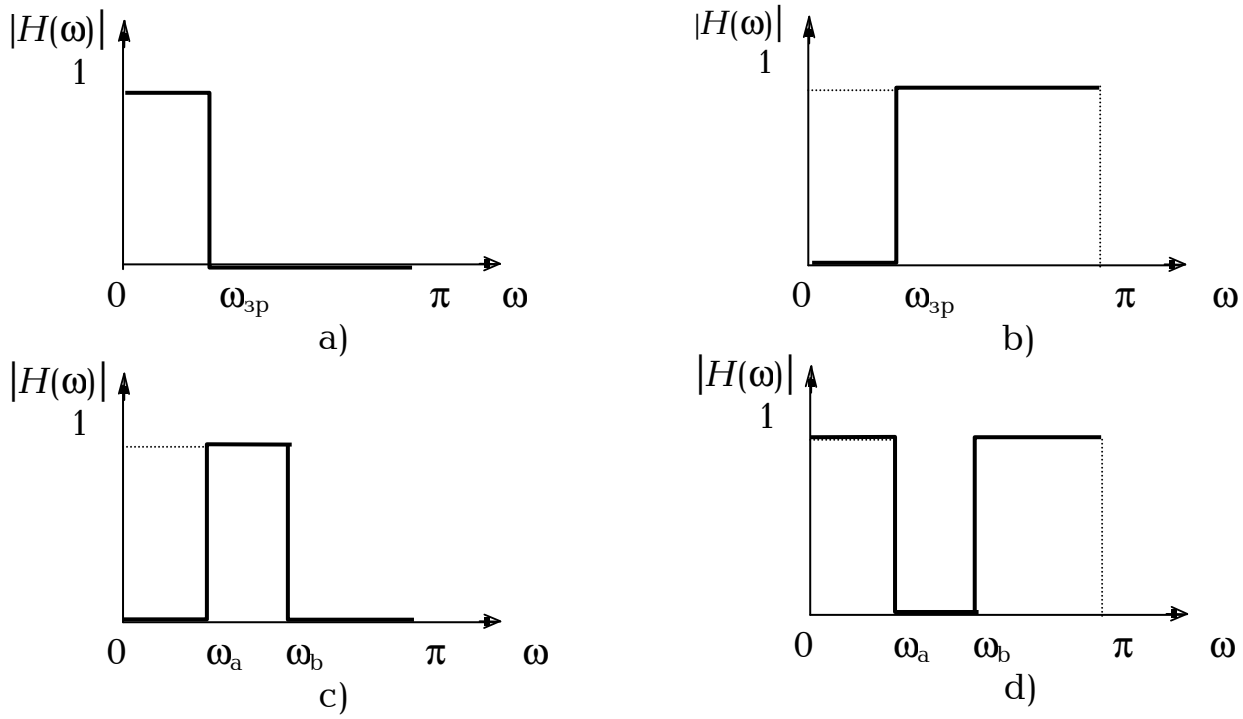


Fig. 1.1. Diagrams of the frequency characteristics of the low-pass (a), high-pass (b), bandpass (c) and rejector (d) filters.

In the frequency response of the bandpass filter, the frequencies ω_a, ω_b are called the lower and upper cutoff frequencies. Such a filter passes the frequencies in the range (ω_a, ω_b) and suppresses the frequencies in other ranges (Fig. 1.1, c). The rejecting filter, on the contrary, suppresses the frequencies in the range (ω_a, ω_b) (Fig. 12, d).

There are phase filter distinguished, which characteristic $|H(\omega)| = 1$ for all frequencies. A special case of a phase filter is the Hilbert filter, which performs the phase shift of the input signal at an angle $\pi/2$.

There are multi-band filters that have multiple passbands and stopbands. The comb filter is a kind of multi-band filter that has several lanes. These lanes are equidistant in a period of frequency, and therefore its AFC resembles a comb in shape.

In fact, it is not possible to achieve the ideal form of the frequency response, as in Fig. 1.1. In practice, the frequency response is determined by the initial data of the passbands $(0,\omega_п)$, stopbands $(\omega_н,\pi)$, the transition band $(\omega_п,\omega_н)$ of the pulsations δ_1 in the passband and pulsations δ_2 in the stopband (Fig. 1.2).

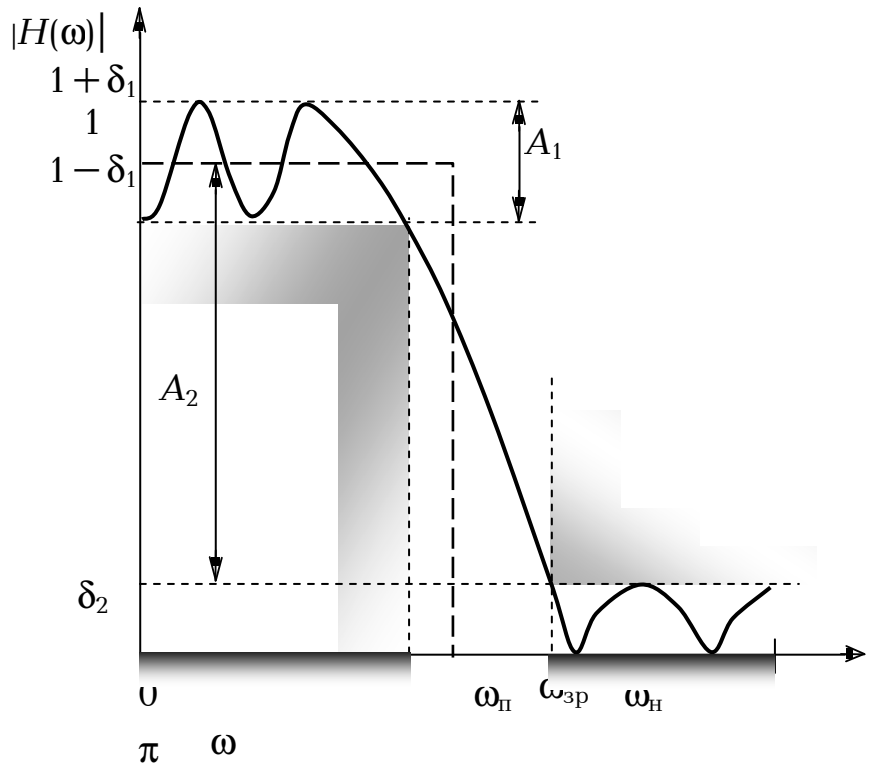


Fig. 1.2. AFC diagram of a real low pass filter

These ripple levels are usually given in relative units — in decibels, as the magnitude error A_1 in the passband and the minimum level A_2 of the filter suppression:

$$A_1 = 20 \log\left(\frac{1+\delta_1}{1-\delta_1}\right), \text{ db};$$
$$A_2 = 20 \log \delta_2, \text{ db}. \quad (1.3)$$

1.1.2 Z-transform and filter characteristics

Z-transfer $X(z)$ of the sequence $x(n)$ is given as:

$$X(z) = \sum_{n=-\infty}^{\infty} x(n) z^{-n}, \quad (1.4)$$

where z is a complex variable. If z is represented in the polar coordinates $z = re^{j\omega}$, and $r = 1$, then $|z| = 1$, then the z-transformation looks like a discrete Fourier transform. Consequently, z-transformation is a generalized spectral representation of discrete signals, just as the Laplace transform is a spectral representation of continuous, that is, analog signals. In the z-space we can show the spectral properties of linear systems that are invariant under the signal shifting.

The z-transform is linear one:

if $f(n) = \alpha y_1(n) + \beta y_2(n)$,

then $F(z) = \alpha Y_1(z) + \beta Y_2(z)$.

The signal shifting in time $f(n–m)$ is mapped to the multiplication in z-space: $z^{-m}F(z)$. Therefore, the function z^{-m} performs the signal delay to m cycles of the quantization.

The convolution operation $h_1(n) * h_2(n)$ is mapped to the multiplication: $H_1(n)H_2(n)$ in the z-space.

The impulse response of any linear system is derived as [3]:

$$h(n) = \frac{y(n)}{x(n)} \ . \quad (1.5)$$

And it is reflected in the z-space as a transfer function:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{r=0}^{M} b_r z^{-r}}{1 + \sum_{k=1}^{N} a_k z^{-k}}, \tag{1.6}$$

where a_k, b_r are real numbers.

The transfer function of the FIR filter is more simpler that one, which is given as (1.6):

$$H(z) = \sum_{r=0}^{M} b_r z^{-r}. \tag{1.7}$$

If M is an even number, that is, $N = 2M$, then (1.7) can be written as

$$H(z) = \sum_{r=-N}^{N} b_{r+N} z^{-r-N} = z^{-N} \sum_{r=-N}^{N} h_r z^{-r}. \tag{1.8}$$

Let the coefficients h_r have a symmetry: $h_r = h_{-r}$. Then, the frequency response, according to (1.8), is calculated by the formula

$$H(\omega) = e^{-j\omega N} \sum_{r=-N}^{N} h_r e^{-j\omega r} = e^{-j\omega N} \left(h_0 + 2 \sum_{r=1}^{N} h_r \cos(\omega r) \right), \tag{1.9}$$

where the factor $e^{-j\omega N}$ means only a delay of N cycles and can be eliminated from the analysis.

If M is an odd number, then in (1.9) there will be no single term h_0. If the coefficients of the impulse response h_r are real numbers, then $H(\omega)$ is a real function. The PFC of this filter is equal to

$$\varphi(\omega) = \begin{cases} -\omega N & \text{by } H(\omega) \geq 0, \\ \pi - \omega N & \text{by } H(\omega) < 0. \end{cases} \tag{1.10}$$

So, under the condition of the symmetry of the impulse response, PFC is linear, and every π radian changes of the angle $\varphi(\omega)$ the value of the AFC changes the sign. Even when the impulse response is antisymmetric, ie when $h_0 = 0$, $h_r = -h_{-r}$, the PFC is linear. According to (1.2), the group delay of such a FIR filter is equal to $T\text{d}(\omega) = M/2$, that is, it corresponds to the position of the middle member of the impulse response.

Properties of impulse response symmetry are usually used to reduce computational volumes. According to (1.5), the FIR filter is calculated as

$$y(n) = \sum_{r=0}^{M} b_r x(n-r). \qquad (1.11)$$

When taking into account the symmetry of the coefficients of b_r, the expression (1.11) can be rewritten as

$$y(n) = \sum_{r=0}^{M/2-1} b_r [x(n-r) + x(n-M+1+r)]. \qquad (1.12)$$

Comparing (1.11) and (1.12), it is seen that the number of multiplications in the case of the case is reduced by a half.

1.1.3 FIR filter synthesis

Till now, the FIR filter synthesis means the searching for its coefficients b_i or h_i, which are rounded for the filter program implementation in the integer arithmetics. FIR filters with different characteristics differ in the values and the number of their coefficients b_i. There are several methods of synthesizing a set of these coefficients. The most versatile and effective is the Parx and McLellan method, based on the Remez procedure of the polynomial optimasisation.

The initial data for the synthesis of the coefficients b_i or h_i (1.8) by this method is their number M, the pass band $(0,\omega_п)$ and stop band $(\omega_н,\pi)$, for example, for LPF, the amplitude error in the pass band A_1, the minimum level A_2 of suppression (1.3), or the degree of importance of the errors δ_1, δ_2 minimizing (Fig. 1.2).

According to the method, the AFC (1.9) is represented as a polynomial

$$H(\omega) = \sum_{r=0}^{N} a_i \cos^i(\omega). \qquad (1.13)$$

The method consists in selection a series of $N + 1$ frequencies ω_k , formulating and solving the system of equations (1.13). The results of the solutions of these equations are the coefficients a_i, which give the errors $\delta_1 = 0$ or $\delta_2 = 0$ at

frequencies ω_k. Then, the frequency response is calculated for other frequencies between ω_k and ω_{k+1}. If the error of such a polynomial approximation does not predominate A_1 and A_2, then the desired coefficients b_i are calculated from the a_i. If the coefficients do not fit, then other frequencies ω_k are selected and the process of optimization is repeated from the solution of the system of equations. The optimization process can be aimed at minimizing the criterion $w_1 \max(\delta_1)+w_2 \max(\delta_2)$ as well.

The preliminary value of the filter length M can be estimated from the empirical formula [4]:

$$M \approx \frac{-10 \lg(\delta_1 \cdot \delta_2)}{2{,}324\Delta\omega} - 13, \qquad (1.14)$$

where $\Delta\omega = |\omega_н - \omega_п|$ is the transitional band width. If, after calculating the coefficients of the filter, no decent levels A_1, A_2, are reached, the number M is increased, or the distance between $\omega_п$ та $\omega_н$ is increased. Since the behavior of the frequency response in the transition band is not controlled by the method, the excessive values $|\omega_н - \omega_п|$ can force the unwanted distortion of the frequency response.

The Parx and McLellan method is implemented in many CAD tools, such as Scilab or Matlab.

The overwhelming majority of FIR filters are calculated in the computers in the arithmetic of integers. When programming such a filter, a set of coefficients is synthesized that satisfy the given requirements and are represented by a floating point. Then the number of quantization bits of the coefficients n_c, input data n_x and results n_y are chosen.

Typically, $n_x, n_y \geq \log_2 10 \cdot D/20$,, where D is the dynamic range of the signal, dB. That is, for every 6 decibels of the dynamic range there is at least one bit of data. The coefficients are scaled and rounded, so that the integer coefficients are equal to

$$b'_i =]2^{nx} \cdot b_i + 0.5[\ . \qquad (1.15)$$

The results of the filter are calculated by formula (1.11) or (1.12) with a choice of such accumulator bit width, so that there is no overflow. And the product bit is equal to $n_p = n_c + n_x$, and the adder's bit width must be not less than

$$n_S = \log_2 S + n_C + n_x,$$

where S is the theoretically possible maximum result of the formula (1.11). It is easy to prove that S is equal to the sum of modules of all coefficients of a filter, i.e.

$$n_s = \log_2 \left(\sum_{i=0}^{M} |b_i| \right) + n_c + n_x. \qquad (1.16)$$

Since n_s can be quite large and the probability of achieving the result (1.11) of the maximal value is small, in the practice, the n_s value is chosen to be somewhat smaller, and the addition in (1.11) is performed by the algorithm of accumulation with saturation. Under this algorithm, if there is an overflow of the sum, then the result is substituted by the maximum number with the bit width n_s with the corresponding sign. The result of the filter $y(n)$ is taken as the most significant bits of the sum (1.11) with the truncation of the lower digits.

The rounding of the coefficients and the truncation of the result irreversibly distorts the response of the filter. Therefore, it is necessary to calculate the frequency response and the PFC, and compare them with the initial characteristics. If the characteristics of the filter do not satisfy the specified requirements, then it is necessary to increase the values of n_c and n_x, or M, and repeat the cycle of the coefficient synthesis. Sometimes it is possible to improve the frequency response, thanks to the correction of the lower digits of the coefficients b'_i.

1.1.4 Filter algorithm representation

By Tuering, the algorithm is the definition of a computational process on a particular computational model, which is described by means of mathematical concepts. The kinds of the algorithms differ by the type of computational models that have a variety of representations. An important requirement for such a model

is the convenience of the perception of the algorithm given on this model and the efficiency of the implementation of this algorithm [8].

Traditionally, the DSP algorithm is defined using a formula such as (1.11). It is assumed, that there is a source of signal samples $x(n)$, a receiver of the result signal $y(n)$, and someone or something that calculates the actual formula. It is necessary to recalculate this formula at each step, which has the number n and to follow the iterative increase of n. Also, some intermediate results $y(n-k)$, $x(n-r)$ have to be rewritten. Consequently, the statement of the algorithm by the formula declares that it is necessary to calculate, but does not show how and in what order to do so.

The algorithm in the form of a formula is implemented in the MathCAD system [9]. In this case, it is necessary to limit the number of processed input and output signal samples in advance, by a specific natural number. Since the formula only declares the rules of calculation, the efficiency of its implementation in the computer is low.

DSP algorithms, as a rule, foresee their implementation in the reactive computing system, that is, in such an computing system, for which the unacceptable loss of data due to the inability to process in time in the case of the limited operating speed. The data comes at the input of the DSP device with a constant period of T_S, which is inversely proportional to the sampling frequency $f_S = 1/T_S$. Therefore, if the device does not have time to execute the cycle of the algorithm for a given sample $x(n)$ during the cycle T_S, then the following sample $x(n + 1)$ will not be processed, because it will be lost.

Such algorithms are advisable to represent in the form of the data flow graph (DFG) model. A graph model is a computational model that processes the data flows. In this case, the *data flow* is the name of two categories: both the actual data set — the signal — and the means of the data transmission between the elements of the model.

The graph of the data flows consists of nodes-actors, which process data by some elementary operations and data streams-edges in the form of arcs connecting the neighboring nodes. Each node-actor reads the data from its input streams according to the certain rules, processes them and places the results in its output streams. The execution of the algorithm is a series of the firings of the nodes-actors, for which there are ready data in the input flows. Such operations can be executed in parallel at all nodes. A data flow graph is interpreted as a parallel program, in which the subroutines represent the nodes-actors, and the memory cells or buffer arrays are the data flows.

Often, the DSP algorithm is represented as a signal graph, sometimes is as a synchronous data flow graph (SDF) (see Fig. 1.3). SDF differs from DFG model, because all its data flows are synchronous ones.

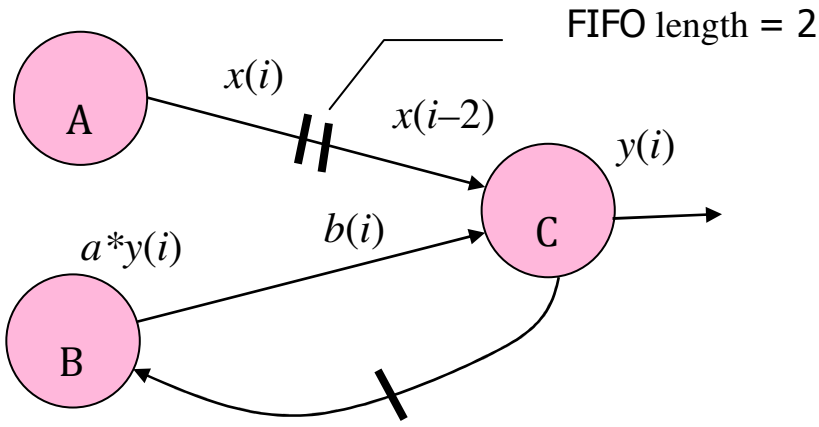


Fig. 1.3. Example of SDF

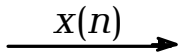
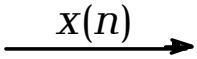
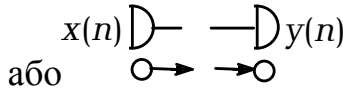
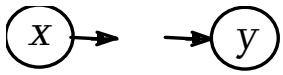
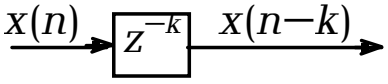
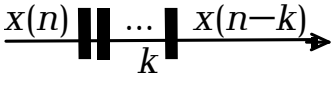
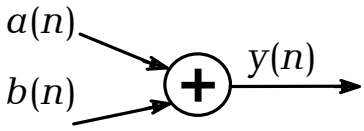
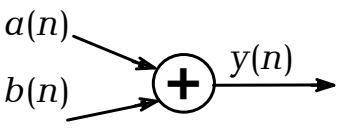
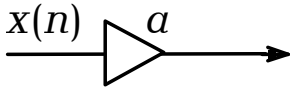
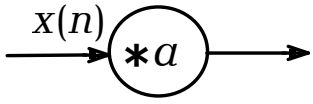
Two data flows in the graph model are synchronous if there is a mutual correspondence between all the data in one and the other flow. For example, data in a synchronous flow can be renumbered, and therefore the flows $y(n)$ and $x(n)$ are synchronous, since there is a correspondence between the n-th markers in them.

Moreover, n is considered as the number of the sampling cycle or clock cycle, or iteration of the algorithm.

Consequently, in the vast majority of DSP algorithms, the signal flows are synchronous. Therefore, such algorithms can be represented by SDF. If in a resultant flow the presence of data is conditionally dependent on the input flow, then such flows may be non-synchronous. These are, for example, flows in the compressor of signals, which replaces the chains of the null samples with the length codes of these chains [10].

The signal graph and the homogeneous SDF are equivalent models. In Table 1.1 the graphical symbols of the elements of the siginal graph, SDF and their correspondence to parts of the DSP algorithm are shown [11].

Table 1.1. Designations of the elements of the signal graph and SDF

Algorithm element	Signal graph	Uniform SDF
Signal $x(n)$		
Input and output ports, source and destination of signals $x(n)$,$y(n)$		
Delay to k cycles		
Signal addition, adder node $y(n) = a(n)+b(n)$		
Multiplication the signal to a constant $y(n) = a \cdot x(n)$, multiplier node		

Consider the signal graph of the algorithm, represented by formula (1.12) when the filter length M is odd. It is illustrated by Fig.1.4. When M is even, then such a filter has the multiplication number less to one.

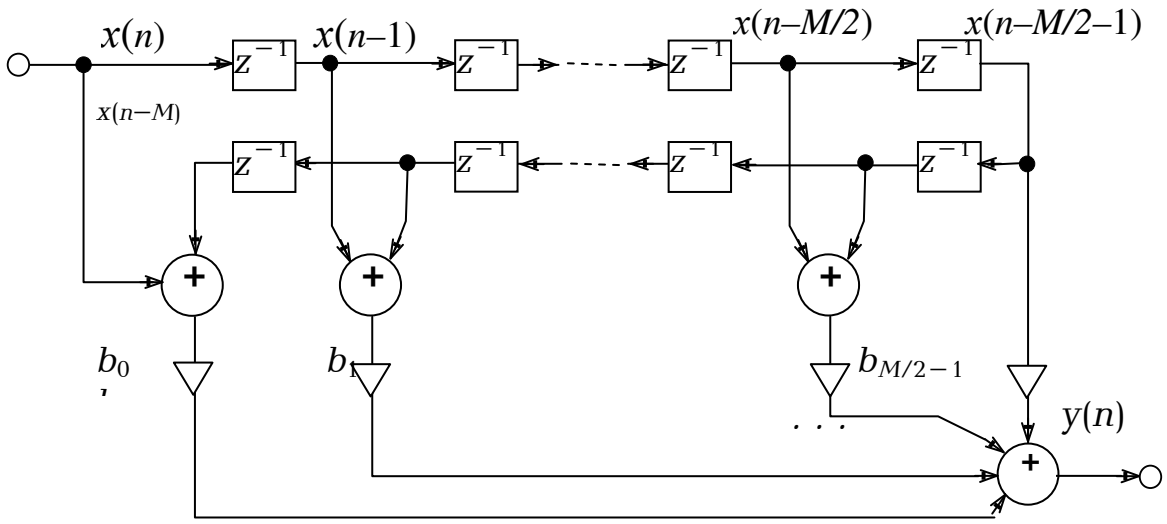


Fig. 1.4 Signal graph of the symmetrical FIR filter

Consequently, the signal graph and SDF are the convenient and obvious form for the DSP algorithm. Next, the algorithms will often be represented using such graphs. It is also possible to describe the SDF graph by VHDL and then, automatically synthesize the digital netlist for the FPGA configuration, which will be discussed further.

So, FIR filters are usually implemented in the programs. There is a wide theoretical material about the filter coefficient synthesis, representing algorithms by graphs and their optimization efforts. But there are a few works devoted to the filter structure synthesis. The expanding the FPGA technology needs to look at the FIR structure synthesis more closely.

1.2 FPGA as the computing environment for FIR filter implementation

1.2.1 FPGA architecture

Below, the properties of the Xilinx FPGAs are considered, because this company is valued as the larger FPGA supplier. But the proposed reasons are true for FPGAs of other companies as well.

In Xilinx FPGAs, the basic building blocks are Configurable Logic Blocks (CLBs). In Spartan-6 devices, the CLBs are made up of two logic slices which are independently connected to the general routing on the FPGA and to a carry chain structure [12]. There are two types of logic slices in Spartan-6, SLICEL and SLICEM. SLICEL can be seen as the basic logic slice type, and contains four 6-input look-up-tables (LUTs), together with four D-type flip-flops (DFFs) and multiplexers for routing purposes. The LUTs can implement any 6-input logic function. SLICEM slices contain shift register functionality and provide the option of using the LUTs as distributed user RAM, as well as the basic resources described for SLICEL slices. When used as distributed RAM, LUTs are configured as memories for user data storage.

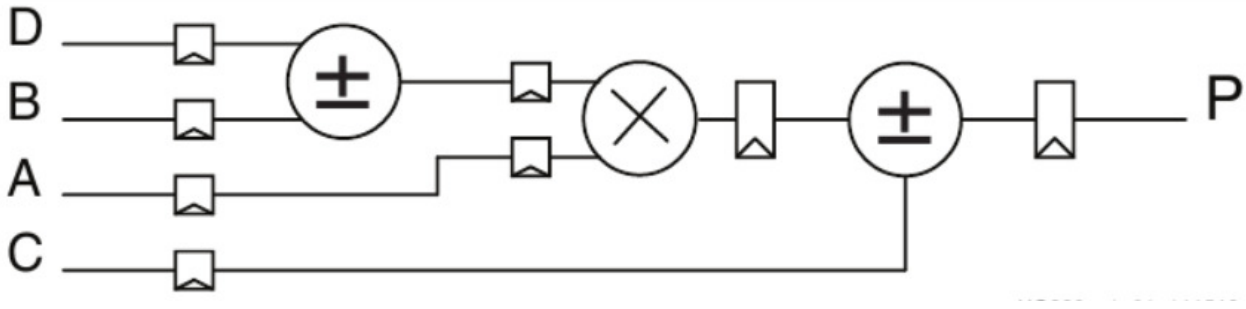
Other resources on the FPGA include Digital Clock Managers (DCM), Phase-Locked Loops (PLL), Block RAMs, DSP blocks, I/O blocks (IOBs) and buffers for connecting package pins. The FPGA resources are connected together by a configurable routing matrix. A common way of describing FPGAs is as configurable logic “islands” connected together by a “sea” of configurable routing paths.

When synthesising an FPGA design, the circuit function defined by the designer is mapped to these resources by synthesis tools. This mapping makes up the configuration of the device, and is stored in the SRAM-based configuration memory.

The configuration memory defines the function and operation of all the described resources as well as the routing and connections on the FPGA, and can be seen as an underlying device definition layer.

SRAM-based FPGAs are programmed using a binary bit-stream, usually stored offchip. For space applications, this off-chip configuration storage is usually in the form of EEPROM or Flash. Since the SRAM-based configuration memory is volatile, the bit stream has to be reprogrammed onto the FPGA on startup and power-cycling. The programming logic is responsible for writing the configuration memory via one of the configuration interfaces.

Xilinx Spartan-6 FPGAs contain dedicated DSP circuitry, in the form of DSP48A slices. Fig. 2.1 shows a simplified view of a DSP48A slice, featuring a 18x18 multiplier, internal pipelining registers and an arithmetic unit. DSP blocks are hard ASIC blocks embedded in the FPGAs array of programmable logic, and are much more area efficient compared to soft logic implementations of the same functionality [13]. As such, DSP blocks are not defined by an underlying configuration layer. The DSP48A is well suited for common DSP operations such as multiply-accumulate.



.

Fig.1.5. Simplified view of a DSP48A slice

The configuration vectors can be synthesised as constants or as signals originating from other parts of the system. DSP slices are arranged on the FPGA so that they can be cascaded through the use of fixed carry and shift lines to create wider operators than what would fit into a single DSP slice.

Block RAM, or BRAM, in Spartan-6 are made up of 36 kB SRAM memory blocks. These blocks can be cascaded and divided into a number of different configurations. For example, a single 36kB block can be used as a 36kx1 RAM, or as two functionally separate 18kx1 RAMs. It is also possible to create wider or larger RAM blocks by cascading BRAMs together.

So, when choosing an FIR filter algorithm, one should keep in mind the features of an FPGA structure that has CLB resources, multipliers, adders, multiplication blocks, but does not have divisions. For its rapid execution, the FIR filter should be implemented as a parallel structure that allows the pipelinined operations, because this mode is effectively supported in FPGA.

1.2.2 Implementation of FIR filters in FPGA

The FPGA architecture is adapted to the FIR filter algorithms to provide the excellent throughput values. For this purpose, the adder cascade network is used. Fig. 1.6 illustrates such a network for implementing the 8-staged filter [14].

The respective SDF of the *k*-staged adder cascade network is shown in Fig. 1.7. Here x_i, y_i are the input and output data, the circle, triangle and bar represent addition, multiplication to the coefficient and delay to a single clock cycle, respectively. This graph is mapped to the respective structure by the one to one mapping providing the high pipelined computations with the maximized clock frequency

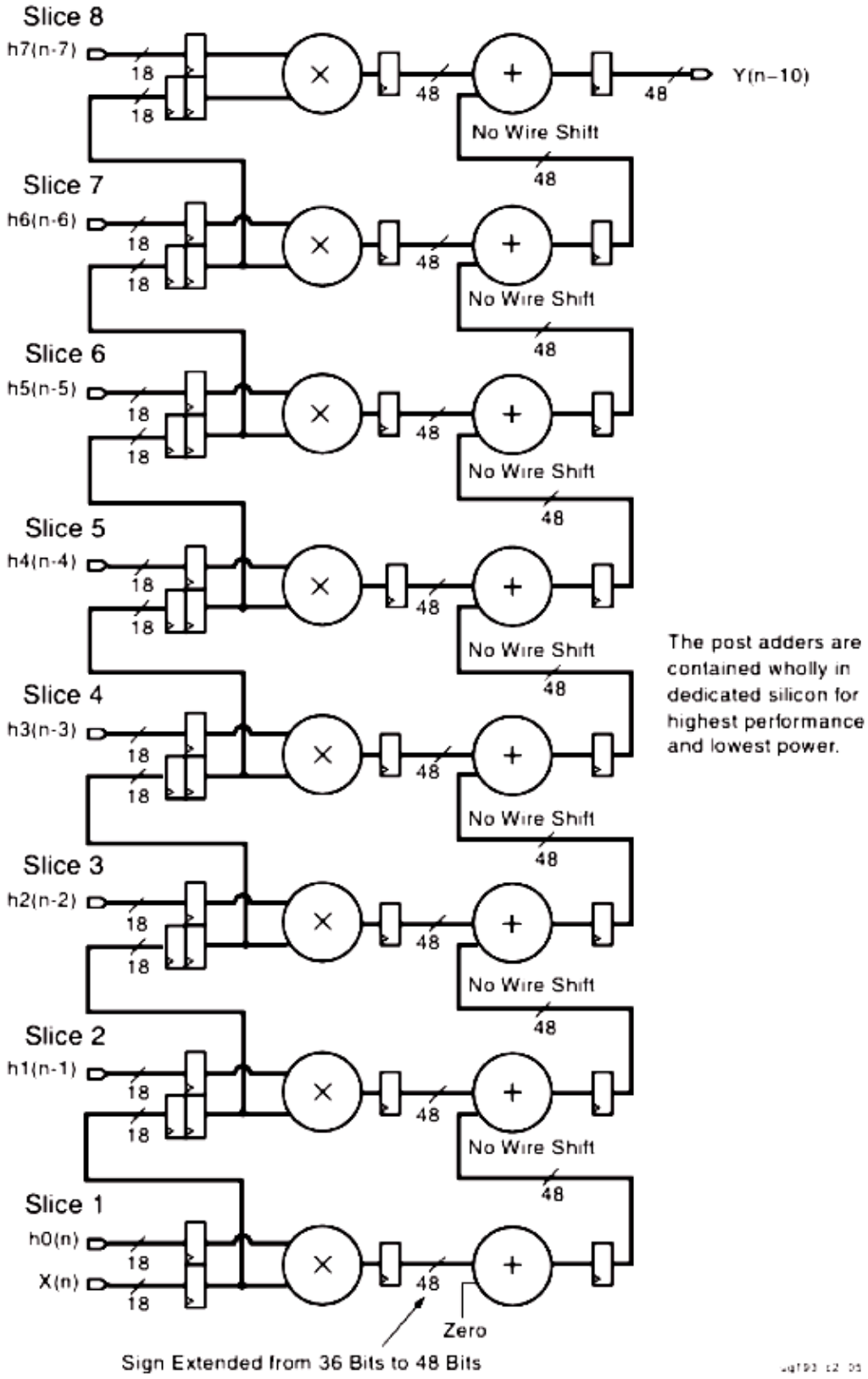


Fig. 1.6. Adder cascade network for the FIR filter implementation

.

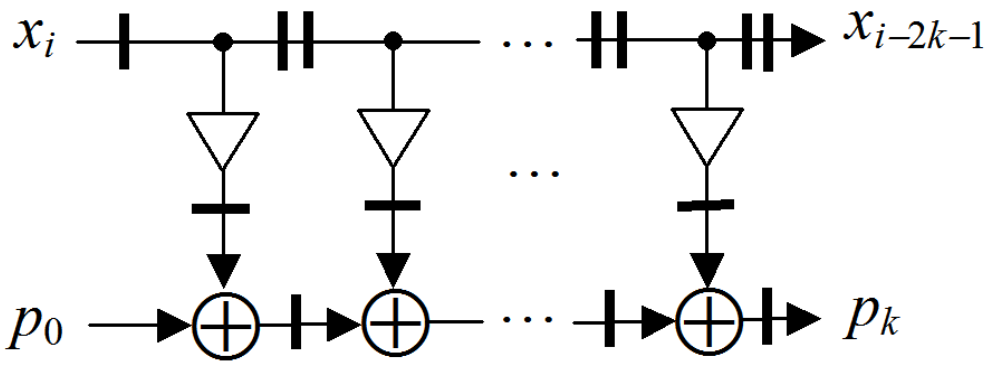


Fig.1.7. FIR filter signal flow graph as a systolic structure

It is interesting to know, that the adder cascade network has the well-known systolic structure [15].

Using the cascade paths in the the adder cascade network to implement the FIR filter significantly improves the power consumption and speed. But the maximum number of cascades in a path is limited by the total number of DSP48 slices in one column in the chip. The height of the DSP column can differ between the Virtex-4 and Virtex-8, and Spartan-6 devices and should be considered while both designing the system and porting designs between the devices. This heigh is varied in range from 4 to 240, but the total DSP slice count is limited by CAD tools and there are no more 60 DSP slices per column without adding the special interconnections [16].

Spanning columns is possible by taking P bus output from the top of one DSP column and adding fabric pipeline registers to route this bus to the C port of the bottom DSP48E slice of the adjacent DSP column. Alignment of input operands is also necessary to span multiple DSP columns.

Additionally, for the FIR filter implementation, Xilinx recommends that:

— Small multiplies (e.g., 9x9 multiply) and small bit width adders and counters should be implemented using the fabric LUTs and carry chain. If the

design has a large number of small add operations, the designer should take advantage of the SIMD mode and implement the operation in the DSP48 slice.

— Always sign extend the input operands when implementing smaller bit width functions. For lower fabric power, push operands into MSBs and ground (GND) LSBs.

— While cascading different DSP48E slices, the pipe stages of the different signal paths should be matched.

— SRL16s in the CLB and block RAM should be used to store filter coefficients or act as a register file or memory elements in conjunction with the DSP48E slice. The bit pitch of the input bits (4 bits per interconnect) is designed to pitch match the CLB and block RAM.

— A pipeline register should be used at the output of an SRL16 before connecting it to the input of the DSP48E slice. This ensures the best performance of input operands feeding the DSP48E slice.

As a result, the FIR filters in FPGA can be realized as the high-speed pipelined networks with the clock frequency up to several hundreds of megahertz. But, for example, the FIR filter which is generated by the Xilinx Coregen tool for the Spartan-6 devices have the length, which is limited by the numbers from 8 to 48 [13]. And the more this number the more expensive the chip is. Besides, when FPGA is used only for the filtering, then the configured hardware like look-up tables (LUTs), registers is underloaded and is used ine□ectively.

So, it is preferably to searching for the filter structure solutions, which can do without the DSP48 units.

1.3 Multiplier-free FIR filter implementation

1.3.1 Review of the multiplier minimizations

As it was shown above, the FIR filters have the large number of multiplication operations. Therefore, the complexity of the filter is mainly due to thelarge number of multiplications required [17].

A variety of approaches have been proposed to speed-up the multiplication operation in a filter structure [18-20]. These either completely eliminate the existing multiplier unit or reduce the complexity of the multiplication operation. Two frequently used approaches are: ROM based implementations [21] and distributed arithmetic (DA) based implementations [22-25].

In [26] an approach is proposed, when the multiplcand is divided into slices, each of them is multiplied by a coefficient in a set of LUTs, where the multiple coefficients are stored.

The authors in [27] present a new kind of FPGA implementation algorithm for DA FIR filter which is based on Remainder theorem. This minimizes the required logic resources. Another approach uses the divided-LUT method to reduce the required memory units [28].

Bit-serial arithmetic has been extensively used in filtering applications. These have the advantage that communication demands are independent of the word length. As a result the low-capacity FPGAs can efficiently implement the FIR filters [29]. Special purpose bit-serial implementations include power-of-two sum or difference approaches. This allows multiplication to be replaced with faster shift and addition operations [30-32].

Linear systolic filter structures like that illustrated by Fig. 1.7, are also categorized as bit-serial architectures [33]. The authors in [34] and [35] take a systolic approach for designing high-throughput filter structures by using multipliers based on direct ROM and DA approaches. Similarly, the work reported

in [36] uses systolic structures but focuses on replacing the original adder unit by using a parallel prefix adder (PPA) with minimum-depth algorithm.

The canonic signed digit (CSD) representation has been used to reduce the complexity of multiplication operation thereby resulting in efficient filter structures [37]. Constant coefficient multipliers have been reported in [38–40].

Poly-phase decomposition is a technique that has been used to design high-speed and low-power parallel FIR filters [41-43]. A modification of poly-phase decomposition is the fast FIR algorithm (FFA). Filters based on FFA are area-efficient utilizing fewer multiplier units [43-45].

All the above-mentioned approaches use the technology-independent optimizations to enhance the performance of the filtering structure. They can be adapted to the FPGA architecture as well.

1.3.2 Constant multiplication in FPGA

FPGAs of the first generation have not hardware multipliers. Therefore, the FIR filters in them are implemented using the constant coefficient multipliers (CCM). Below is a review of the known methods of CCM design which provide different levels of optimizations.

A generic multiplication of two unsigned binary B-bit integer numbers $x = (x_{B-1} ... x_1 x_0)$ and $y = (y_{B-1} ... y_1 y_0)$, with x_i , $y_i \in \{0,1\}$ can be written as

$$x \cdot y = x \cdot \left(\sum_{i=1}^{B-1} 2^i y_i \right) = \sum_{i=1}^{B-1} 2^i x \cdot y_i, \tag{1.17}$$

where $2^i x \cdot y_i$ ist the partial product.

In a generic multiplier, the partial products $x \cdot y_i$ can be obtained by a bitwise AND-operation. The final product is then obtained by adding the bit-shifted partial products. Now, if y is a constant bit vector, all bits y_i , which are zero, lead to zero partial products and the corresponding adders can be removed.

Thus, the number of required adders for the constant multiplication using this representation is equal to the number of non-zero elements in the binary representation of y minus one [45].

To illustrate this, consider a multiplier with $y = 93$. Its binary representation $y = 1011101_2$ has five ones and requires four adders in the corresponding add-and-shift-realization $93x = (2^6 + 2^4 + 2^3 + 2^2 + 1)x$ as illustrated in Fig. 1.8,a. Note that bit shifts to the left are indicated by left arrows in Fig. 1.8.

The total number of operations can be further reduced by allowing subtract operations in the add-and-shift network. As the subtract operation is nearly equivalent in hardware cost, both are referred to as adders in the following. The adder reduction can be realized by converting the constant to the signed digit (SD) number system [47–48], in which each digit is represented by one of the values of $\{-1, 0, 1\}$.

In the example, the coeffient can be represented by

$$93 = 10\bar{1}00\bar{1}01_{SD},$$

where digit $\bar{1}$ corresponds to -1, i. e.,

$$93 = 2^7 - 2^5 - 2^2 + 2^0.$$

Now, the corresponding circuit uses one adder less compared to the binary representation as illustrated in Fig 1.8,b.

The signed digit number system is not unique as there may exist several alternative representations for the same number. For example, 93 could also be represented by

$$93 = 1\bar{1}0\bar{1}00\bar{1}\bar{1}_{SD},$$

which has the same number of non-zero digits as the binary representation. A unique representation with minimal number of non-zero digits is given by the canonic signed digit (CSD) representation. A binary number can be converted to the CSD with the algorithm, described in [48,49]. The algorithm is based on the

searching for the bit sequence in the form “011...11” and replace it with the sequence “10…0$\bar{1}$” of the same length.

The CSD representation is unique and guarantees a minimal number of non-zeros. However, there may still be several SD representations with a minimal number of non-zeros. The representations with a minimal number of non-zeros are called minimum signed digit (MSD). The SD representation after the first iteration for 93 is an example of an MSD number which is not a CSD number.

Starting from the CSD representation, valid MSD representations can be constructed by replacing the bit patterns “10$\bar{1}$” with “011” or “$\bar{1}$01” with “0$\bar{1}\bar{1}$”. Doing this for all combinations results in a set of MSD numbers. For the example of the integer 93, four di□erent MSD representations can be constructed:

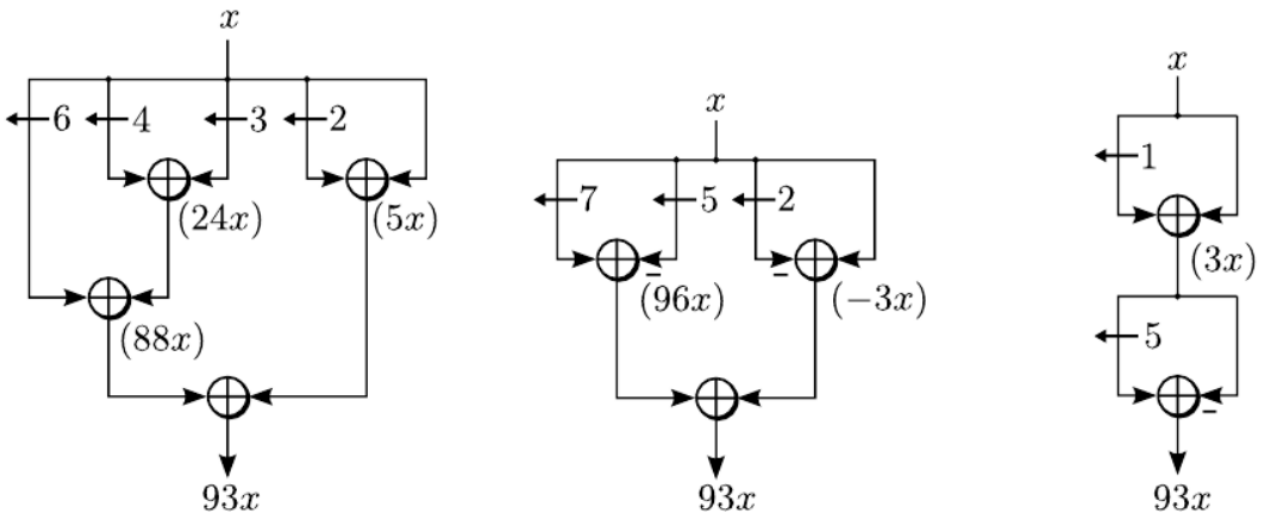
$$93 = 10\bar{1}00\bar{1}01_{CSD} = 01100\bar{1}01_{MSD} = 10\bar{1}000\bar{1}\bar{1}_{MSD} = 0110000\bar{1}\bar{1}_{MSD}.$$

Although the arithmetic complexity of the constant multiplier can be reduced by using an MSD representation of the constant, it is not guaranteed to be minimal. Consider again the example number 93 which can be factored into

$$93 = 3 \cdot 31.$$

With $3 = 11_{MSD}$ and $31 = 10000\bar{1}_{MSD}$, the cascade of these two constant multipliers reduces the required adders to only two as shown in Fig.1.8,c.

This concept is known as sub-expression sharing and the corresponding optimization method is called common subexpression elimination (CSE) [50,51].



Fig, 1.8. Different networks of CCM to a constant 93

It is worth to mention, that the modern FPGAs have the 6-input LUTs with the additional outputs, which provide the 3-input adders-subtractors implementation. This fact shows that the FPGA architecture is adapted to the hardware implementation of the CCM, based on CSD, MSD, or CSE constant representation.

The CCM blocks have the following limitations. Firstly, CCM can compete with the DSP48 units, when the adder number in it is less than 20 adders. This fact is proven in the next section. Taking into account that the MSD representation needs less than $n/2$ inputs of the result adder, we can consider, that the CCM is effective to use, when the bit width $n < 42$. This is enough for the most of FIR filter projects.

Secondly, the many input adder has the large critical path, which for some number of inputs is greater than the DSP48 block delay, which substantially infers to the filter throughput. But this factor depends on the CSD decomposition in each particular situation, and therefore, it needs investigations. The critical path is

substantially decreased by the pipelining the CCM. But the critical path limit depends on the placing and routing this CCM in FPGA and in the worst case can in many times supercede the delay of DSP48. Anyway, when the constant bit width is rather small, say $n < 12$, then it is preferably to design the CCM.

1.3.3 Multiple constant multiplication in FIR filters

The FIR filter structure, which is represented by the signal graph in Fig. 1.4, or SDF in Fig.1.7, is named as the filter in direct form. The FIR filter in the transposed form is considered as well [1]. Its signal graph is shown in Fig. 1.9.

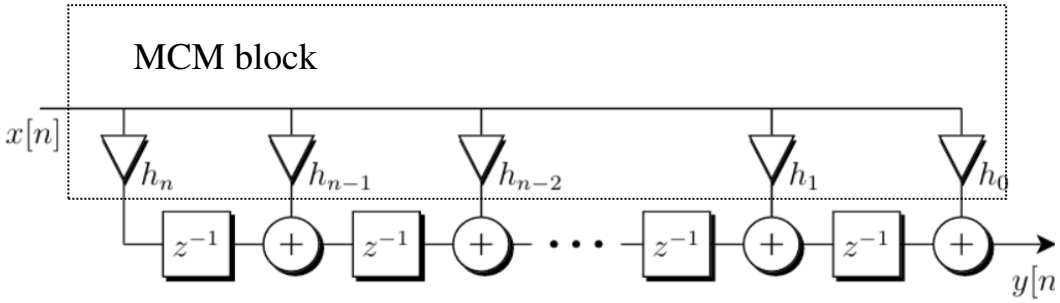


Fig.1.9. Transposed form of the FIR filter

In the transposed form, a single input has to be first multiplied by several constants which is called a multiple constant multiplier block or MCM block (dashed box in Fig.1.9). The remaining filter consists of adders and registers. An *N*-tap FIR filter requires the multiplication with *N* constants.

In the MCM block, the intermediate adder results can be shared between different cofficients such that the overall complexity is reduced. Take, formexample, the multiplication with the constants 19 and 43. Their CSD based multipliers as introduced above are shown in Fig. 1.10,a. One MSD representation of the example constants is $19 = 10011_{\text{MSD}}$ and $43 = 101011_{\text{MSD}}$.

In both representations, the bit pattern 11_{MSD} is found, which can be shared to reduce one adder. The resulting adder circuit is shown in Fig. 1.10,b. However, as demonstrated in Fig. 1.10,c, another adder configuration can be found, which does the same multiplications with one adder less. It can not be obtained by the CSE approach as no bit pattern of the factor $5 = 101_{MSD}$ can be found in both MSD representations of the constants.

So, the MCM block introduction and its optimization provides the minimum hardware volume for the FIR filter hardware implementation. But the FPGA architecture does not support the inverse FIR filter implementation.

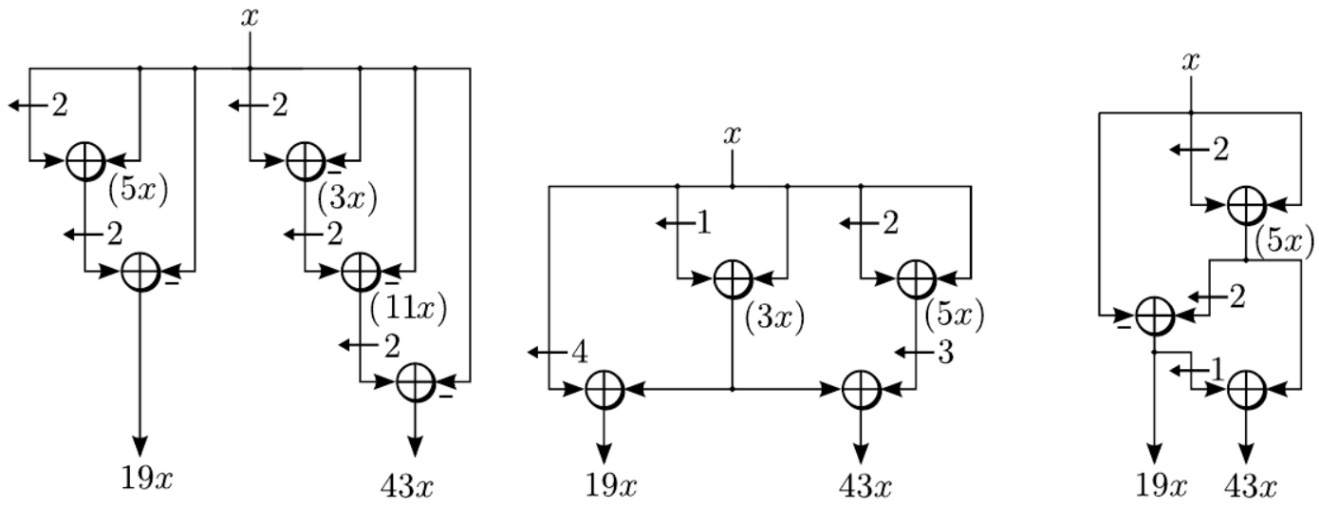


Fig. 1.10 The MCM block implementation: before (a) and after (b),(c) optimization

Firstly, the cascade architecture illustrated by Fig. 1.6, is intentionally adapted to the direct form of the filter, secondly, even if the MCM

The constant coefficient multipliers can provide the multiplier-free FIR filter structures, especially, when the coefficients have the bitwidths which is less than 12.

In the next section, the theoretical basics of the new method of the FIR filter design is developed, which satisfy the mentioned above features.

1.4 Conclusions to the section

The FIR filter IP cores, proposed by the FPGA vendors, have the limitations of the filter order and utilize the configurable hardware, and therefore, they need modernizations.

Multiplier-free FIR filter structures, based on the constant coefficient multipliers, have the minimum hardware costs in the gate number, but do not utilize the multiplier blocks embedded in FPGA.

There is a hypothesis that the combination of the systolic FIR filter structures based on the multiplier blocks with ones based on CCMs can provide both high speed and minimized hardware volume in FPGAs.

The synchronous dataflow graphs are the convenient tool for the representation the FIR filter algorithm and mapping it into the pipelined structure.

2 METHOD OF INCREASING THE EFFICIENCY OF THE FIR DIGITAL FILTERS

2.1 FPGA use considerations for the FIR filter design

In the previous section, it was found out, that the direct form of the FIR filter structure is the best one to be implemented in FPGA, the FIR filter IP cores, which are proposed by the FPGA vendors, are based on the DSP block resources, have the limitations of the filter length, and do not utilize the rest of resources like LUTs, multiplexers, registers, the CCM can provide the multiplier-free FIR filter structures, when the coefficients have the small bitwidths. But it was also clarified, that these factors of the FIR filter structure selection depend on many things. Therefore, this question needs the further investigations.

2.1.1 FPGA project optimization critera

The FPGA resources, which are mentioned in the subsection 1.2, are valuable. Different projects for FPGA, which perform the same task, can be distinguished in different folume of these resources. Moreover, these projects can be of different throughput. To select properly the best project, the effective effectiveness criteria must be selected. Below, some considerations to these criteria selection are considered.

2.1.2 Hardware volume criterium

In advance, we consider, that the processing unit bit width is equal to n, and its hardware is proportional to n in some limitations, and by other equal conditions.

The adder is the main operational unit in FPGA project. Usually, one bit of the adder is implemented in a single LUT, not taking into account the proper carry propagation network. Besides, each LUT output can be stored to the respective register (trigger), as in is shown in Fig. 2.1,a. Thus, the n-bit adder, and the n-bit

register have the same complexity, or cost. Then, such register, and adder have the relative cost, which is equal to a 1.

Also it is important to consider that LUT has the mode SRL16, in which it operates as a shift register with the programmable length of 1 to 16 bits (Fig.2.1,b).

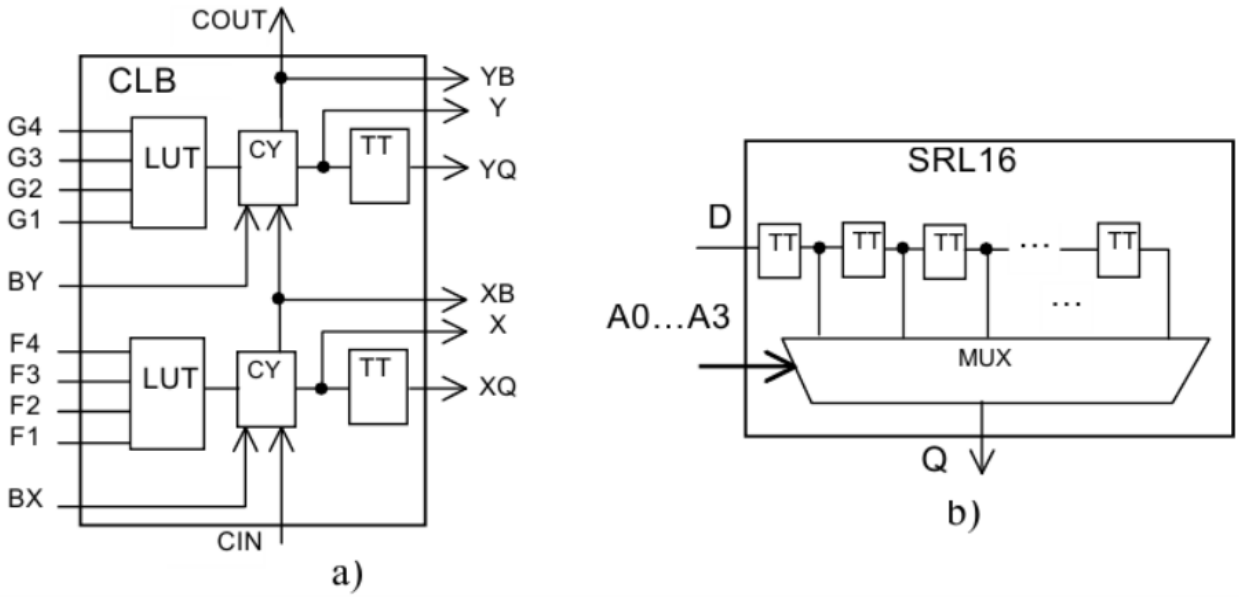


Fig. 2.1. Structure of the Xilinx FPGA elements: CLBS (a), SRL16 (b)

In the FPGA chip one DSP48 unit takes 60–300 CLB slices, averagely, 160 CLB slices. For reference, the hardwired 18x18 bit multiplier is implemented as an equivalent circuit of 208 CLB slices. Consider a DSP processor configured in FPGA with the hardware resources being used effectively. Then all multiplier resources should be loaded by the useful computations, and other computations are distributed among all adders and multiplexers implemented in FPGA. By this condition, one multiplier takes 160 CLB slices. These CLBs are enough to implement up to 20 adders and 20 registers of the same bit width. Thus, the complexity of the multiplier unit is estimated as the complexity of 20 adders. Similarly, the complexity of the Distributed RAM can be estimated.

Table 2.1 shows the complexity of the different elements of the same bit width configured in FPGA, which is expressed in the complexity of a single register.

Table 2.1.

Complexity of elements, configured in FPGA

Type	Complexity
Register	1
Adder	1
Adder-subtractor	1
2-input multiplexor	1
3,4 -input multiplexor	2
5,6-input multiplexor	3
7,8-input multiplexor	4
Registered delay to 2-16 registers (FIFO)	1−2
Multiplier unit	20
16 word RAM	1
1024 word RAM	20

Its analysis shows, that multiplying units should be minimized primarly. Since in the actual application specific processors the 2–5 input multiplexers frequently are used, then the complexity of the multiplexer, which takes to a single input, is equal approximately to 0.27. This means that it is necessary to mimimize not only the number of registers and adders, but also number of multiplexot inputs.

According to the arguments above, the following complexity criterion of the FPGA project is proposed:

$$Q_S = n_R + n_A + 20n_M + 0.27n_x, \qquad (2.1)$$

Where n_R is the register number, including the FIFO number, which are mapped into SRL16 primitive, excluding the registers in the DSP48 modules;

n_A is the adder number, due to the CLB construction, up to three input adder is implemented in a single CLB column, therefore, n_A considers 2- or 3-input adders;

n_M is the multiply unit number;

n_x is the number of the multiplexor inputs [38].

2.1.3 Performance criterion

The signal delay in the multiplier blocks is approximately equal to 4.5 ns for Spartan-6 FPGA. In the two-staged pipelined multiplier the minimum multiplication period is equal to 2–2.5 ns. The adder delay is derived from the carry signal propagation and therefore, it is proportional to the bit width. Since the adder is formed as a line of the locally coupled DLB slices, then its delay is stable, and for 16-bit adder is equal to 1.4–2.5 ns.

It has to taken into considerations, that the proportion of the delay in the logic elements is 35–85% of the clock period depending on the degree of the placing and routing optimization, and on the complexity of the structure.

In the practice, the multiplier delay is about twice te adder delay, taking into account the interconnection delays.

The multiplexer network has far less latency then the adder has. It is not depended on the word length, and is nearly independed on the input number., but depends on the quality of the wiring of the lines, which connect it to the neighboring elements. As a result, the connection of the additional multiplexor to the adder adds a delay of 0.4–1.6 ns depending on the multiplexor number (1 or 2) and routing quality.

Thus, the proposed performance criterion is:

$$Q_{\mathrm{T}} = n'_A + c_{TM}\, n'_M + c_{TX}\, n'_x, \qquad (2.2)$$

where c_{TM}, c_{TX} are the ratios of the multiplier and multiplexor delay to the adder delay, $c_{TM} = 2.2$, $c_{TX} = 0.5$;

n'_A is the adder number;

n'_M is the number multipliers;

n'_x is the number of multiplexers,

staying in the critical path, which connects the output of one register and the input of another one. Here, a single unit delay is estimated as the delay of the adder with the average delays in the communication lines.

Really, Q_T is equal to the minimum clock period, derived for the current placed and routed project, when the results are outputted in each clock cycle. It is hold on when the processing unit is implemented as a whole combinational network, which performs the elementary function, or if it is wholly pipelined network.

The real processing unit projects can calculate the algorithm for $L > 1$ clock cycles not in the pipelined mode. Thus, the expression (2.2) must be multiplied by the value of L:

$$Q_T = L\,(n'_A + c_{TM}\,n'_M + c_{TX}\,n'_{x.}). \qquad (2.3)$$

The integral criterium has to take into account both hardware volume and performance criteria. Then, it can be selected as:

$$Q = Q_S \cdot Q_T \qquad (2.4)$$

This criterium shows, how many adders are needed to calculate, say, one million of results per second. The better solution has the smaller value of Q, because it has smaller hardware volume and/or higher clock frequency, which is proportional to the processor performance.

2.2 Synchronous dataflow graph for the FIR filter description

2.2.1 Synchronous dataflow graph mapping to the structure

The FIR filter processing module belongs to the datapaths. The modern high-performance computers operate with high clock frequencies, thanks to the pipelined mode of data processing and transmission. There are various methods for the design and optimization of the pipelined datapaths. These methods are based on the structural synthesis of the datapath, describing it at the register transfer level and further conversion to the gate level. The basis of many methods is a representation of the algorithm as a synchronous dataflow graph (SDF) and its transformation [52]. Some definitions about SDF was given in the subsection 1.1.4.

SDF is isomorphic to the graph of the computer structure, which performs a predetermined algorithm. The nodes of such a graph correspond to the computing resources like adders, multipliers, processing units (PUs). The edges correspond to the communication lines, and the labels on them are mapped to the registers. Consequently, SDF is a directed graph $G = (V, E)$, representing the computer structure, where $v \in V$ represent some logic network with delay of d time units. The edge $e \in E$ corresponds to a link and is loaded by $w[e]$ labels, which is equal to the depth of the FIFO buffer.

The minimum duration of the clock cycle T_C is equal to the maximum delay of the signal from one register output to the input of another register, i.e., to the critical path through the adjacent nodes with delays d, for which $w[e] = 0$. It should be noted, that with such a one-to-one mapping of SDF, the duration of the algorithm cycle T_A coincides with the duration of a clock period, i.e., $T_A = T_C$, that in the other algorithm mapping is not respected.

2.2.2. SDF optimizations

By the pipelined structure synthesis, a set of optimizations is used. Such SDF optimization techniques as retiming, folding, unfolding and pipelining, are widely used in microelectronics, and design of digital signal processing (DSP) devices [53].

The retiming is such a exchange of the labels in SDF edges, which does not affect the algorithm results. Usually it is realized as a sequence of elementary retimings, each of them consists of a transferring a group of labels (i.e., registers) from the input edges of some node v to its outputs.

In most cases, it is allowed to increase the latent delay of the algorithm and to insert the additional registers on the inputs or outputs of SDF. After retiming such modified SDF, the pipelined network with low value of T_C is achieved. This technique is called as *SDF pipelining*.

A *cut-set retiming* is an effective metod, which implements the pipelining, and therefore, is widely used for the pipelined datapath design. The *cut-set* in an SFG is a minimal set of edges, which partitions the SFG into two parts. The procedure is based upon two simple rules [46].

Rule 1: *Delay scaling*. All delays D presented on the edges of an original SFG may be scaled, i.e., $D' \rightarrow \alpha D$, by a single positive integer α, which is also known as the pipelining period of the SFG. Correspondingly, the input and output rates also have to be scaled by a factor of α (with respect to the new time unit D'). Time scaling does not alter the overall timing of the SFG.

Rule 2: *Delay transfer*. Given any cut-set of the SFG, which partitions the graph into two components, we can group the edges of the cut-set into inbound and outbound, depending upon the direction assigned to the edges. The delay transfer rule states that a number of delay registers, say k, may be transferred from

outbound to inbound edges, or vice versa, without affecting the global system timing.

These rules provide a method of systematically adding, removing and distributing delays in a SFG and therefore adding, removing and distributing registers throughout a circuit, without changing the function. The cut-set retiming procedure is then employed, to cause sufficient delays to appear on the appropriate SFG edges, so that a number of delays can be removed from the graph edges and incorporated into the processing blocks, in order to model pipelining within the processors; if the delays are left on the edges, then this represents pipelining between the processors.

SDF has the properties that it can be described by VHDL, and then, be translated into the FPGA bit stream [54].

2.2.3 Transfer SDF to the VHDL description

The VHDL language defines the algorithms presented by the graph model of SDF. In this case, the absence of blocking of the algorithm is checked at the compilation time and during execution of the algorithm in the VHDL simulator. SDF is a kind of DFG model, which is subject to certain restrictions. These limitations are reflected in the description of the algorithm in VHDL language.

The main limitation is that the node of the graph must produce a result at each time it is firing. That is, the result of $y(n)$ has the same number or number with constant changes as the input x (n). Then these signals – the input data and the result – will be synchronous. For this, the delay element should be simulated by the processor running on the clock signal, which is the only one for the whole model. In order to satisfy for the delays in closed loops to have a certain initial value, the process operators must have successive operators of initial setup of delayed

signals. These conditions are taken into account when forming the correspondence between the elements of SDF and the operators of the VHDL program (Table 2.2).

Таблиця 2.2.

Representation of the SDF elements by the VHDL operators and clauses

Algorithm element	VHDL-code
Signal $u(n)$	**signal** u: real;
Input and output ports, source and destination of $x(n)$,$y(n)$	**port**(clk, rst: **in** std_logic; x: **in** real; y:**out** real);

Delay $u(n)$ to 1 cycle: $v(n) = u(n-1)$	**process**(clk,rst) **begin**
	if rst ='1' **then**
	v<=0.0;
	elsif clk='1' **and** clk'event **then**
	v<=u;
	end if;
	end process;

Signal addition $y(n) = a(n) + b(n)$	y<=a+b;
Multiply the signal to a constant $y(n) = a \cdot x(n)$	**generic**(a:real:=0,9876); --or **constant** a:real:=0,9876; . . . y<=a*x;

The description of the algorithm in the VHDL language of on the basis of SDF is carried out in the following way. Input and output signals are placed in the IN-OUT ports. Also, the CLK clock signal the total RESET signal are added to the input ports. All data flows coming from the nodes of SDF and delay elements are assigned the corresponding signals of a certain type that appear in the declarative part of the program. The elements of SDF or the signal graph are mapped to the respective VHDL operators according to Table 2.2.

If it is necessary to implement a multi-stroke delay to n times, then a regular type and a signal of this type are declared, for example:

```
type Delay_T is array (0 to n-1) of real;
signal ud: Delay_T;
```

and in the process operator the FIFO memory is programmed for *n* words:

```
process(clk,rst) begin
    if rst ='1' then
        ud<= (others=>0.0);
    elsif clk='1' and clk'event then
        ud<=ud(1 to n-1)&u;
    end if;
end process;
```

Delayed at $m \le n$ cycles, the signal is taken into calculations as ud(m-1).

Setting the filter coefficients, like the **generic** constants, allows us to use this model as a universal filter that is customizable when inserting a component of the filter.

Functions which are performed in the SDF nodes may be arbitrary functions represented by parallel operators. Such functions are always found in the hardware as some logical combinational networks. The process operator must also be reflected in the logical combinational network. For this purpose, the successive conditional operators must be calculated at each start of the process. Such conditional operators have alternating branches that cover all combinations of input condition signals.

For example, the next operator on the left does not appear in the user schema. It describes a certain memory scheme that stores the previous result with the combination a = '0' and b = '1'. The correct operator entry will, for example, be the case in which the else alternative overlaps the rest of the combinations.

```
process(x,v,a,b) begin
    if a ='0' and b='0' then
        y<= x + u;
    elsif a ='1' then
        y<= x - u;
    end if;
end process;
```

```
process(x,v,a,b) begin
    if a ='0' and b='0' then
        y<= x + u;
    elsif a ='1' then
        y<= x - u;
    else
        y<= x;
    end if;
end process;
```

2.3 Method of increasing the FIR filter efficiency

2.3.1 Background of the method

As it is shown in the previous section, there are many methods for implementing the FIR filters that do not have multiplication blocks. Most of them use constant coefficient multiplication (CCM) schemes that have a minimized hardware amount. They have been widely used in FPGA for decades. At the same time, such filters do not use the hardware multiplication blocks that are part of the DSP48 blocks.

It is found out also, that the CCM blocks are effective one if the coefficient bit width is $n < 12$. Consider the usual impulse response of some FIR filter in Fig. 2.2. The filter coefficients are represented by the 16-bit twoth complement data. Among 35 coefficients, up to 30 coefficients can be shortened to 12 digits. Therefore, 5 multiply units can be implemented in DSP48 blocks, and the rest 30 units can be implemented in CCMs.

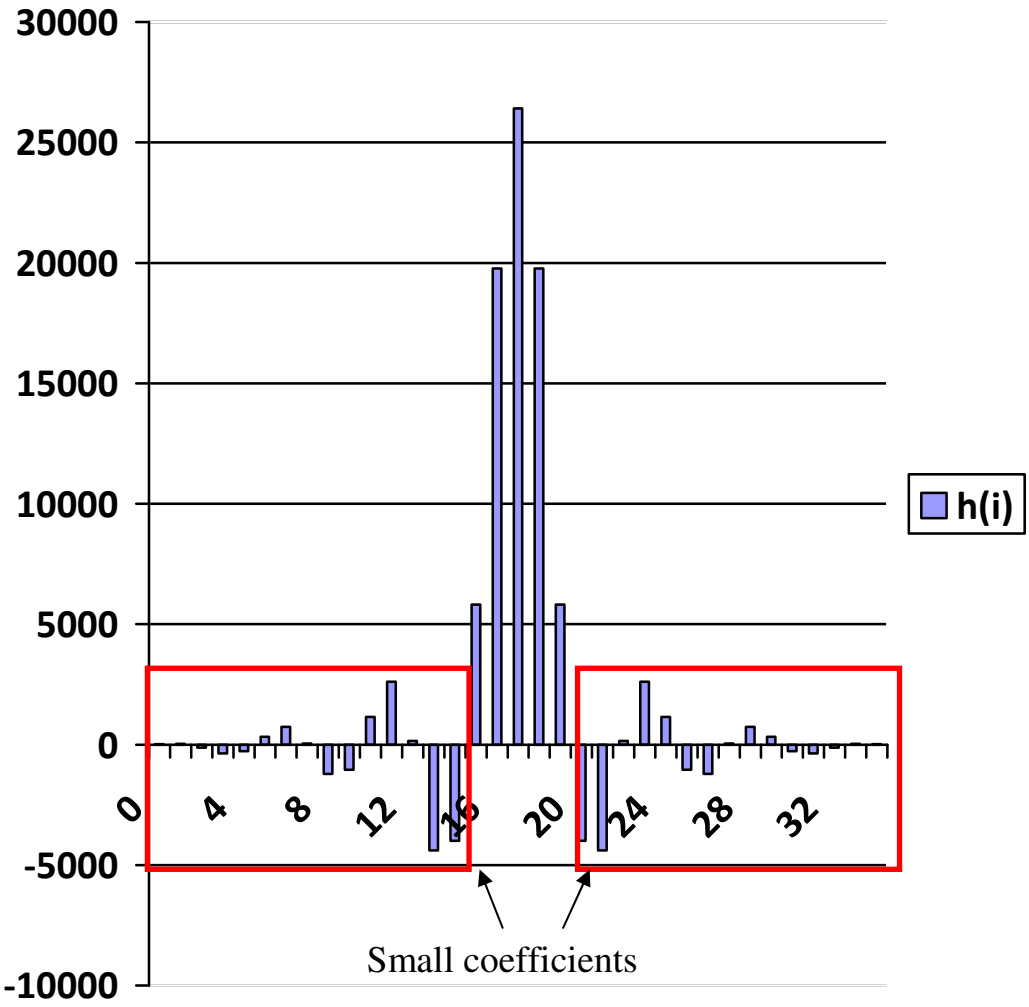


Fig. 2.2. Impulse response of the typical low pass filter, and the selected coefficients, multiplication to which can be implemented in CCM

Therefore, a new approach to the development of a FIR filter can be proposed, which uses both hardware multiplication blocks and CCMs. It provides both an increase of the filter order and a minimized hardware cost, and also supports high filter throughput.

2.3.2 Filter structure optimization

The FIR filter, which is designed for implementation in the Xilinx FPGA, has a well-known systolic structure [15], illustrated by Fig.1.6 and Fig. 1.7 The SDF graph of the systolic structure of the k-th order filter is shown in Fig. 2.3,a, and its symbolic image is shown in Fig. 2.3,b.

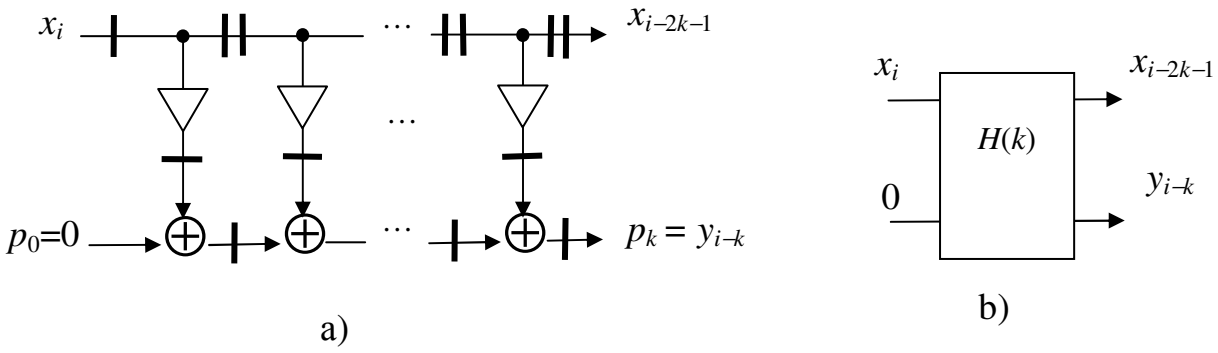


Fig. 2.3 SDFof the systolic structure (a), and its symbolic image (b)

Here, x_i, y_i are the input and output data, circles, triangles and thick segments represent addition, multiplication by a coefficient, and delay for one cycle, respectively. This graph is mapped in the corresponding parallel pipelined structure with a maximum clock rate by means of a one-to-one mapping.

The coefficient set of the n-th order FIR filter form its impulse characteristic $h(n)$, which is similar to one, which is illustrated by Fig.1.6. It is proposed to realize a FIR filter of three blocks, which calculates the convolution with the left $h_1(n_l)$, middle h_2 (n_m), and right h_3 (n_r) subset of the coefficients, where, $n = n_l + n_m + n_r$. An appropriate upgraded SDF is shown in Fig. 2.4.

In most cases, the bit width of $h_1(n_l)$, and $h_3(n_r)$ are much less than the bit width of coefficients $h_2(n_m)$ (see Fig. 1.6). Therefore, it is desirable to implement

part of the filter marked in Fig. 1.8 as H_2 (n_m) using the DSP48 blocks and the rest of the filter using the CCM blocks.

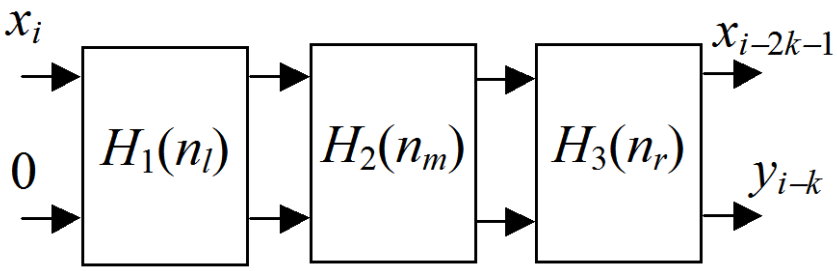


Fig. 2.4. Modernized SDF of the FIR filter

The third block of the filter, which is the last in the graph in Fig. 2.4, should have an increased bit width of the intermediate results in order to maintain a low level of the truncation errors. This can significantly reduce the efficiency of the filter, since the multi-bit adder based on the LUTs is much slower than the corresponding adder in the DSP48 block.

In order to minimize the hardware volume and critical path delay due to the increased adder bit width, the improved SDF is proposed. It is shown in Fig. 2.5. The corresponding improved FIR filter contains two parts of $H_1(n_l)$ and $H_3(n_r)$ with the same bit width, which is much smaller than the bit width of operands in the DSP48 block.

2.3.3 CCM block implementation

As it is shown in the previous paragraph, the submodules $H_1(n_l)$ and $H_3(n_r)$ are preferably be implemented on the base of CCM. In the subsection 1.3 the methods of the CCM design are reviewed.

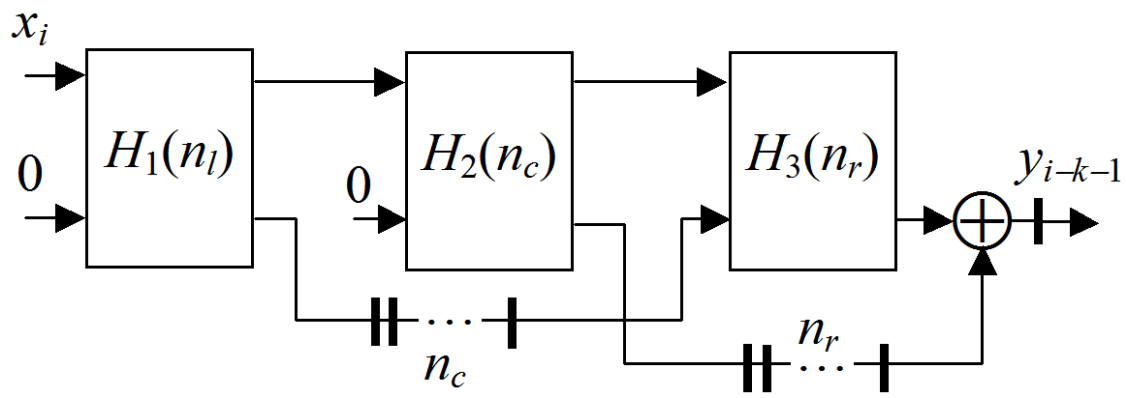


Fig. 2.5. Improved SDF of the FIR filter

And it was shown that the *n*-bit coefficient multiplication can be implemented in approximately *n*/3-input adder tree. In some situations of the MSD coefficient presentation, the numder of the shifted terms can be less than *n*/3.

In the previous chapter, it was found out that the modern FPGA architecture provides the three-input adder-subtractor, which is formed as a single stage of the 6-input LUTs. Such an adder can perform the CCM block for the MSD-represented coefficient with three non-zero bits, like: $\bar{1}00\bar{1}01$. Therefore, such CCM can perform a basic block for the FIR filter part, which as the 6-bit coefficients.

In the subsection 2.1 it was shown, that the delay of the DSP48 block is approximately in two times higher than the delay of two adder stages. Therefore, it is reasonable to implement the CCM block in two stages formed by the three-input adders-subtractors. Therefore, the SDF of the CCM module for some "unconvenient" MSD-coefficient of the maximum bitwidth 205 = 256 – 64 + 16 – 4 + 1 = = $10\bar{1}010\bar{1}01$ looks like one in Fig. 2.6.

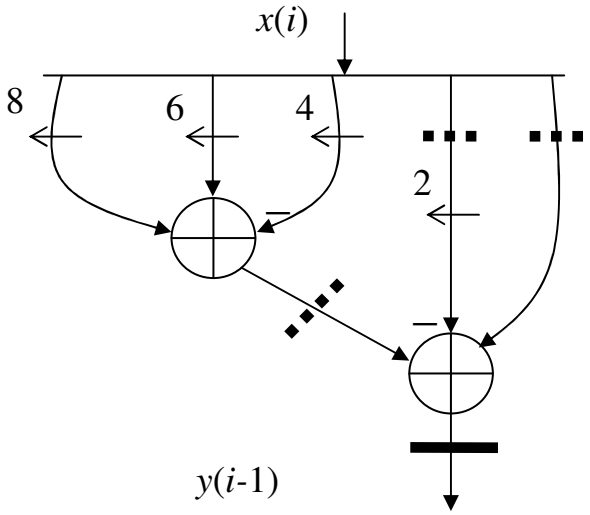


Fig. 2.6. Constant coefficient multiplier SDF for the coefficient 205

SDF in Fig. 2.6. is described in VHDL as follows:

```
Process(CLK, RESET) begin
    If CLK = '1' and CLK'event then
        If RESET = '1' then
            y(i-1)<=(others => '0');
        else
            y(i-1)<= SXT(x(i),y(i-1)'left) – SHL(x(i),"010") +
        (SHL(x(i),"1000") + SHL(x(i),"110") – SHL(x(i),"100"));
        End if;
    End if;
End process;
```

Here, x(i),y(i-1) are components of the respective signal arrays, function SXT expands the bit width of input bit vector to the bit width of the result, function SHL shifts left the operand to the given bit number.

As a result, it is guaranteed to build the CCM block for the 9-bit coefficients, which speed is comparatively equal to the speed of the DSP48 block.

The CCM can be pipelined. The pipelining example is shown in Fig. 2.6. by the dotted lines. Then the limit of the network delay can be removed. But the CCM pipelining infers the pipelining of the whole filter structure. And as a result, the latent delay of the FIR filter may increase.

However, the FPGA architecture provides practically any level of the network pipelining.

The effective approach is to represent the coefficient set $h(n)$ in the MDS-form. Then, the coefficients are selected, in which presentation are no more 5 non-zero bits. These coefficients are implemented in the respective CCM blocks of the submodules $H_1(n_l)$ and $H_3(n_r)$ (see Fig. 2.5).

2.3.4. Scaling the data in filters

The example of the CCM block design in the previous paragraph shows that the integer arithmetic is usually used in the digital network design. This arithmetic is considered as the general data representation in the computer design. So, the bit vectors and data of the type STD_LOGIC_VECTOR are considered as integer values.

However, in DSP applications, the real data, or data in the floating point representation is widely used. In the DSP microprocessors, the main data representation is integer data. But which are scaled.

The fact is that the fixed point data representation is considered in the DSP algorithms, especially, in the FIR filters. Therefore, both the data and coefficients are the fixed point data, in which the point stays between sign bit and the most significant bit, which weight is 2^{-1}.

Taking into account the scaled fixed point data representation, SDF in Fig.2.6 is redrawn as one in Fig.2.7.

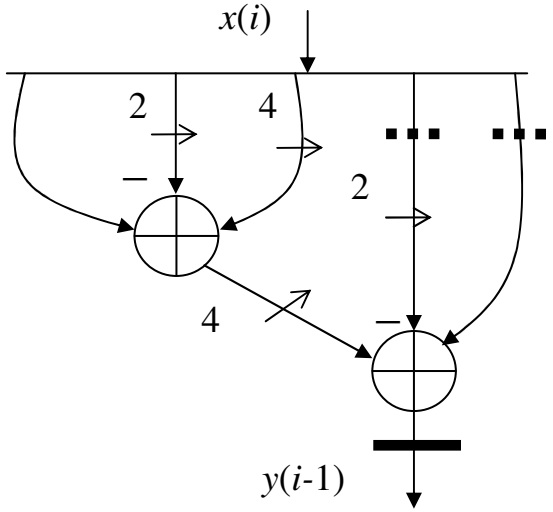


Fig. 2.7. Constant coefficient multiplier SDF for the coefficient 205/128

In FIG. 2.7, the scaled coefficient is decomposed as

$$10\bar{1}010\bar{1}01_{MSD}/128 = 10\bar{1}/8 + 10\bar{1}01/(16*8).$$

Therefore, SDF performs the calculations

$$y = x - x/4 + (x - x/4 + x/16)/16.$$

Then, SDF in Fig. 2.7. is described in VHDL as follows:

```
Process(CLK, RESET) begin
    If CLK = '1' and CLK'event then
        If RESET = '1' then
            y(i-1)<=(others => '0');
        else
            y(i-1)<= SXT(x(i),y(i-1)'left) - SHR(x(i),"010") +
        SHR(x(i) – SHR(x(i),"010") + SHR(x(i),"100")),"100");
        End if;
    End if;
End process;
```

Such CCM performance considers the truncation of both the intermediate and final results. This provides the substantial adder length shortening and respective hardware minimization. But for the preserving of the minimized error level, from 2 to 3 additional least significant bits have to be added to the data representation.

2.3.5 Method for the effective FIR filter design

The initial data for the method are the filter order n, impulse response $h(n)$, input and output data bit widths. The result of the method is the VHDL description of the FIR filter, which is described in the synthesis style, and therefore, can be configured in FPGA.

At the first step, the impulse response $h(n)$ is analyzed. All the coefficients are represented in the twos complement codes. Among them, the coefficients, which have no more than 12 significant bits, are recoded into MSD representation. Then, all the coefficients in $h(n)$ are divided in three sets: $h_1(n_l)$, h_2 (n_m) and $h_3(n_r)$, which are formed from the left (n_l), middle (n_m), and right (n_r) coefficients of the impulse response. And the sets $h_1(n_l)$, and $h_3(n_r)$ are formed by the coefficients, in which MSD presentation no more 5 ones are present.

At the second step, the constant coefficient multipliers are formed, which are implemented according to SDF shown in Fig 2.7.

At the third step, the filter structure is formed, which is shown in Fig. 2.5. Here, the blocks $H_1(n_l)$, and $H_3(n_r)$ are implemented using the coefficient sets : $h_1(n_l)$, and $h_2(n_m)$, respectively, using the CCMs, formed in the previous step. The H_2 (n_m) blocks are implemented as usual, using the multiplier blocks like DSP48. The example of the respective structure of this block is shown in Fig. 1.6.

At the fourth step, the filter structure is described in VHDL. By this process, in each blocks $H_1(n_l)$, H_2 (n_m), and $H_3(n_r)$ the elementary stage is selected, which contains a single multiplier, a single adder, and respective pipeline registers.

This stage is described in VHDL as a single process operator, and is tunable according the respective coefficient-multiplicand. Then, the block model is formed by repeating the process operator using the GENERATE operator. The designed FIR filter model is tested by the FIR filter test bench, which is described in the next section.

2.5 Preliminary conclusions

In this section, the method for the effective FIR filter design is designed. The method is based on implementing the filter structure of three parts, two of them are implemented on the base of the constant coefficient multipliers to the small coefficients of the filter impulse response. Due to this, the filter hardware volume is less, maximum filter length is higher, than by the usual FIR filter implementation, providing the effective resourse utilization of FPGA.

In the next section, the method effectiveness is proven.

3 **IMPLEMENTATION OF THE FIR FILTERS IN FPGA**

3.1 Description of the FIR filter in VHDL

3.1.1. Description of the filter stage

Due to the method, proposed in the previous section, the filter structure is described in VHDL in the following order. The stages of the blocks $H_1(n_l)$, $H_3(n_r)$, which contains a single multiplier, a single adder, and respective pipeline registers. are described in VHDL as a single process operator, and is tunable according the respective coefficient-multiplicand.

By this process, the multiplication to the coefficient is implemented due to the approach described in the subsection 2.3. Besides, taking into account that two stages of adders in the modern FPGA have 5 inputs for the terms, and that each stage has to add the result from the previous stage, then the number of the non-zero terms in the MSD coefficient presentation is selected as 4.

The stage SDF looks like one in Fig. 3.1.

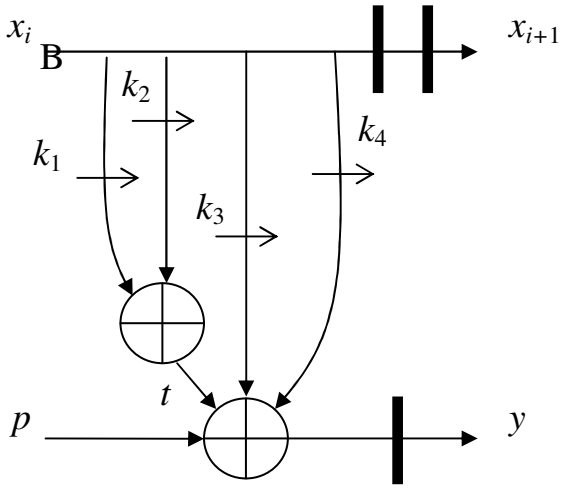


Fig3.1. SDF of a single stage of the FIR filter

The VHDL-description of the SDF graph in Fig.3.1 looks like the following.

```
process(CLK,RST)
        variable t:STD_LOGIC_VECTOR(nb-1 downto 0);
    begin

        if CLK = '1' and CLK'event       then
            if RST='1' then
                AD<= (others=>'0');
                BD<= (others=>'0');
                BD2<= (others=>'0');
                BD3<= (others=>'0');
                pr<= (others=>'0');
                P<= (others=>'0');
            else
                BD<=B;
                BD2<=BD;
                if k3>0 and k4>0 then
                    case s2 is
            when 0 => t := PIN + SHR(B,CV(k1,5))+ SHR(B,CV(k2,5));
            when 1 => t := PIN + SHR(B,CV(k1,5)) - SHR(B,CV(k2,5));
            when 2 => t := PIN - SHR(B,CV(k1,5))+ SHR(B,CV(k2,5));
            when others => t := PIN - SHR(Bd,CV(k1,5))
                                - SHR(B,CV(k2,5));
                    end case;
                elsif k3>0 and k4=0 then
                    case s2 is
                        when 0|1 => t := PIN + SHR(B,CV(k1,5));
                        when others => t := PIN - SHR(B,CV(k1,5));
                    end case;
```

```
                elsif k3=0 then
                end if;
                if k1>0 and k2>0 then
                        case s1 is
            when 0 => p <= t + SHR(B,CV(k3,5)) + SHR(B,CV(k4,5));
            when 1 => p <= t + SHR(B,CV(k3,5)) - SHR(B,CV(k4,5));
            when 2 => p <= t - SHR(B,CV(k3,5)) + SHR(B,CV(k4,5));
            when others => p <= t - SHR(B,CV(k3,5)) - SHR(B,CV(k4,5));
                        end case;
                elsif k1>0 and k2=0 then
                        case s1 is
                                when 0|1 => p <= t + SHR(B,CV(k3,5));
                                when others => p <= t - SHR(B,CV(k3,5));
                        end case;
                elsif k1=0 then
                        p <= t;
                end if;
            end if;
        end if;
    end process;
```

Here, the variables k_1, k_2, k_3, k_4 are the generic variables, which code the number of shifts right of the data when it is multiplied by a constant (see Fig. 2.7). And when $k_i = 0$, then the respective term is absent, i.e., it is multiplied by a zero. The generic variables s_1, s_2 code the signs of the terms in the sum of the shifted multiplicands. So, the stage calculates the following formula, if the real non-zero terms are 4:

$$y = (p + \xi(s_2)x\,2^{k1} + \xi(s_2)x\,2^{k2}) + \xi(s_1)x\,2^{k3} + \xi(s_1)x\,2^{k4},$$

where $\xi(s_i)$ is the sign function, which is equal to ±1 depending on the code s_i.

Then, the block model is formed by repeating the process operator using the GENERATE operator:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all, IEEE.STD_LOGIC_arith.all,
use IEEE.STD_LOGIC_signed.all;
entity STS1 is
  generic(nb: natural:=16;
  k1: natural:=1;
  k2: natural:=3;
  k3: natural:=5;
  k4: natural:=7;
  s1: integer:=0;   -- + +
  s2: integer:=3    -- - -
  );
      port(
            CLK : in STD_LOGIC;
            RST : in STD_LOGIC;
            B   : in STD_LOGIC_VECTOR(15 downto 0); -- input signal
            PIN : in STD_LOGIC_VECTOR(nb-1 downto 0); --input SOP
            P   : out STD_LOGIC_VECTOR(nb-1 downto 0); --output SOP
            BO: out STD_LOGIC_VECTOR(15 downto 0)-- delayed signal
            );
end STS1;
```

Here, the nb is the generic constant which is equal to the intermediate data bit width.

The stage with the combinatorial multiplier are described as usual.

The whole FIR filter is described using the GENERATE operator. For example, the H_1 filter section (see Fig. 2.5) for some low pass filter is shown below.

```
        type Tc is array (1 to n1) of integer;
        constant k1a1:Tc:= (6, 4,4,4,2, 2,7,1);
        constant k2a1:Tc:= (9, 6,7,7,5,11,0,5);
        constant k3a1:Tc:= (10,9,9,9,10,0,0,7);
        constant k4a1:Tc:= (0,11,11,11,0,0,0,0);
        constant s1a1:Tc:= (0,0,1,2,2,2,0,1);
        constant s2a1:Tc:= (0,3,0,1,0,0,0,0);
        type Tsign16 is array (0 to nst) of STD_LOGIC_VECTOR(15 downto 0);
        type Tsign40 is array (0 to nst) of STD_LOGIC_VECTOR(39 downto 0);
type Tsignnb1 is array (0 to n1) of STD_LOGIC_VECTOR(nb1-1 downto 0);
type Tsignnb2 is array (0 to nst) of STD_LOGIC_VECTOR(nb2-1 downto 0);
        signal bd: Tsign16;
        signal pd: Tsign40;
        signal pd1:Tsignnb1;
        signal pd2:Tsignnb2;
        signal pdd1:STD_LOGIC_VECTOR(nb1-1 downto 0);
        signal pdd2:STD_LOGIC_VECTOR(39 downto 0);
        signal DOi:STD_LOGIC_VECTOR(16 downto 0);
        constant zz:STD_LOGIC_VECTOR(39 downto 0):=(others=>'0');
        type Tsign16 is array (0 to nst) of STD_LOGIC_VECTOR(15 downto 0);
        type Tsign40 is array (0 to nst) of STD_LOGIC_VECTOR(39 downto 0);
type Tsignnb1 is array (0 to n1) of STD_LOGIC_VECTOR(nb1-1 downto 0);
type Tsignnb2 is array (0 to nst) of STD_LOGIC_VECTOR(nb2-1 downto 0);
        signal bd: Tsign16;
```

```
        signal pd1:Tsignnb1;
begin
        pd1(0)<=(others=>'0');
        bd(0)<= DI;
        stageSM1:for i in 0 to n1-1 generate
                U1: STS2 generic map(nb=>nb1,
                        k1 =>k1a1(i+1),
                        k2=>k2a1(i+1),
                        k3=>k3a1(i+1),
                        k4=>k4a1(i+1),
                        s1=>s1a1(i+1),
                        s2=>s2a1(i+1)
                        )
                port map(CLK, RST,
                        B =>bd(i),
                        PIN => pd1(i),
                        P   => pd1(i+1),
                        BO  => bd(i+1)
                        );
        end generate;
```

The full version of the FIR filter description is represented in Appendix 1.

3.2 Modeling the filter

To test the filter, some special signals are inputted in it and the filter reaction is investigated. The sine waves are usually selected as such input signals because they do not exchange their form but only their magnitude. This is explained by the fact that the sine wave is the eigenfunction for the linear filters. To say more precisely, the genuine eigenfunction is the analytical (complex) signal, which is represented by the couple of cosine (inphase) and sine (quadrature) signals.

A set of reactions to the analytical signal with different frequencies is named as the frequency characteristic of the given filter. The magnitude-frequency characteristic and phase-frequency characteristic are usually distinguished for the filter. It is recommended to test the DSP filters by deriving these characteristics.

For this purposes the filter testbench component is proposed for the use in the digital filter design and investigations of them. The connection of the filter component to the tested filter instantiations is shown in Fig.3.2.

It uses two identical copies of the system $H(z)$, to which the analytic signal is applied, which consists of sine and cosine parts. Accordingly, the system outputs the components of the analytical signal of the system response. Unlike analog systems, the designer can always get two identical digital systems for such an experiment.

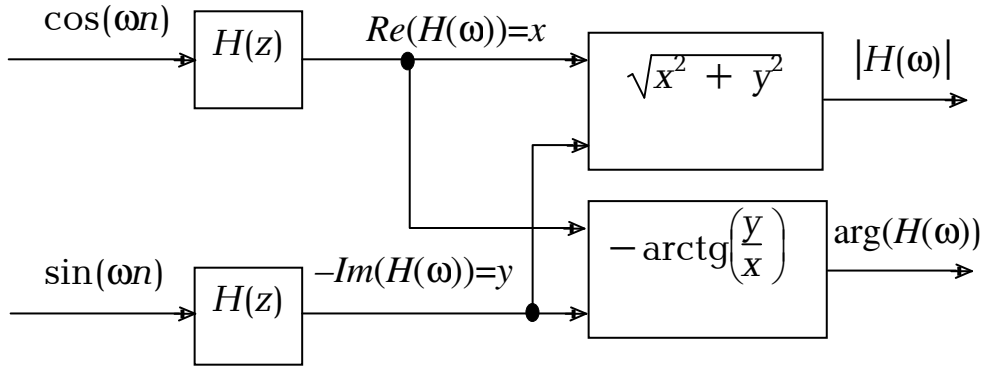


Fig. 3.2. Measurement of frequency characteristics of the real filter $H(z)$

Due to these relations, the testbench for the filter testing is proposed in [55]. The filter module, which is designed due to a new method is inserted into the testbench, as shown in Fig. 3.3. With the help of this testbench, the diagrams of the AFC and PFC are plotted, as shown in Fig. 3.4.

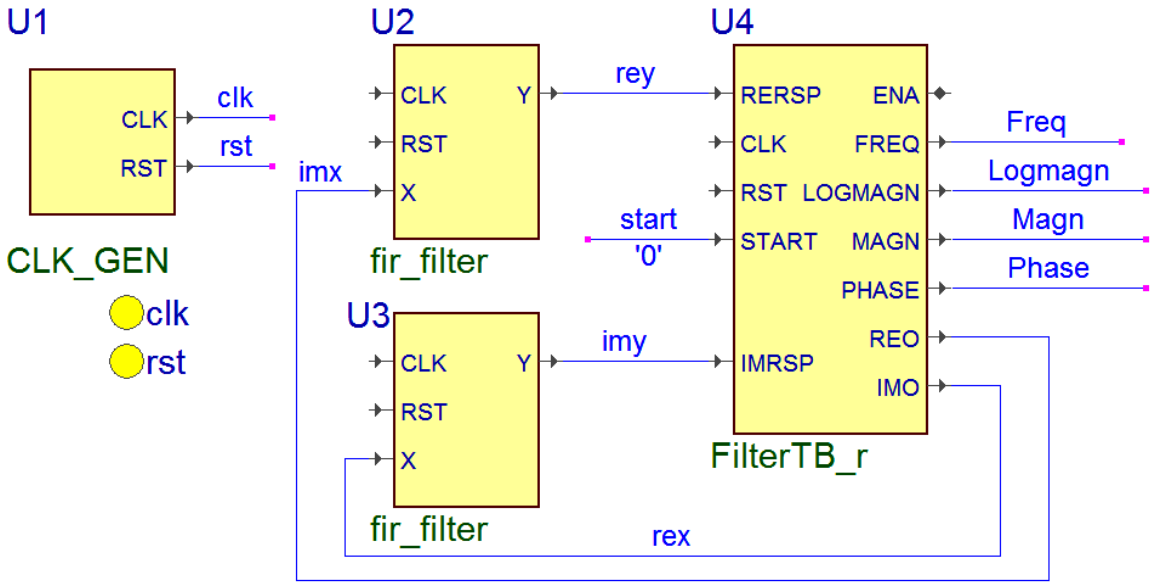


Fig.3.3. Testbench for the FIR filters

So, the testbench in Fig.3.3 is the effective tool for the FIR filter testing and parameter measurement. It can show both the truncation effect for the coefficient and intermediate results and the effectiveness of the CCM use.

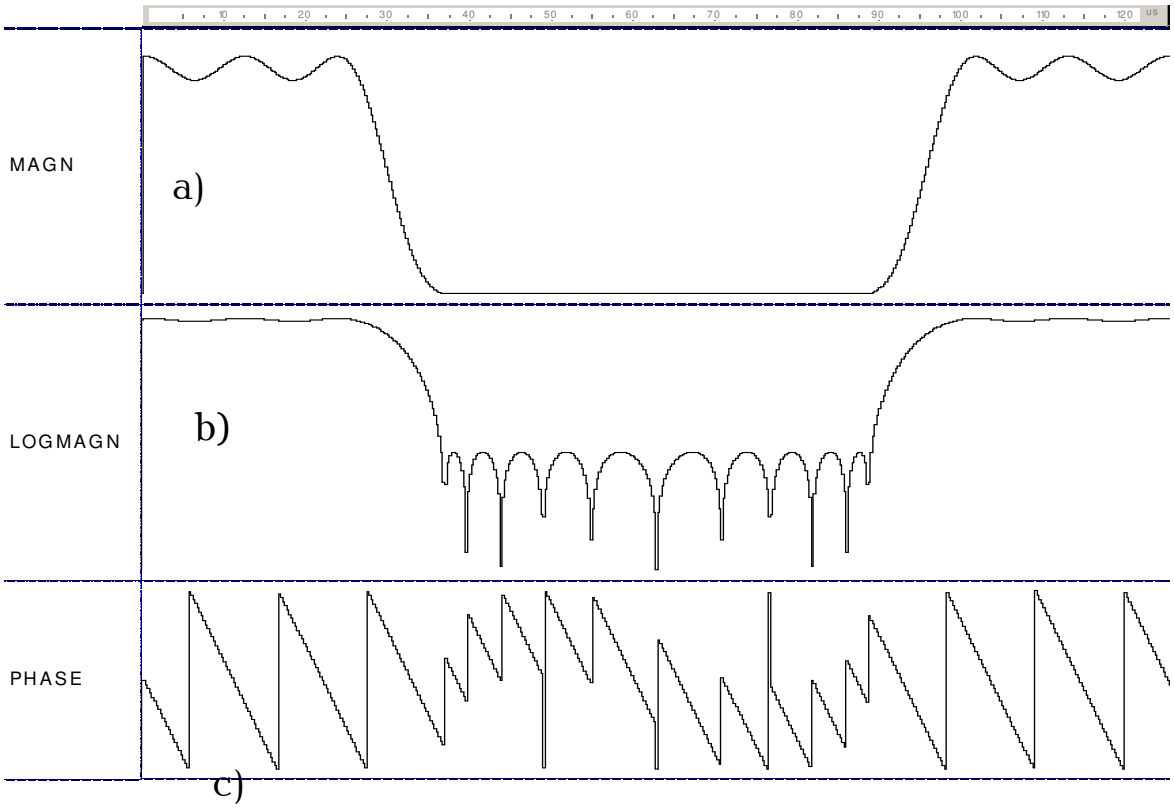


Fig.3.4. Measured characteristics of the designed example of the FIR filter: AFC (a), logarithmic AFC (b), and PFC (c)

3.3 Implementation in FPGA

The low pass FIR filter of the 32-d order were designed according to a new method. The filter structures in Fig 1.6, 2.4, 2.5 were described by the VHDL language, as it is shown above. Then, the filter projects were synthesized for the Xilinx Spartan-6 FPGA. The results of the synthesis for filters are shown in the Table 3.1. To estimate the whole hardware volume *Q,* it was considered, that an 18-bit multiplier in the FPGA requires 200 equivalent LUTs for its implementation.

Table 3.1.

Characteristics of the FIR filters with different structures

FIR structure	Hardware volume				Maximum f_C, MHz	f_C/Q
	LUTs	Registers	DSP48	Q		
Systolic ,Fig.1.6	0	0	33	33	390	11,8
Modernized, Fig.2.4	772	1538	17	20,9	146	7,0
Improved, Fig.2.5	914	1639	17	21,6	267	12,4

The Table 3.1 analysis shows that the filter structure based on DSP48 blocks (Fig.1.6), provides the maximum sampling clock frequency f_C at the cost of the hardware volume. The modernized structure gives the smallest value of f_C, but with the minimum hardware volume. The improved structure provides much less hardware volume and better ratio throughput to hardware volume f_C/Q.

3.4 Concluding remark

In this section, two FIR filters are designed. One of them has the modernized structure, and the others is designed exactly according to the proposed method. The filters are tested and configured in FPGA.

The results of the condiguration show that the modernized structure gives the small value of the clock frequency, but has the minimum hardware volume? And the improved structure provides much less hardware volume and better ratio throughput to hardware volume comparing to the best solution of the Xilinx company.

CONCLUSIONS

This thesis has presented a detailed description and analysis of the high-speed digital FIR filter design for the FPGA implementation. On the base of the thesis materials the following conclusions are made.

1) The FIR filter IP cores, proposed by the FPGA vendors, have the limitations of the filter order and utilize the configurable hardware, and therefore, they need modernizations.

2) A method for the effective FIR filter design is developed. The method is based on implementing the filter structure of three parts, two of them are implemented on the base of the constant coefficient multipliers to the small coefficients of the filter impulse response. Due to this, the filter hardware volume is less, maximum filter length is higher, than by the usual FIR filter implementation, providing the effective resourse utilization of FPGA.

3) The FIR filters which are designed according to the new method provide much less hardware volume (in 1.5 times) and better ratio throughput to hardware volume (to 5%) comparing to the best solution of the Xilinx company.

REFERENCES

1. Рабинер Л., Гоулд Б. Теория и применение цифровой обработки сигналов. М.: Мир. 1978. — 848 с.
2. Хемминг Р. В. Цифровые фильтры. М.: Сов. Радио. 1980. — 224 с.
3. Alter D. M. Efficient Implementation of Real-Valued FIR Filters on the TMS320C55x DSP. Application Report SPRA655/ TI com. April 2000. – 27 P. [Electronic resource] Available at: http://web.iitd.ac.in/~saifkm/docs/EEL319/Practical/fir.pdf.
4. FIR II IP Core User Guide // Altera Inc. – 62 P. [Electronic resource] Available at: https://www.altera.com/en_US/pdfs/literature/ug/ug_fir_compiler_ii.pdf.
5. FIR Compiler v7.2 LogiCORE IP Product Guide Vivado Design Suite. PG149/ Xilinx inc.// November 18, 2015. –131 P. [Electronic resource] Available at: https://www.xilinx.com/support/documentation/ip_documentation/fir_compiler/v7_2/pg149-fir-compiler.pdf
6. Quadir S.H. Minimized FIR Filter Design Implemented in FPGA / Vinogradov Ju.N., Sergiyenko A.M., // System analysis and information technology: 20-th International conference SAIT 2018, Kyiv, Ukraine, May 21 – 24, 2018. Proceedings. – ESC “IASA” NTUU “Igor Sikorsky Kyiv Polytechnic Institute”, 2018. – P. 165–166.
7. Кадір С. Х. фільтри зі скінченною характеристикою з мінімізованими апаратними витратами / А. Сергієнко, А. Сергієнко // Праці міжнародної конференції "Безпека, Відмовостійкість, Інтелект", 10-11 травня 2018. — С. 70 — 73.
8. Constantinides G. Synthesis and Optimization of DSP Algorithms / ,P.Y.K. Cheung,W. Luk / Kluver. 2004. – 165 p.
9. Sabin W.E. Discrete-Signal Analysis and Design. John Wiley. 2008. –174 P.

10. Khan S. A. Digital design of signal processing systems: a practical approach. – John Wiley & Sons. –2011. –586 p.
11. Сергієнко А. М. Цифрова обробка сигналів. Комп′ютерний практикум мовою VHDL / Ю. М. Виноградов, Т. М. Лесик / К.: НТУУ«КПІ», 2012. – 104 с. [Електронний ресурс] Режим доступу: http://kanyevsky.kpi.ua/wp-content/uploads/2017/11/DSP_LabS.pdf
12. Series FPGAs CLB User Guide. UG474 (v1.7). November 17, 2014. – 74 P. [Electronic resource]. Available at www.xilinx.com
13. Spartan-6 FPGA DSP48A1 User Guide. UG389 (v1.2). May 29, 2014. – 46 P. [Electronic resource]. Available at www.xilinx.com
14. Virtex-5 XtremeDSP Design Considerations User Guide. UG193 (v2.7) December 11, 2007. – 114 p. [Electronic resource]. Available at www.xilinx.com
15. Kung S. Y. VLSI and modern signal processing /H. J. Whitehouse / Prentice-Hall. 1985. 481 P.
16. 7 Series DSP48E1 Slice User Guide UG479 (v1.10). March 27, 2018. –58 p. [Electronic resource]. Available at www.xilinx.com
17. Yuan, X. Improved Design of Multiplier in the Digital Filter / T. Ying, G. Chunpeng // International Conference on Computer and Communication Technologies in Agriculture Engineering. 2010.
18. Abedelgwad A. High speed and area efficient multiply Accumulate (MAC) Unit for Digital Signal Processing Applications // IEEE International Symposium on Circuits and Systems, ISCAS, 2007.
19. Fayed A. A merged Multiplier Accumulator for High Speed Signal Processing Applications // IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP). 2002.

20. Kolling P. FPGA Implementation of High Performance FIR filter / K. Abbot/ Proceedings IEEE International Symposium on Circuits and Systems, ISCAS. 1997.
21. Meher P.K. Low-Latency Hardware-Efficient Memory-Based Design for Large Order FIR Digital Filters, 6th International Conference on Information, Communication and Signal Processing. 2007. — P. 10–13.
22. Goslin G.R. A Guide to using FPGAs for application-specific DSP performance, in Xilinx application note. 1998.
23. Meher P.K. FPGA Realization of FIR Filters by Efficient and Flexible Systolization using Distributed Arithmetic / S. Chanderasekaran, A. Amira //. IEEE Transactions on Signal Processing, vol. 56, 2008. No. 7.
24. Wei G. The Implementation of FIR Low-Pass Filter Based on FPGA and DA / W.F. Ying // 4th International Conference on Intelligent Control and Information Processing, Beijing, China. 2013.
25. Zhao L. Design of digital FIR band pass filter using distributed algorithm based on FPGA / , W.H. Bi, F. Liu // Electron. Meas. Technol. V. 30, No. 7, 2010. — P. 101–104.
26. Sergyienko, A. FIR filter soft core generator / O. Vasylienko, V., Maslennikow// IV Konferencja Krajowa „Reprogramowalne uklady cyfrowe”, RUC’2001. Szczecin, Poland. 2001. – P. 167-172.
27. Shi P. Design of linear phase FIR filters with high probability of achieving minimum number of adders / Y.J. Yu //. IEEE Circ. Syst. Soc. V.58, No.1, 2011.— P. 126–136.
28. Mu N. Study on the FPGA Implementation Algorithm of Effective FIR Filter Based on Remainder Theorem / G. Liu // 2nd International Conference on Consumer Electronics, Communication and Networks, 2012. — P. 21–23.
29. Zhon Y. Distributed Arithmetic for FIR Implementation on FPGA / P. Shi // International Conference on Multimedia Technology, Hangzhou, China. 2011.

J.E. Carletta, M.D. Rayman, Practical Considerations in the Synthesis of High Performance Digital

30. Filters for Implementation on FPGAs. Springer, FPL 2002, LNCS 2438, 2002. — P. 886–896.
31. Chou J. FPGA Implementation of Digital Filters / S. Mohanakrishnan, J.B. Evans // in Proceedings 4th International Conference on Signal Processing Applications and Technology. 1993.
32. Woods R. FPGA Synthesis on the XC6200 using IRIS and Trianus/Hades / ., S. Ludwig, J. Heron, D. Trainor, S. Gehring // Proceedings 5th IEEE Symposium on FPGA based Custom Computing Machines, 1997. — P. 155–164.
33. Parhi K.K. Chapter 13, Bit-Level Arithmetic Architectures. VLSI Digital Signal Processing Systems Design and Implementation. Wiley, 1999.
34. Meher P.K. Low-Latency Hardware-Efficient Memory-Based Design for Large Order FIR Digital Filters // 6th International Conference on Information, Communication and Signal Processing, 2007. — P. 10–13.
35. Meher P.K. FPGA Realization of FIR Filters by Efficient and Flexible Systolization using Distributed Arithmetic / ., S. Chanderasekaran, A. Amira // IEEE Transactions on Signal Processing, vol. 56, 2008. No. 7.
36. Uma R. Systolic FIR Filter Design with Various Parallel Prefix Adders in FPGA: Performance Analysis / J. Ponnian // International Symposium on Electronic System Design, Kolkatta, 2012. —P. 19–22.
37. Yuan X. Improved Design of Multiplier in the Digital Filter / T. Ying, G. Chunpeng // International Conference on Computer and Communication Technologies in Agriculture Engineering. 2010.
38. Chapman K. Constant Coefficient Multipliers for the XC4000E // Xilinx Technical Report. 1996.

39. Wirthlin M.J. Constant coefficient multiplication using look-up tables / J. VLSI Signal Process. V.36, 2004. — P. 7–15.
40. Wirthlin M.J. Efficient Constant Coefficient Multiplication Using Advanced FPGA Architectures / B. McMurtrey // International Conference on Field Programmable Logic and Applications (FPL). 2001.
41. Cheng C. Hardware efficient fast parallel FIR filter structures based on iterated short convolution / K.K. Parhi // IEEE Trans. Circ. Syst. I Reg. Pap. V. 51, 2004. No. 8. — P. 1492–1500.
42. Pavlovic V.D. 1D and 2D Economical FIR filters generated by Chebyshev polynomials of the first kind / N. Doncov, D. Ciric // Int. J. Electronic. 2013.
43. Tsao Y.C. Area efficient parallel FIR digital filter structures for symmetric convolutions based on fast FIR algorithm / K. Choi // IEEE Trans. VLSI Syst. V. 20. 2010. No.2. — P. 366–371. Y.C.
44. Tsao Y.C. Hardware-Efficient VLSI Implementation for 3-Parallel Linear-Phase FIR Digital Filters of Odd Length /K. Choi // IEEE International Symposium on Circuits and Systems, Seoul. 2012.
45. Kumm M. Multiple Constant Multiplication Optimizations for Field Programmable Gate Arrays. Springer, 2016. — 206 p.
46. Woods R. FPGA-based Implementation of Signal Processing Systems / J. McAllister, G. Lightbody, Y.Yi / John Wiley & Sons, Ltd. 2008. — 364 P.
47. Avizienis A. Signed-Digit Number Representations for Fast Parallel Arithmetic // IEEE Transactions on Electronic Computers, vol. EC-10, 1961. No. 3. — P. 389–400.
48. Поспелов Д.А. Арифметические основы вычислительных машин дискретного действия. — 1970. — 308 с.
49. Hwang K. Computer Arithmetic: Principles, Architecture, and Design. New York: Wiley-Interscience, 1979.

50. Potkonjak M. Multiple Constant Multiplications: E□cient and Versatile Framework and Algorithms for Exploring Common Subexpression Elimination / M. B. Srivastava, and A. P. Chandrakasan // IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 15, 1996. No. 2. — P. 151–165.
51. Hartley R. Subexpression Sharing in Filters Using Canonic Signed Digit Multipliers // IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, Vol. 43, 1996. No. 10. — P. 677–688t.
52. Edwards S. Design of Embedded Systems: Formal Models, Validation, and Synthesis / L. Lavagno, E.A. Lee, A. Sangiovanny-Vincentelli // Proc. IEEE, vol.85, pp.366–390, March 1997.
53. Khan S. A, Digital Design of Signal Processing Systems. John Wiley & Sons. 2011.
54. Сергиенко А.М. Отображение периодических алгоритмов в программируемые логические интегральные схемы / В.П. Симоненко // Электронное моделирование. Т. 29. No.2. 2007. – С.49–61.
55. .Sergyienko A. Testbench for the filter testing. [Electronic resource] Available at: http://kanyevsky.kpi.ua/en/useful-ip-cores/testbench-for-the-filter-testing/

APPENDICES

APPENDIX 1

VHDL programs of the FIR filter

```
------------------------------------------------------------------------------
--
-- Title       : FIR1
-- Design      : FIR1
------------------------------------------------------------------------------
-- File        : STS1.vhd
-- Generated   : Tue Feb 20 10:18:11 2018
-- From        : interface description file
-- By          : Itf2Vhdl ver. 1.20
------------------------------------------------------------------------------
--
-- Description :  FIR stage, multiplication by shift&add
--             t = PIN + b^(-k1)+ b^(-k2)
--             p = t + b^(-k3) + b^(-k4)
--          1-staged stage
------------------------------------------------------------------------------
-- Spartan6
-- 48 LUT 12 CLBs 61 TG 5.479ns

library IEEE;
use IEEE.STD_LOGIC_1164.all, IEEE.STD_LOGIC_arith.all,IEEE.STD_LOGIC_signed.all;
entity STS1 is
   generic(nb: natural:=16;
   k1: natural:=1;
   k2: natural:=3;
   k3: natural:=5;
   k4: natural:=7;
   s1: integer:=0;       -- + +
   s2: integer:=3        -- - -
   );
         port(
                  CLK : in STD_LOGIC;
                  RST : in STD_LOGIC;
                  B   : in STD_LOGIC_VECTOR(15 downto 0);
                  PIN : in STD_LOGIC_VECTOR(nb-1 downto 0);
                  P   : out STD_LOGIC_VECTOR(nb-1 downto 0);
                  BO  : out STD_LOGIC_VECTOR(15 downto 0)
                  );
end STS1;
architecture synt of STS1 is
signal AD,BD,BD2,BD3:STD_LOGIC_VECTOR(15 downto 0);
signal pr:STD_LOGIC_VECTOR(nb-1 downto 0);
alias CV is IEEE.std_logic_arith.CONV_STD_LOGIC_VECTOR [INTEGER, INTEGER return
STD_LOGIC_VECTOR];
```

```
begin
	process(CLK,RST)
		variable t:STD_LOGIC_VECTOR(nb-1 downto 0);
	begin

		if CLK = '1' and CLK'event	then
			if RST='1' then
				AD<= (others=>'0');
				BD<= (others=>'0');
				BD2<= (others=>'0');
				BD3<= (others=>'0');
				pr<= (others=>'0');
				--t<= (others=>'0');
				P<= (others=>'0');
			else
				BD<=B;
				BD2<=BD;
				if k3>0 and k4>0 then
					case s2 is
					when 0 => t:= PIN + SHR(B,CV(k1,5))+ SHR(B,CV(k2,5));
					when 1 => t := PIN + SHR(B,CV(k1,5)) - SHR(B,CV(k2,5));
					when 2 => t := PIN - SHR(B,CV(k1,5))+ SHR(B,CV(k2,5));
					when others=>t:=PIN-SHR(Bd,CV(k1,5))-SHR(B,CV(k2,5));
					end case;
				elsif k3>0 and k4=0 then
					case s2 is
						when 0|1 => t := PIN + SHR(B,CV(k1,5));
						when others => t := PIN - SHR(B,CV(k1,5));
					end case;
				elsif k3=0 then
					--t <= PIN;
				end if;

				if k1>0 and k2>0 then
					case s1 is
					when 0 => p <= t + SHR(B,CV(k3,5)) + SHR(B,CV(k4,5));
					when 1 => p <= t + SHR(B,CV(k3,5)) - SHR(B,CV(k4,5));
					when 2 => p <= t - SHR(B,CV(k3,5)) + SHR(B,CV(k4,5));
					when others=>p<=t- SHR(B,CV(k3,5)) - SHR(B,CV(k4,5));
					end case;
				elsif k1>0 and k2=0 then
					case s1 is
						when 0|1 => p <= t + SHR(B,CV(k3,5));
						when others => p <= t - SHR(B,CV(k3,5));
					end case;
				elsif k1=0 then
					p <= t;
				end if;
			end if;
		end if;
```

```
        end process;
                BO <= BD2;
end synt;



--------------------------------------------------------------------------------
-- Title      : FIR1
-- Design     : FIR1
--------------------------------------------------------------------------------
--
-- File       : FIR2.vhd
-- Generated  : Tue Feb 20 10:18:11 2018
-- From       : interface description file
-- By         : Itf2Vhdl ver. 1.20
--
--------------------------------------------------------------------------------
--
-- Description :   FIR contains 2 multiplier-free parts before DSP48s
--
--------------------------------------------------------------------------------
--in Spartan6 slx75 only 48 stages fit
--           slx45 only 32

-- 8+8 stagesSum + Del
-- 914LUT 313 CLBs 1639 Tg + 17 DSP48   3.739 ns  nb1=16
-- 947LUT 368 CLBs 1472 Tg + 17 DSP48   3.727 ns  nb1=12
-- 955LUT 302 CLBs 1668 Tg + 17 DSP48   3.66 ns   retiming
-- 837LUT 280 CLBs 1545 Tg + 17 DSP48   3.782 ns  all+++0
-- 743LUT 273 CLBs 1526 Tg + 17 DSP48   3.884ns  all++00
-- 724LUT 270 CLBs 1547 Tg + 17 DSP48   3.32ns   all+0+0
-- 785LUT 265 CLBs 1549 Tg + 17 DSP48   3.145ns  all+0+0 retiming


library IEEE;
use IEEE.STD_LOGIC_1164.all, IEEE.STD_LOGIC_arith.all,IEEE.STD_LOGIC_signed.all;
entity FIR3 is
        port(
                CLK : in STD_LOGIC;
                RST : in STD_LOGIC;
                DI : in STD_LOGIC_VECTOR(15 downto 0);
                DO : out STD_LOGIC_VECTOR(15 downto 0)
                );
end FIR3;

architecture synt of FIR3 is
        component STX is
                generic(coef: integer:= 30000);
                port(
                        CLK : in STD_LOGIC;
                        RST : in STD_LOGIC;
```

```
                B   : in STD_LOGIC_VECTOR(15 downto 0);
                PIN : in STD_LOGIC_VECTOR(39 downto 0);
                P   : out STD_LOGIC_VECTOR(39 downto 0);
                BO  : out STD_LOGIC_VECTOR(15 downto 0)
                );
end component ;

component STS2 is
        generic(nb: natural:=16;
                k1: natural:=1;
                k2: natural:=3;
                k3: natural:=5;
                k4: natural:=7;
                s1: integer:=0;         -- + +
                s2: integer:=3 -- - -
                );
        port(
                CLK : in STD_LOGIC;
                RST : in STD_LOGIC;
                B   : in STD_LOGIC_VECTOR(15 downto 0);
                PIN : in STD_LOGIC_VECTOR(nb-1 downto 0);
                P   : out STD_LOGIC_VECTOR(nb-1 downto 0);
                BO  : out STD_LOGIC_VECTOR(15 downto 0)
                );
end component ;

component DELAYL is
generic(nd: natural:=13*2;
        nb:natural:=16);
port(
        CLK : in STD_LOGIC;
        DI : in STD_LOGIC_VECTOR(nb-1 downto 0);
        DO : out STD_LOGIC_VECTOR(nb-1 downto 0)
        );
end component ;

constant nst: natural:= 33;--+16-1;
constant n1: natural:=8;
constant nb1: natural:= 16;
constant nb2: natural:= 16;

type Tcoef is array (1 to nst-n1) of integer; -- -n1
--0,000000000100110, 10011011, 1110101, -1101101, -110111110,-111111111
--10000,1110110000,1011001101,1001001110, -11000000000,
--10.13.14,   8.10.-13.-15, 8.-11.13.15,  -8.11.13.-15, -6.9.14, -6.15,
--11,    5.-9.11,
constant cf:Tcoef:=(
--16#26#, 16#9B#, 16#75#, 16#FF93#, 16#FE42#, 16#FE01#, 16#10#, 16#3B0#,
16#59A#, 16#24E#, 16#FA00#, 16#F2BB#, 16#F594#, 16#7E8#, 16#258D#,
16#4154#,    16#4CAF#,
```

```
        16#4154#, 16#258D#, 16#7E8#, 16#F594#, 16#F2BB#, 16#FA00#, 16#24E#,
16#59A#,
        16#3B0#, 16#10#, 16#FE01#, 16#FE42#, 16#FF93#, 16#75#, 16#9B#, 16#26#--,

        );

        --LPF  100011, 1111100,101101100,100010100,101001010,1011100111,101110
        -- 35,-124,-364,-276,330,743,46,-1211,-1045,1152,2615,152,-4392,-3992,5809,19768,
        --26418,
        --19768,5809,-3992,-4392,152,2615,1152,-1045,-1211,46,743,330,-276,-364,-124,35

        --

        --HB LPF 35  1110110,         101101,101111010,1011000000,10011000011
        -- 76,0,-180,0,378,0,-704,0,1219,0,-2038,0,3447,0,-6497,0,20705,
        --32767,
        --20705,0,-6497,0,3447,0,-2038,0,1219,0,-704,0,378,0,-180,0,76


        --HPF  10011, 1001110,111010, 110110,11011111,11111111,1000,111011000,
        --constant cf2:Tcoef:=(
        ---19,-78,-58,54,223,255,-8,-472,-717,-295,768,1699,1334,-1012,-4807,-8362,
        --22953,
        ---8362,-4807,-1012,1334,1699,768,-295,-717,-472,-8,255,223,54 ,-58,-78,-19
        --);
        --Hilbert         1001100, 10110100, 101111010,10011000010,11111110110
        -- -76, 0,-180,0,-378,0,-704,0 ,-1218,0,-2038,0,-3447,0,-6497,0,-20705,0,
        -- 20705,0,6497,0,3447,0,2038,0,1218,0,704,0,378,0,180,0,76

        --diff  101010,1110000,11000001,100111111, 111110001,1011100100,10000101000
        -- 42,-112,193,-319,497,-740,1064,-1488,2041,-2764,3719,-5023,6906,-9904,15643,-
32264,
        --0,32264,-15643,9904,-6906,5023,-3719,2764,-2041,1488,-1064,740,-497,319,-
193,112,-42

        type Tc is array (1 to n1) of integer;
        constant k1a1:Tc:= (6, 4,4,4,2, 2,7,1);
        constant k2a1:Tc:= (9, 6,7,7,5,11,0,5);
        constant k3a1:Tc:= (10,9,9,9,10,0,0,7);
        constant k4a1:Tc:= (0,11,11,11,0,0,0,0);
        constant s1a1:Tc:= (0,0,1,2,2,2,0,1);
        constant s2a1:Tc:= (0,3,0,1,0,0,0,0);

        constant k1a2:Tc:= (6, 4,4,4,2, 2,7,1);
        constant k2a2:Tc:= (9, 6,7,7,5,11,0,5);
        constant k3a2:Tc:= (10,9,9,9,10,0,0,7);
        constant k4a2:Tc:= (0,11,11,11,0,0,0,0);
        constant s1a2:Tc:= (0,0,1,2,2,2,0,1);
        constant s2a2:Tc:= (0,3,0,1,0,0,0,0);
```

```
        type Tsign16 is array (0 to nst) of STD_LOGIC_VECTOR(15 downto 0);
        type Tsign40 is array (0 to nst) of STD_LOGIC_VECTOR(39 downto 0);
        type Tsignnb1 is array (0 to n1) of STD_LOGIC_VECTOR(nb1-1 downto 0);
        type Tsignnb2 is array (0 to nst) of STD_LOGIC_VECTOR(nb2-1 downto 0);

        signal bd: Tsign16;
        signal pd: Tsign40;
        signal pd1:Tsignnb1;
        signal pd2:Tsignnb2;
        signal pdd1:STD_LOGIC_VECTOR(nb1-1 downto 0);
        signal pdd2:STD_LOGIC_VECTOR(39 downto 0);
        signal DOi:STD_LOGIC_VECTOR(16 downto 0);
        constant zz:STD_LOGIC_VECTOR(39 downto 0):=(others=>'0');

begin
        pd1(0)<=(others=>'0');
        bd(0)<= DI;


        stageSM1:for i in 0 to n1-1 generate
                U1: STS2 generic map(nb=>nb1,
                        k1 =>k1a1(i+1),
                        k2=>k2a1(i+1),
                        k3=>k3a1(i+1),
                        k4=>k4a1(i+1),
                        s1=>s1a1(i+1),
                        s2=>s2a1(i+1)
                        )
                port map(CLK, RST,
                        B =>bd(i),
                        PIN => pd1(i),
                        P   => pd1(i+1),
                        BO  => bd(i+1)
                        );

        end generate;

        DELN:DELAYL generic map(nst-n1-n1,nb1)
        port map(CLK,
        DI=>pd1(n1),
        DO=>pdd1);

        pd(n1)<=(others=>'0');
        stagesDSP: for i in 0 to nst-1-n1-n1 generate        --
                Ui: STX generic map(cf(i+1))
                port map(CLK,RST,
                        B => bd(i+n1),
                        PIN => pd(i+n1),
                        P   => pd(i+1+n1),
                        BO      => bd(i+1+n1));
```

```
        end generate;


        DELN2:DELAYL generic map(n1,40)
        port map(CLK,
        DI=>pd(nst-n1),
        DO=>pdd2
        );


                pd2(nst-n1)<=SXT(pdd1,nb2);

        stageSM2:for i in 0 to n1-1 generate
                U1: STS2 generic map(nb=>nb2,
                        k1 =>k1a2(n1-i),
                        k2=>k2a2(n1-i),
                        k3=>k3a2(n1-i),
                        k4=>k4a2(n1-i),
                        s1=>s1a2(n1-i),
                        s2=>s2a2(n1-i)
                        )
                port map(CLK, RST,
                        B =>bd(i+nst-n1),
                        PIN => pd2(i+nst-n1),
                        P   => pd2(i+1+nst-n1),
                        BO  => bd(i+1+nst-n1)
                        );

        end generate;

        process(CLK,RST) begin
                if CLK'event and CLK='1' then
                        if RST='1' then
                                DOi<=(others=>'0');
                        else
                                DOi<= pd2(nst)(nb2-1 downto 3) + pdd2(39 downto 23);
                        end if;
                        end if;
                        end process;

        DO<= DOi(16 downto 1);

end synt;
```

APPENDIX 2

Copies of publications

Міжнародна конференція "Безпека, Відмовостійкість, Інтелект" (ICSFTI2018)10 – 12 травня 2018 року, Київ.

УДК 004.383

Анатолій Сергієнко, Кадір Сафван Хусейн, Анастасія Сергієнко

ФІЛЬТРИ ЗІ СКІНЧЕННОЮ ХАРАКТЕРИСТИКОЮ З МІНІМІЗОВАНИМИ АПАРАТНИМИ ВИТРАТАМИ

Anatoliy Sergiyenko, Quadir Safwan Husein, Anastasia Serhienko

MINIMIZED HARDWARE FIR FILTER DESIGN

Розглядається розробка паралельних нерекурсивних фільтрів, які реалізуються в програмованих логічних інтегральних схемах. Новий метод полягає в тому, що блоки множення на коефіцієнти, які мають невелику амплітуду, заміняються на блоки постійної пам’яті, які зберігають кратні значення цих коефіцієнтів. За рахунок цього зменшуються апаратні витрати на реалізацію фільтра та збільшується його пропускна спроможність.

Ключові слова: ПЛІС, нерекурсивний фільтр, конвеєр, .

Рис.: 3. Табл.:1. Бібл.: 4.

The development of the parallel finite impulse response filters for the FPGA implementation is considered. A new method consists in substituting the multipliers to the small coefficients to the constant coefficient multipliers, which store the multiplied values of these coefficients. Due to this, the filter hardware volume is minimized and its throughput is increased.

Key words: FPGA, FIR filter, pipeline.

Fig.: 3. Tabl.:1. Bibl.: 4.

Вступ. Програмовані логічні інтегральні схеми (ПЛІС) широко використовуються для високошвидкісної обробки цифрових сигналів. Архітектура ПЛІС адаптована до реалізації фільтрів зі скінченною імпульсною характеристикою (СІХ). Для цього, наприклад, ПЛІС фірми Xilinx містять блоки DSP48, кожен з яких призначений для розрахунку одного ступеня СІХ-фільтра в конвеєрному режимі. Але порядок такого фільтра обмежений об'ємом ПЛІС та кількістю блоків DSP48 в одному стовпці елементів мікросхеми. Як результат, порядок СІХ-фільтра, який генерується утилітою Xilinx Coregen для ПЛІС Spartan-6, обмежена діапазоном від 8 до 48 [1]. Крім того, якщо ПЛІС використовується лише для фільтрації, то в ній неефективного використовується решта програмованих ресурсів, таких як конфігуровані логічні блоки та їх логічні таблиці (ЛТ).

Існує багато методів реалізації СІХ-фільтрів, які не мають блоків множення. Більшість із них використовують схеми множення на константу (СМК), які мають мінімізований об’єм апаратного забезпечення. Вони широко використовуються у ПЛІС протягом десятиліть. При цьому такі фільтри зовсім не застосовують апаратні блоки множення, які входят у склад блоків DSP48 [2,3,4].

У роботі запропоновано новий підхід до розробки CIX-фільтру, який використовує як апаратні блоки множення, так і СМК. Він забезпечує як збільшення порядку фільтра, так і мінімізовані апаратні витрати, а також підтримує високу пропускну здатність фільтра.

Структура CIX-фільтра. CIX-фільтр, якиф призначений для реалізації у ПЛІС фірми Xilinx FPGA, має відому систолічну структуру [1]. Граф синхронних потоків даних (ГСПД) систолічної структури фільтра k-го порядку показана на рис.1, а його умовне зображення — на рис.1, б. Тут x_i, y_i є вхідними та вихідними даними, кола, трикутники та товсті відрізки представляють собою додавання, множення на коефіцієнт і затримку на один такт, відповідно. Цей граф відображається у відповідну паралельну конвеєрну структуру з максимальною тактовою частотою за допомогою відображення один до одного.

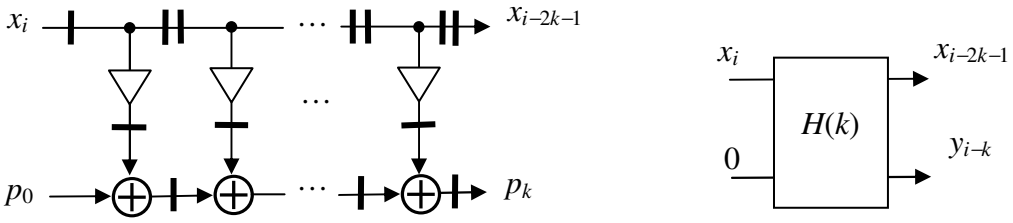


Рис.1. ГСПД систолічного CIX-фільтра (а) та його умовне зображення (б)

а) б)

Набір коефіцієнтів CIX-фільтра n-го порядку формує його імпульсну характеристику $H(n)$. Пропонується реалізувати CIX-фільтр з трьох блоків, які обчислюють згортку з лівою $H_1(n_l)$, середнєю $H_2(n_m)$ та правою $H_3(n_r)$ підмножинами коефіцієнтів, $n = n_l + n_m + n_r$. Відповідний модернізований ГСПД показано на рис.2. У більшості випадків розрядність коефіцієнтів $H_1(n_l)$ та $H_3(n_r)$ набагато менше, ніж коефіцієнтів $H_2(n_m)$. Тому бажано реалізувати частину фільтра, позначену на рис.2 як $H_2(n_m)$, використовуючи блоки DSP48, а решту фільтра —з використанням СМК.

Третій блок фільтра, який стоїть останнім у графі на рис.2, повинен мати підвищену розрядність проміжних результатів, щоб зберегти низький рівень помилок від їх усікання. Це може суттєво знизити ефективність фільтра, оскільки багаторозрядний суматор на основі ЛТ є значно повільнішим, ніж відповідний суматор у блоці DSP48.

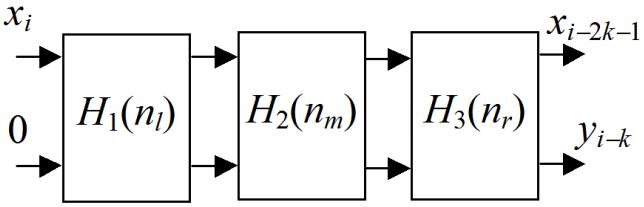


Рис. 2. Модернізований ГСПД фільтра

Щоб звести до мінімуму цей фактор, запропоновано удосконалений ГСПД, який показано на рис.3. Відповідний удосконалений фільтр містить дві частини $H_1(n_l)$ та $H_3(n_r)$ з однаковою розрядністю, яка набагато менша, ніж розрядність операндів блоку DSP48. Результат фільтрації формується в окремому суматорі зі збільшеною розрядністю. Як результат, розроблений CIX-фільтр містить лише n_m блоків DSP48, яких може бути набагато менше, ніж порядок фільтра n.

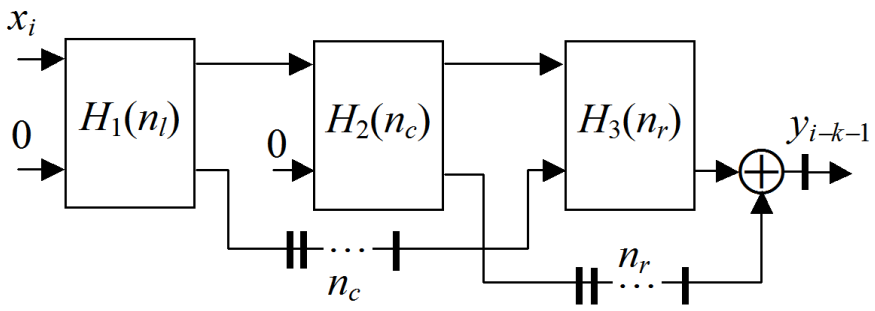


Рис. 3. Удосконалений ГСПД фільтра

Експериментальні результати. Для оцінки ефективності НІХ-фільтрів з запропонованою структурою, було випробувано три реалізації фільтрів нижніх частот порядку n = 35, коефіцієнти, вхідні та вихідні дані яких мають розрядність 16. При цьому СМК були побудовані з використанням представлення коефіцієнтів у канонічній двійковій системі числення [4], причому дерево суматорів СМК має не більше п'яти вхідних складових. За цими умовами, перший та третій блоки фільтрів (див. рис. 2, 3) мають як мінімізовані апаратні витрати, так і високу пропускну здатність.

ГСПД, представлені на рис. 1-3 були описані мовою VHDL. Потім проекти фільтрів були синтезовані для ПЛІС Xilinx Spartan-6. Результати синтезу для деяких фільтрів низьких частот показані в таблиці 1. Щоб визначити інтегральну характеристику апаратних витрат Q, вважалося, що для реалізації 18-розрядного блоку множення, що входить у склад блоку DSP48, потрібно 200 ЛТ.

Таблиця 1.

Прараметри СІХ-фільтрів

Структура фільтра	Апаратні витрати				Максимальна частота f_C, МГц	f_C/Q
	ЛТ	тригерів	DSP48	**Q**		
Систолічна, рис.1	0	0	33	33	390	11.8
Модернізована, рис.2	772	1538	17	20,9	146	7.0
Удосконалена, рис.3	914	1639	17	21,6	267	12.4

Аналіз таблиці показує, що структура фільтра на базі лише блоків DSP48 забезпечує максимальну тактову частоту дискретизації f_C за рахунок збільшених еквівалентних апаратних витрат Q. Модернізована структура має меншу величину f_C, але забезпечує мінімальний обсяг апаратного забезпечення. І удосконалена структура має менші апаратні витрати ніж систолічна, а її відношення пропускної здатності до апаратурних витрат f_C/Q досягає максимуму.

Висновки. Запропоновано модернізацію та удосконалення систолічної структури цифрового фільтра зі скінченною імпульсною характеристикою, яка реалізована у ПЛІС. Модернізація полягає в тому, що в структурі використовуються як універсальні апаратні блоки множення, так і блоки монження на коефіцієнт, які реалізовані на основі суматорів. Нова структура фільрів забезпечує як оптимальне відношення пропускної здатності до апаратних витрат, так і збільшення максимального порядку реалізованих фільтрів. Одержані фільтри описані мовою VHDL і мають приблизно удвічі менше число блоків множення, ніж фільтри, які згенеровані програмою Xilinx Coregen.

Список використаних джерел

1. Spartan-6 FPGA DSP48A1 Slice User Guide. UG389 (v1.2) / Xilinx Inc. / May 29, 2014. —46 p.
2. Meyer-Baese. U. Digital Signal Processing with Field Programmable Gate Arrays. Springer, 4-th Ed. 2014. — 930 p.
3. Sergyienko A. FIR filter soft core generator / A. Sergyienko, V. Vasylienko, O. Maslennikow // Prace IV Konferencji Krajowej „Reprogramowalne uklady cyfrowe", RUC'2001. –Szczecin, Poland. 2001. —P. 167-172.
4. Kumm M. Multiple Constant Multiplication Optimizations for Field Programmable Gate Arrays. Springer, 2016. — 206 p.

***Vinogradov Ju. N.*[1], *Sergiyenko A. M.*[1], *Quadir S. H.*[1]**
[1]*Computer Engineering Department of NTUU "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine*

Minimized FIR Filter Design Implemented in FPGA

Introduction. The field programmable gate array (FPGA) is widely used for the high-speed digital signal processing. FPGA architecture is adapted for the effective finite impulse response (FIR) filter implementation. For this purpose the Xilinx FPGAs contain the DSP48 blocks, each of them is intended for a single filter stage calculation in the pipelined mode. But the filter length is limited by the FPGA volume and by the number of the DSP48 blocks in a single column of the chip. So, the length of the FIR filter which is generated by the Xilinx Coregen tool for the Spartan-6 devices is limited by the numbers from 8 to 48 [1]. And the more this number the more expensive the chip is. Besides, when FPGA is used only for the filtering, then the configured hardware like look-up tables (LUTs), registers is underloaded and is used ineffectively.

There are many methods of the multiplier-less FIR filter implementation. The most of them use the constant coefficient multipliers (CCMs), which have the minimized hardware volume. They are widely used in FPGAs for decades. But no one of them uses the hardware multipliers like the DSP48 block [2–4]. In the presentation, a new method of the FIR filter design is proposed which utilizes both hardware multipliers and CCMs. It provides both the increased filter length and minimized hardware providing the high throughput.

Filter structure. The FIR filter, which is intended for the Xilinx FPGA implementation, has the well-known systolic structure [1]. The synchronous dataflow graph (SDF) of the k-staged systolic structure is illustrated by Fig. 1,a, and its symbol is shown in Fig. 1,b. Here x_i, y_i are the input and output data, the circle, triangle, and bar represent addition, multiplication to the coefficient and delay to a single clock cycle, respectively. This graph is mapped to the respective structure by the one-to-one mapping providing the high pipelined computations with the maximized clock frequency. So, such SDF represents the filter structure as well.

The FIR filter coefficient set is named as an impulse response $H(n)$. It is proposed to implement the FIR filter from three units, which calculate the convolution to the left $H_1(n_l)$, middle $H_2(n_m)$, and right $H_3(n_r)$ subsets of the coefficients, $n = n_l + n_m + n_r$. The respective SDF is shown in Fig.2.

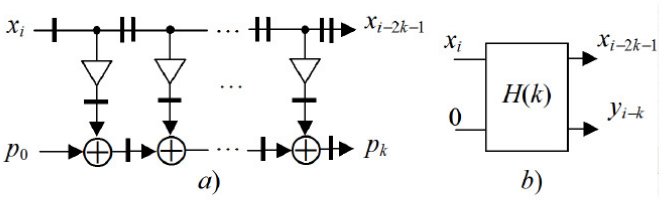


Figure 1. SDF of the systolic FIR filter (a), and its symbol (b)

In the most of cases, the bit width of the coefficients $H_1(n_l)$, and $H_3(n_r)$ is much less, than the bit width of the coefficients $H_2(n_m)$. Therefore, it is favorably to implement the filter part, marked in Fig. 2 as $H_2(n_m)$, using the DSP48 blocks, and others parts using CCMs.In the last situation, the filter stage can be implemented as the small tree of adders with the small bit width. Such a tree can have both high throughput and small hardware volume.

Due to this filter schema, the third filter unit must have the increased adder bit width to preserve the low level of the truncation errors. This can decrease the filter performance dramatically, because the long adder, based on LUTs, is much slower than the respective adder in the DSP48 block. To minimize this disadvantage, the improved SDF is proposed, which is illustrated by Fig.3.

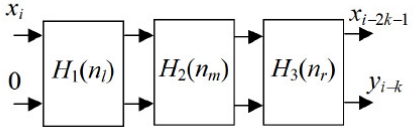


Figure 2. Modified SDF

The proposed filter contains two parts $H_1(n_l)$, and $H_3(n_r)$ with the equal bit width, which is much less than the bit width of the DSP48 unit. The filter result is formed in the separate adder with the increased bit width. As a result, the designed FIR filter contains n_m DSP48 blocks, which can

be much less than the filter length n.

Experimental results. The design of FIR filters of the order of n = 35 for the input, output data, and coefficient bit width 16 was considered. The CCMs were built using the canonical signed digit coefficient representation [2–4], and the adder tree. The FIR filter design experience shows that about a half of the 16-bit filter coefficients can be represented in the canonical signed digit form using only four nonzero digits. For example, the filter stage, which performs the multiplication of x to 0.000000101101110_2 and addition of the sum of products p_i from the previous filter stage, looks like the following:

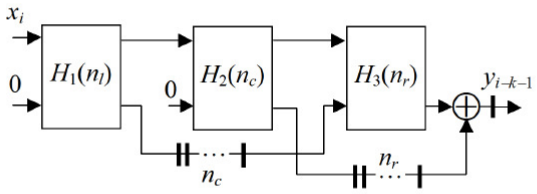


Figure 3. Improved SDF

$p_{i+1} = p_i + 2^{-6}x - 2^{-8}x - 2^{-11}x - 2^{-14}x$.

The modern FPGAs, which have the six input LUTs, support the design of the high-speed three input adders. So, such CCM with the addition of the sum of products is implemented in the two staged pipelined network of LUTs providing the maximum clock frequency. Under these conditions, the first and third filter units (see Fig. 2, 3) have both minimized hardware volume and high throughput.

The DFGs in Fig 1-3 were described by the VHDL language. Then, the filter projects were synthesized for the Xilinx Spartan-6 FPGA. The results of the synthesis for some low pass filter of the 33-rd order are shown in Table 1. To estimate the whole hardware volume Q, it was considered, that an 18-bit multiplier in the FPGA requires 200 equivalent LUTs for its implementation.

The Table 1 analysis shows that the filter structure based on DSP48 blocks, provides the maximum sampling clock frequency f_C at the cost of the hardware volume. The second combined structure gives lower value f_C, but it provides much less hardware volume and the better ratio of throughput to hardware volume f_C/Q.

Table 1. Parameters of the FIR filters implemented in Xilinx Spartan-6 FPGA

Structure	LUTs	Registers	DSP48	Q	f_C, MHz	f_C/Q
Systolic	0	0	33	33	390	11.8
Modified	772	1538	17	20.9	146	7.0
Improved	914	1639	17	21.6	267	12.4

Conclusion. A method of the systolic type FIR filter design for FPGA is proposed, which utilizes both built-in multipliers and application specific constant coefficient multipliers.The method provides both the increased filter length and minimized hardware volume providing the high throughput. The derived filter structure has approximately in two times fewer hardware multipliers than the usual filter generated by the tool like Xilinx Coregen.The resulting filter is represented by the VHDL program and therefore, it can be effectively implemented in FPGA of any type and manufacturer.

References. **1.** "Spartan-6 FPGA DSP48A1 Slice User Guide". UG389, v1.2, Xilinx Inc. May 29, 2014. 46 p. **2.** U. Meyer-Baese, "Digital Signal Processing with Field Programmable Gate Arrays". Springer, 4-th Ed. 2014, 930 p. **3.** A. Sergyienko, V. Vasylienko, O. Maslennikow, "FIR filter soft core generator Prace IV Konferencji Krajowej „Reprogramowalne uklady cyfrowe", RUC'2001. Szczecin, Poland. 2001. pp. 167-172. **4.** M. Kumm, "Multiple Constant Multiplication Optimizations for Field Programmable Gate Arrays". Springer, 2016. 206 p.