А.В. Строгонов

РЕАЛИЗАЦИЯ АЛГОРИТМОВ ЦИФРОВОЙ ОБРАБОТКИ СИГНАЛОВ В БАЗИСЕ ПРОГРАММИРУЕМЫХ ЛОГИЧЕСКИХ ИНТЕГРАЛЬНЫХ СХЕМ

УЧЕБНОЕ ПОСОБИЕ

Издание шестое, исправленное и дополненное

САНКТ ПЕТЕРБУРГ МОСКВА КРАСНОДАР
2024

УДК 004.383
ББК 32.85я73

С 86

Строгонов А. В. Реализация алгоритмов цифровой обработки сигналов в базисе программируемых логических интегральных схем : учебное пособие для вузов / А. В. Строгонов. – 6-е изд., испр. и доп. – Санкт-Петербург : Лань, 2024. – 436 с. – Текст : непосредственный.

ISBN 978-5-507-49591-7

В учебном пособии рассматривается реализация алгоритмов цифровой обработки сигналов в базисе ПЛИС. Даются практические примеры проектирования цифровых устройств с использованием системы визуально-имитационного моделирования MATLAB/Simulink, САПР ПЛИС Altera Quartus II и Xilinx ISE Design Suite.

Издание соответствует требованиям Федеральных государственных образовательных стандартов высшего образования по направлениям подготовки бакалавров и магистров 11.03.04 «Электроника и наноэлектроника», 28.03.02 «Наноинженерия», 11.04.04 «Электроника и наноэлектроника», 11.04.01 «Радиотехника» а также специалистов по 11.05.01 «Радиоэлектронные системы и комплексы».

УДК 004.383
ББК 32.85я73

Рецензенты:

Е. Н. БОРМОНТОВ – доктор физико-математических наук, профессор, зав. кафедрой физики полупроводников и микроэлектроники Воронежского государственного университета;

В. Б. СТЕШЕНКО – кандидат технических наук, доцент, зам. генерального конструктора по электронно-компонентной базе АО «Российские космические системы».

Обложка
Ю. В. ГРИГОРЬЕВА

ВВЕДЕНИЕ

ПЛИС – цифровые БИС высокой степени интеграции, имеющие программируемую пользователем внутреннюю структуру и предназначенные для реализации сложных цифровых устройств. Использование ПЛИС и САПР позволяет в сжатые сроки создавать конкурентоспособные устройства и системы, удовлетворяющие жестким требованиям по производительности, энергопотреблению, надежности, массо-габаритным параметрам, стоимости. Обработка сигналов может осуществляться с помощью различных технических средств. В последнее десятилетие лидирующее положение занимает цифровая обработка сигналов (ЦОС), которая по сравнению с аналоговой имеет следующие преимущества: малую чувствительность к параметрам окружающей среды, простоту перепрограммирования и переносимость алгоритмов. Одной из распространённых операций ЦОС является фильтрация. Вид импульсной характеристики цифрового фильтра определяет их деление на фильтры с конечной импульсной характеристикой (КИХ-фильтры) и с бесконечной импульсной характеристикой (БИХ-фильтры).

Широкое применение цифровых КИХ-фильтров вызвано тем, что свойства их хорошо исследованы. Использование особенностей архитектуры ПЛИС позволяет проектировать компактные и быстрые КИХ-фильтры с использованием так называемой распределённой арифметики.

Учебное пособие предназначено для студентов уже освоивших математические методы цифровой обработки сигналов. В первой главе рассматриваются основы двоичной арифметики, представление чисел со знаком, арифметические операции. Основное внимание уделено расчету параметров КИХ-фильтра в системе M*atlab*/Simulink (пакет Signal Processing, среда FDATool). Обсуждаются вопросы проектирования с учетом эффектов квантования, возникающих

при переходе от имитационной модели КИХ-фильтра, разработанной в системе M*atlab*/Simulink, к функциональной, реализованной в САПР ПЛИС Quartus II компании Altera. Рассматриваются различные методики представления коэффициентов КИХ-фильтра в формате с фиксированной запятой с использованием программных пакетов Xilinx System Generator IDS и Altera DSP Builder системы M*atlab*/Simulink, которые обеспечивают высокоуровневое представление систем цифровой обработки сигналов, абстрагированное от конкретной аппаратной платформы, с возможностью автоматической компиляции в базис ПЛИС.

Во второй главе демонстрируются различные варианты реализации параллельных КИХ-фильтров на мегафункциях САПР Altera Quartus II и с применением Altera DSP Builder.

В третьей главе рассматривается проектирование последовательных КИХ-фильтров как на умножителях так и с использованием блоков умножения и накопления (MAC-блоков) в САПР Altera Quartus II и в системе визуально-имитационного моделирования M*atlab*/Simulink с использованием MAC-блоков в Altera DSP Builder.

В четвертой главе затрагиваются вопросы проектирования высокопроизводительных КИХ-фильтров на последовательной и параллельной распределенных арифметиках, учитывающих архитектурные особенности ПЛИС с использованием САПР Quartus II и Altera DSP Builder.

В пятой главе рассматривается проектирование систолических КИХ-фильтров в базисе ПЛИС с использованием системы цифрового моделирования ModelSim-Altera.

В шестой главе уделено внимание методологии объектно-ориентированного проектирования с использованием программного пакета расширения Xilinx System Generator в системе визуально-имитационного моделирования M*atlab*/Simulink.

1. ОПРЕДЕЛЕНИЕ КОЭФФИЦИЕНТОВ ФИЛЬТРОВ

1.1. Двоичная арифметика

Положительные двоичные числа можно представить только одним способом, а отрицательные двоичные числа – тремя способами. В табл. 1.1 приведены в качестве примера десятичные числа со знаком и их эквивалентные представления в прямом, обратном и дополнительном двоичном коде.

Прямой код. Знак – старший значащий разряд (СЗР) указывает знак (0 – положительный, 1 – отрицательный). Остальные разряды отражают величину, представляющую положительное число:

```
Знак
СЗР     МЗР
 0  110   1   = + 13
 1  110   1   = - 13
```

Это представление чисел удобно для умножения и деления, но при операциях сложения и вычитания нецелесообразно и поэтому используется редко.

В ЭВМ положительные числа представляются в прямом коде, а отрицательные – в виде дополнений, т. е. путем сдвига по числовой оси исходного числа на некоторую константу. Если $z-$ положительное число, то $-z$ представляется в виде $K-z$, где K таково, что разрядность положительна. Обратный код отличается от дополнительного только выбором значения K.

Дополнение до единицы (обратный код) – отрицательные числа получаются путем инверсии всех разрядов их положительных эквивалентов. Старший значащий разряд указывает знак ($0-$ положительный, $1-$ отрицательный).

Таблица 1.1

Представление чисел в прямом, обратном и дополнительном четырехразрядном двоичном коде

ДЧ со знаком	Прямой код	Обратный код* (инверсия $\lvert X_{10}\rvert$ и 1 в знаковый разряд)	Дополнительный код** (инверсия $\lvert X_{10}\rvert$, плюс 1 к МЗР и 1 в знаковый разряд)
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
0	0000 1000	0000 1111	0000
–1	1001	1110	1111
–2	1010	1101	1110
–3	1011	1100	1101
–4	1100	1011	1100
–5	1101	1010	1011
–6	1110	1001	1010
–7	1111	1000	1001
–8	–	–	1000

* при суммировании чисел циклический перенос к МЗР;

** при суммировании чисел перенос игнорируется

Пусть X_{10} – десятичное число со знаком, которое необходимо представить в обратном коде. Необходимо найти $n-$ разрядное представление числа X_{10}, включая знак и часть абсолютной величины, которая считается $(n-1)-$ разрядной. Если $X_{10} \geq 0$, то обратный код содержит 0 в старшем, знаковом разряде и обычное двоичное представление X_{10} в остальных

$n-1$ разрядах. Таким образом, для положительных чисел обратный код совпадает с прямым. Если же $X_{10} \le 0$, то знаковый разряд содержит 1, а остальные разряды содержат двоичное представление числа:

$$2^{n-1} - 1 - |X_{10}|.$$

Дополнение до единицы формируется очень просто, однако обладает некоторыми недостатками, среди которых двойное представление нуля (все единицы или нули).

Рассмотрим положительное число +13. Выбрав шестиразрядное представление, включая знак $n = 6$, получим обратный код, равный 001101. Под абсолютную величину числа отводим пять разрядов. Рассмотрим отрицательное число -13_{10}, считая представление шестиразрядным, включая знак. В пятиразрядном представлении $|-13_{10}| = 13_{10} = 01101_2$ и $2^5 - 1_{10} = 31_{10} = 11111_2$, тогда:

$$(2^{6-1} - 1 - 13)_{10} = (11111 - 01101)_2 = 10010_2.$$

Добавив шестой, знаковый, разряд, получим шестиразрядный код для -13_{10}, равный 110010.

Дополнение до двух (дополнительный код). Его труднее сформировать чем дополнение до единицы, но использованием данного кода удается упростить операции сложения и вычитания. Дополнение до двух образуется путем инверсии каждого разряда положительного числа и последующего добавления единицы к МЗР:

Знак	МЗР	
0	110 1	= + 13
1	001 1	= - 13

.

Если $X_{10} \ge 0$, то так же, как для прямого и обратного кодов, имеем 0 в знаковом разряде и обычное двоичное представление числа X_{10} в остальных $n-1$ разрядах. Если же

$X_{10} < 0$, то имеем 1 в знаковом разряде, а в остальных $n-1$ разрядах двоичный эквивалент числа $2^{n-1} - |X_{10}|$.

Рассмотрим число -13_{10}. Представим его в шестиразрядном дополнительном коде. Так как $|-13_{10}| = 13_{10} = 01101_2$ и $2^5_{10} = 32_{10} = 100000_2$, то получим в пятиразрядном представлении:

$$2^{n-1} - |X_{10}| = (2^{6-1} - 13)_{10} = (100000 - 01101)_2 = 10011_2 .$$

Добавляя шестой знаковый разряд, получаем дополнительный код числа -13_{10}, равный 110011. Ноль в дополнительном коде имеет единственное представление.

Сложение положительных чисел происходит непосредственно, но перенос в разряд знака нужно предотвратить и рассматривать как переполнение. Когда складываются два отрицательных числа или отрицательное число с положительным, то работа сумматора зависит от способа представления отрицательного числа. При представлении последних в дополнительном коде сложение осуществляется просто, но необходим дополнительный знаковый разряд, любой перенос за пределы положения знакового разряда просто игнорируется.

```
+14  01110        +7  00111        -4  11100
 -7  11001       -14  10010        -3  11101
-----------      -----------      -----------
 +7  00111        -7  11001        -7  11001
```

Если используется дополнение до единицы, то перенос из знакового разряда должен использоваться как входной перенос к МЗР.

```
+14  01110        +7  00111        -4  11011
 -7  11000       -14  10001        -3  11100
-----------      -----------      -----------
     00110        -7  11000            10111
   +     1                       +         1
-----------                      -----------
 +7  00111                       -7  11000
```

Рассмотрим такое понятие, как «расширение знака». Рассмотрим десятичное число -3_{10} в дополнительном, а число $3_{10}-$ прямом кодах в трехразрядном представлении:

$$-3 \quad 101 \quad (-2^2 + 2^0)$$
$$3 \quad 011 \quad (2^1 + 2^0)$$

в четырехразрядном представлении:

$$1101 \quad (-2^3 + 2^2 + 2^0)$$
$$0011 \quad (2^1 + 2^0).$$

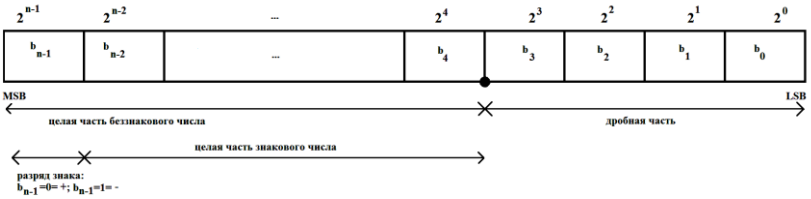
Таким образом, добавление единиц для отрицательных чисел в дополнительном коде и нулей для положительных чисел старше знакового разряда (дублирование знакового разряда) не изменяет представление десятичного числа, этим свойством воспользуемся при проектировании накапливающего сумматора.

1.2. Представление чисел со знаком

Числа с фиксированной запятой характеризуются длиной слова в битах, положением двоичной точки (binary point) и могут быть беззнаковыми или знаковыми. Позиция двоичной точки определяет число разрядов в целой и дробной частях машинного слова. Для представления знаковых чисел (отрицательных и положительных) старший разряд двоичного слова отводится под знак числа (sign bit). При представлении беззнаковых чисел с фиксированной точкой разряд знака отсутствует, и он становится значимым разрядом. Отрицательные числа представляются в дополнительном коде. Данные с фиксированной запятой могут быть следующих типов: целыми (integers); дробными (fractional); обобщёнными (generalize). Обобщённый тип не имеет возможности определить позицию двоичной запятой по умолчанию и требует

явного указания её положения. Этот тип данных специфицируют ufix и sfix форматами.

Ниже показано двоичное число с фиксированной запятой обобщенного типа, где b_i – i-й разряд числа; n – длина двоичного слова в битах; b_{n-1} – старший значимый разряд (MSB); b_0 – младший значимый разряд (LSB); 2^i – вес i-го разряда числа. Двоичная запятая занимает четвёртую позицию от младшего (LSB) разряда числа. При этом длина дробной части числа $m = 4$.



В файле помощи Fixed-Point Blockset системы M*atlab* для представления такого числа применяется следующая формула:

$$V = SQ + B,$$

где V – точное значение действительного десятичного числа; S – наклон; Q – квантованное (двоично-взвешенное) значение целого числа; B – смещение. Наклон S представляется следующим образом: $S = F2^E$, где F – наклон дробной части, является нормализованной величиной $1 \le F \le 2$; E – показатель степени $E = -m$. При проектировании устройств цифровой обработки сигналов принимают $B = 0$ и $F = 1$:

$$V \approx 2^{-m} Q.$$

Квантованное значение Q приближённо представляет истинное значение действительного числа V в виде суммы

произведений весовых коэффициентов b_i на веса 2^i соответствующих двоичных разрядов машинного слова. Для беззнаковых чисел с фиксированной точкой определяется формулой:

$$Q = \sum_{i=0}^{n-1} b_i \times 2^i .$$

Квантованное значение знаковых чисел определяется по формуле:

$$Q = -b_{n-1} \times 2^{n-1} + \sum_{i=0}^{n-2} b_i \times 2^i .$$

Так как целые числа не имеют дробной части $(m = 0)$, то выражение для V имеет вид:

$$V = \sum_{i=0}^{n-1} b_i \times 2^i$$

и для знакового целого числа:

$$V = -b_{n-1} \times 2^{n-1} + \sum_{i=0}^{n-2} b_i \times 2^i .$$

В формате с фиксированной запятой без знака вещественное число V можно считать обозначением полинома:

$$V = S * \left[\sum_{i=0}^{n-1} b_i 2^i \right].$$

Например, двоичное число в дополнительном коде 0011.0101 при длине машинного слова $n = 8$ и $m = 4$ представляет собой беззнаковое (MSB = 0) вещественное число 3.3125:

$$3.3125 = 2^{-4} \left(\begin{array}{l} 0*2^7 + 0*2^6 + 1*2^5 + 1*2^4 + 0*2^3 + \\ +1*2^2 + 0*2^1 + 1*2^0 \end{array} \right).$$

При MSB=1 будем иметь уже другое число -4.6875:

$$-4.6875 = 2^{-4} \left(\begin{array}{l} -1*2^7 + 0*2^6 + 1*2^5 + 1*2^4 + 0*2^3 + \\ +1*2^2 + 0*2^1 + 1*2^0 \end{array} \right).$$

1.3. Расчет параметров КИХ-фильтров с использованием среды FDATool системы визуально-имитационного моделирования M*atlab*/Simulink

Рассмотрим особенности проектирования КИХ-фильтра в системе M*atlab*/Simulink (пакет Signal Processing, среда FDATool).

Главным достоинством среды FDATool от других программ расчета КИХ-фильтров является возможность генерации кода языка VHDL с помощью приложения Simulink HDL Coder. Сгенерированный в автоматическом режиме код языка VHDL может быть использован в системе цифрового моделирования ModelSim (Mentor Graphics HDL simulator).

На рис. 1.1 показана амплитудно-частотная характеристика (АЧХ) КИХ-фильтра. Серые области на рис. 1.1 демонстрируют допуски, превышать границы которых АЧХ фильтра не должна. Исходные данные для расчета КИХ-фильтра: частота взятия отчетов F_S; выбор порядка фильтра n; граница полосы пропускания f_p; граница полосы задерживания (подавления) f_s; неравномерность АЧХ в полосе (полосах) пропускания δ_1 (R_p); минимальное затухание в полосе задерживания δ_2 (R_S).

На практике, как правило, вместо δ_1, δ_2 задают логарифмические величины R_p, R_s, заданные в децибелах:

$$A_p = 20\lg\frac{1+\delta_1}{1-\delta_1}.$$
$$A_a = 20\lg\delta_2$$

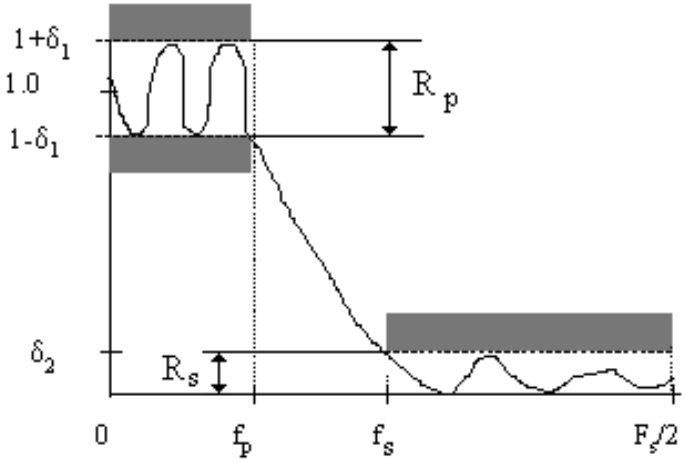


Рис. 1.1. Амплитудно-частотная характеристика фильтра нижних частот

Для построения специализированного устройства, реализующего алгоритм цифровой фильтрации, могут быть использованы регистры, умножители, сумматоры и т. д. – и соответствующее управляющее устройство для управления последовательностью операций. После расчета коэффициентов и выбора структуры фильтра решаются вопросы выбора кодирования чисел (прямой или дополнительный код), способов их представления (с фиксированной или плавающей запятой) и выбора элементной базы.

Исходные данные для расчета КИХ-фильтра нижних частот показаны в табл. 1.1. Пример расчета КИХ-фильтра в среде FDATool показан на рис. 1.2.

Среда FDATool представляет графический интерфейс для расчета фильтров и просмотра их характеристик. На вкладке Design Filter зададим тип синтезируемой АЧХ- фильтр нижних частот, тип фильтра – нерекурсивный (FIR), метод синтеза – метод окон (синтез с использованием весовых функций).

Таблица 1.2

Исходные данные для расчета КИХ-фильтра нижних частот

Параметры фильтра	Значение
Фильтр нижних частот	Low Pass
Частота взятия отсчетов F_s, Гц	48000
Неравномерность АЧХ в полосе пропускания R_p, Дб	1
Минимальное затухание в полосе задерживания R_s, Дб	80
Переходная полоса, Гц	2400
Частота среза, F_c, Гц	9600
Тип окна	Blackman

Среда FDATool поддерживает больше методов синтеза, чем мегаядро Mega Core FIR САПР ПЛИС Quartus II. При проектировании КИХ-фильтра в среде FDATool используются следующие методы: Equiriple – синтез фильтров с равномерными пульсациями АЧХ методом Ремеза; Least-Squares – минимизация среднеквадратичного отклонения АЧХ от заданной и метод окон (Window). В разделе Filter Order задается порядок КИХ-фильтра. Расчет фильтра осуществляется нажатием кнопки Design Filter. На рис. 1.2 показана АЧХ, вычисленная с использованием формата с плавающей (штрих-пунктирная линия) и формата с фиксированной запятой (непрерывная линия).

На рис. 1.3 показана синтезируемая АЧХ (задается комплексный коэффициент передачи $|H(f)|$, определенный в диапазоне частот от нуля до $F_2/2$). Частота среза задается равной $F_c = 9600$ Гц. В методе окон $|H(f)|$ обратное преобразование Фурье этой характеристики дает бесконечную в обе стороны последовательность отсчетов импульсной характеристики.

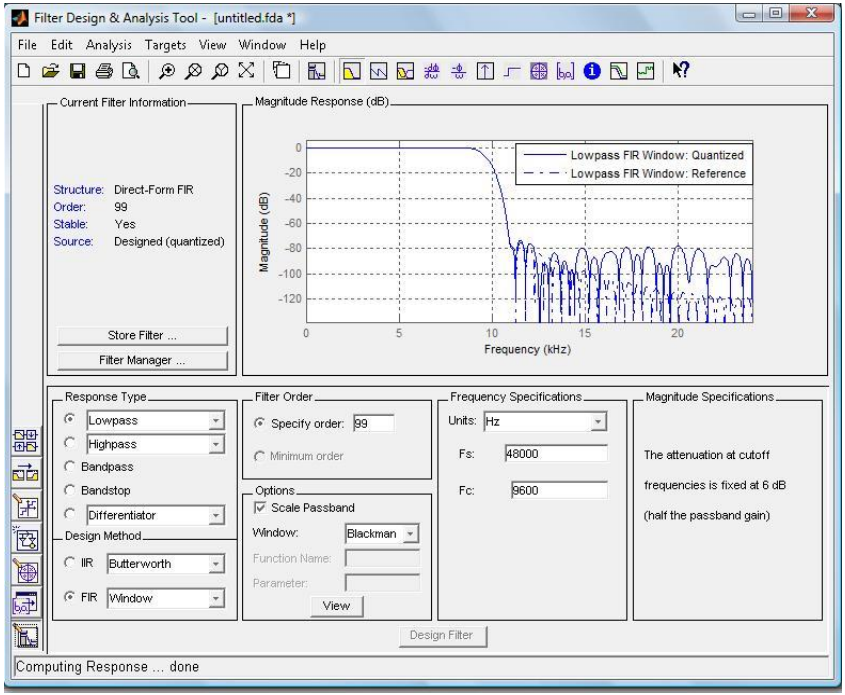


Рис. 1.2. Интерфейс среды FDATool. Пример расчета АЧХ КИХ-фильтра

Для получения КИХ-фильтра заданного порядка эта последовательность усекается путем выбора центрального фрагмента нужной длины. Для ослабления паразитных эффектов в этом методе синтеза усеченная импульсная характеристика умножается на весовую функцию (окно), плавно спадающую к краям.

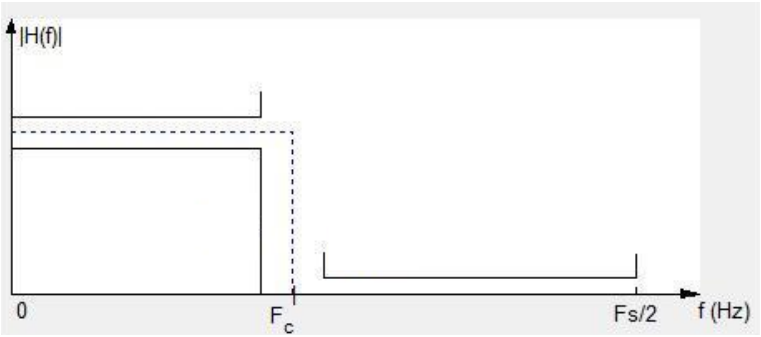


Рис. 1.3. Характеристики синтезируемой АЧХ (окно Blackman) КИХ-фильтра в среде FDATool

Вкладка Realize Model позволяет импортировать спроектированный КИХ-фильтр (модель) в Simulink (рис. 1.2). В этом случае строится модель КИХ-фильтра с использованием базовых элементов (задержка, сумма, коэффициент усиления). Меню Targets опция Generate HDL позволяет сгенерировать код фильтра на языке VHDL. Можно заказать параллельную архитектуру КИХ-фильтра, обладающего высокой производительностью.

1.4. Проектирование квантованных параллельных КИХ-фильтров

Рассмотрим два варианта извлечения кода языка VHDL из моделей расширения Simulink (среда моделирования систем непрерывного и дискретного времени) системы M*atlab* (существует еще вариант извлечения кода непосредственно из алгоритмов M*atlab*). Первый вариант извлечения кода из описания структуры фильтра базовыми элементами расширения Simulink. Второй вариант — из описания структуры фильтра с помощью языка M-файлов Simulink.

На рис. 1.4 показана имитационная модель (верхний уровень иерархии) параллельного симметричного КИХ-фильтра на восемь отводов, взятая из демонстрационного примера Simulink HDL Coder Examples Symmetric FIR Filters. На вход фильтра поступает сигнал с линейной частотной модуляцией (ЛЧМ-сигнал, 2001 отсчет):

```
x_in = cos(2.*pi.*(0:0.001:2).*(1+(0:0.001:2).*75)).'.
```

Рассмотрим используемые параметры сигнала. Частота дискретизации сигнала $F_S = 1$ кГц ([0:1/fs:2] – вектор дискретных значений времени; 1+[0:1/fs:2].*75 – частота импульса). Шаг модельного времени (Sample time) – 1. Ниже показан пример расчета временной функции в системе M*atlab*.

```
>> fs=1e3; % частота дискретизации 1кГц
>> t=0:1/fs:2; % вектор дискретных значений
>> f0=1+[t*75]; % частота импульса
>> s1=cos(2.*pi.*t.*f0); % сконструированный сигнал
>> plot(s1);
```

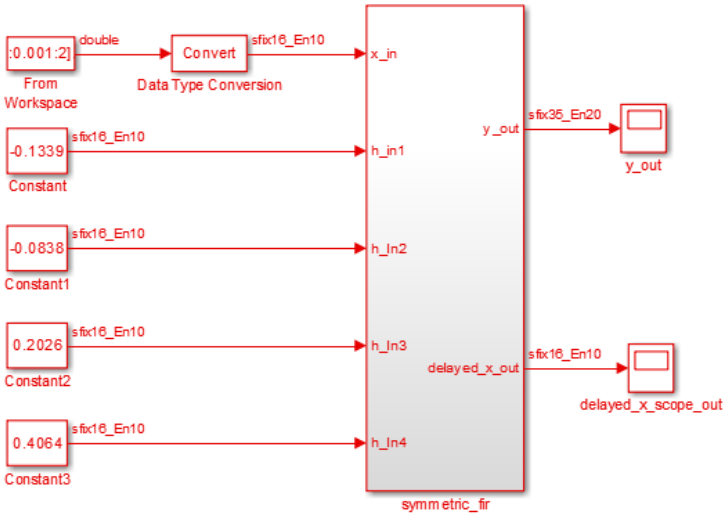
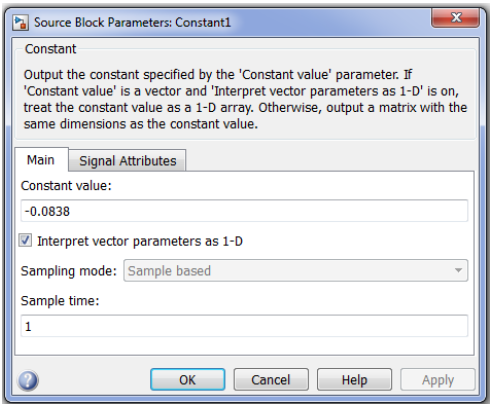


Рис. 1.4. Имитационная модель симметричного КИХ-фильтра на восемь отводов. Верхний уровень иерархии

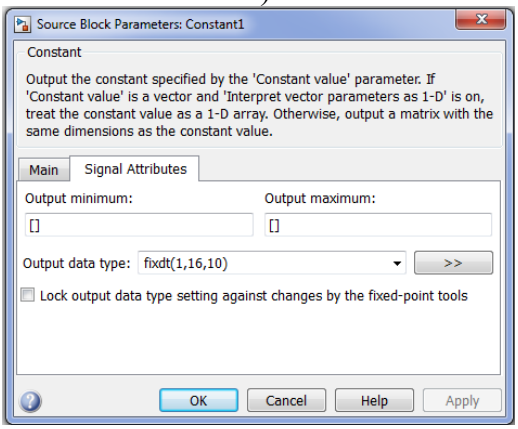
Предположим, что коэффициенты фильтра известны и хранятся в функциональных блоках Constant с одноименными именами Constant–Constant3. Выходные сигналы представляются в формате с фиксированной запятой fixdt(1,16,10): h1 = –0.1339; h2 = –0.0838; h3 = 0.2026; h4 = =0.4064. Преобразовывать значения из формата с плавающей запятой в формат с фиксированной запятой позволяют «автоматизированные мастера», встроенные в функциональные блоки (рис. 1.5).

Коэффициенты фильтра и входной сигнал, подлежащий фильтрации, подвергаются масштабированию путем умножения на масштабный множитель 1024 (2^{10}) в соответствии с выбранным форматом 16.10 (sfix16_En10) и последующему округлению. Например, h1*1024= –137,1136 округляется до целого значения –137D или ff77 в дополнительном коде при длине машинного слова 16 бит).

В шестнадцатеричной системе счисления с учетом знака числа они будут выглядеть следующим образом: ff77, ffaa, 00cf, 01a0. При этом следует помнить, что формат квантования коэффициентов КИХ-фильтра влияет на его частотные и временные характеристики.



а)



б)

Рис. 1.5. Настройка функционального блока Constant с именем Constant1: а) задается вещественное число –0.0838, являющееся одним из коэффициентов фильтра (закладка Main); б) выходной формат представления этого числа fixdt(1,16,10) (закладка Signal Attributes)

Значения входного сигнала, который необходимо профильтровать, изменяются по амплитуде от –1 до +1. Они представлены в формате с фиксированной запятой (sfix16_En10). Функциональный блок Data Tupe Conversion осуществляет преобразование формата double в fixdt(1,16,10).

В дальнейшем необходимо предусмотреть деление коэффициентов и входных значений сигнала на 1024 или профильтрованных значений на 1048576. Например, начальные значения сигнала выглядят следующим образом: 0400, 03ff, 03ff, 03ff, 03ff, 03ff, 03fe. Для перевода в формат double их необходимо разделить на масштабный множитель 1024: 1, 0.9990234375 и т. д.

Рассмотрим имитационную модель КИХ-фильтра на рис. 1.6. После запуска модели над входным сигналом x_in, коэффициентами фильтра, выходными сигналами y_out и delayed_x_scope_out (выход линии задержки) указывается используемый формат.

При использовании арифметики с фиксированной запятой операции сложения не приводят к необходимости округления результатов – они могут лишь вызвать переполнение разрядной сетки. Поэтому выходы с пресумматоров Add-Add3 представляются в формате sfix17_En10 для учета переполнения. Выходы с умножителей Product –Product 3 представляются в формате sfix33_En20, так как умножение чисел с фиксированной запятой приводит к увеличению числа значащих цифр результата по сравнению с сомножителями, которые в данном случае 16- и 17-разрядные, и, следовательно, к необходимости округления. Выходы с сумматоров Add5–Add6 первого уровня дерева сумматоров представляются в формате sfix34_En20, а второго уровня — sfix35_En20. Следовательно, результат фильтрации (сигнал y_out) представляется в формате sfix35_En20 (рис. 1.6).

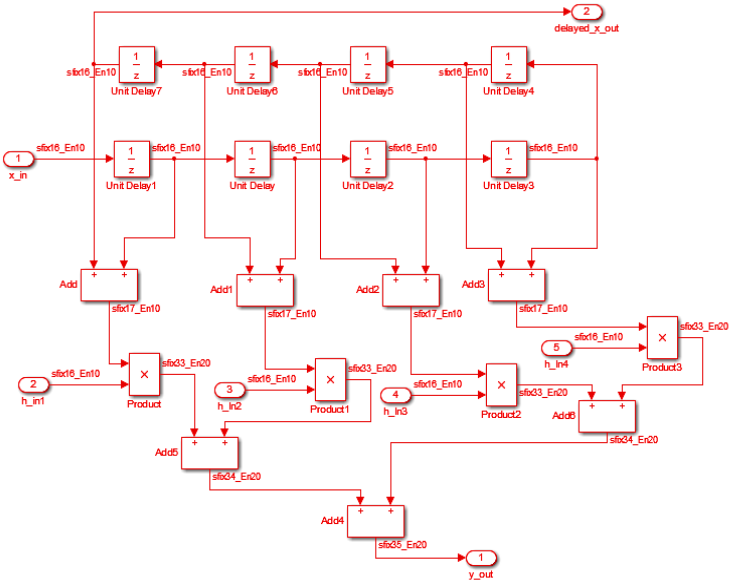


Рис. 1.6. Имитационная модель симметричного КИХ-фильтра на восемь отводов с указанием формата сигналов. Нижний уровень иерархии. Структура фильтра в элементах системы M*atlab*/Simulink

Далее с помощью приложения Simulink HDL Coder системы M*atlab*/Simulink извлечем в автоматическом режиме код языка VHDL КИХ-фильтра. Запуск приложения осуществляется из меню Code/HDL Code/Generate HDL. Предварительно с помощью меню Code/HDL Code/Option необходимо заказать определенные опции и осуществить настройки.

Рассмотрим второй вариант реализации модели. Разработаем имитационную модель симметричного КИХ-фильтра на восемь отводов в формате с фиксированной запятой с использованием fi-объектов и языка М-файлов системы M*atlab* (рис. 1.7 и рис. 1.8). Будем использовать следующий формат для представления десятичных чисел:

a = fi(v, s, w, f),

где *v* – десятичное число; *s* – знак (0 (false) – для чисел без знака и 1 (true) – для чисел со знаком); *w* – размер слова в битах (целая часть числа); *f* – дробная часть числа в битах. Это можно осуществить с использованием следующего формата:

```
a = fi(v, s, w, f, fimath).
% HDL specific fimath
hdl_fm = fimath(...
'RoundMode', 'floor',...
'OverflowMode', 'wrap',...
'ProductMode', 'FullPrecision', 'ProductWordLength', 32,...
'SumMode', 'FullPrecision', 'SumWordLength', 32,...
'CastBeforeSum', true);
```

Данные настройки вычислений в формате с фиксированной запятой приняты в системе Simulink по умолчанию. Можно задать режим округления (Roundmode) – ‘floor’ – округление вниз; реакцию на переполнение (OverflowMode) – ‘wrap’ – перенос, при выходе значения v из допустимого диапазона, «лишние» старшие разряды игнорируются. При выполнении операций умножения (‘ProductMode’) и сложения (SumMode) для повышения точности вычислений (precision) используется машинное слово шириной в 32 бита.

Пример 1 показывает М-файл симметричного КИХ-фильтра на восемь отводов с использованием fi-объектов. На рис. 1.9 показан сигнал до и после фильтрации.

Далее, в автоматическом режиме с помощью приложения Simulink HDL Coder из имитационной модели симметричного КИХ-фильтра на восемь отводов на основе М-файлов и fi-объектов извлекается код языка VHDL и формируется тестбенч для последующего функционального моделирования.

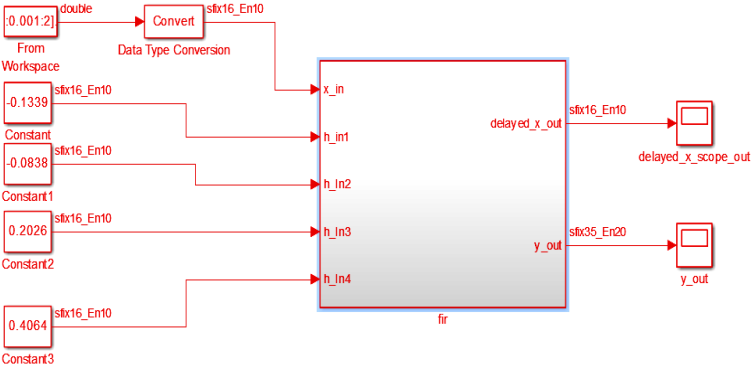


Рис. 1.7. Верхний уровень иерархии имитационной модели симметричного КИХ-фильтра на восемь отводов с использованием М-файла

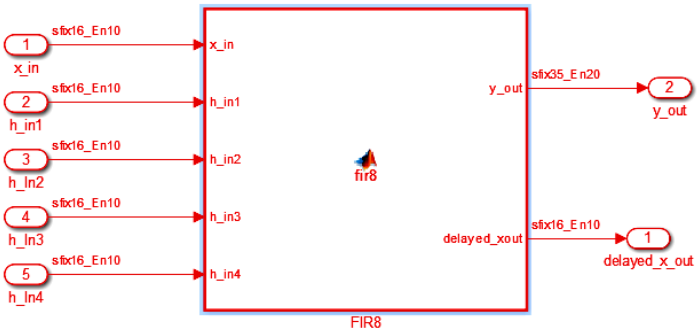
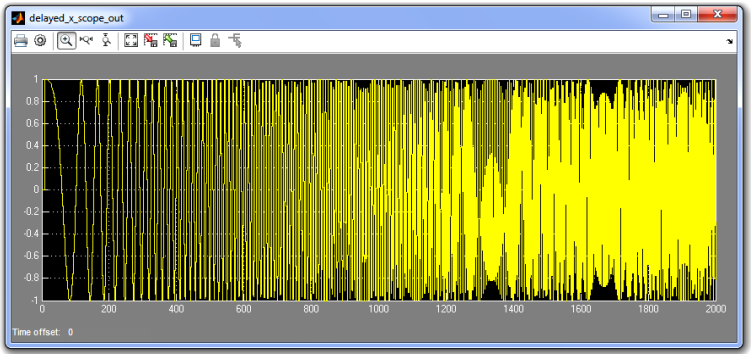


Рис. 1.8. Подсистема симметричного КИХ-фильтра на восемь отводов (подключение входных и выходных портов к М-файлу)

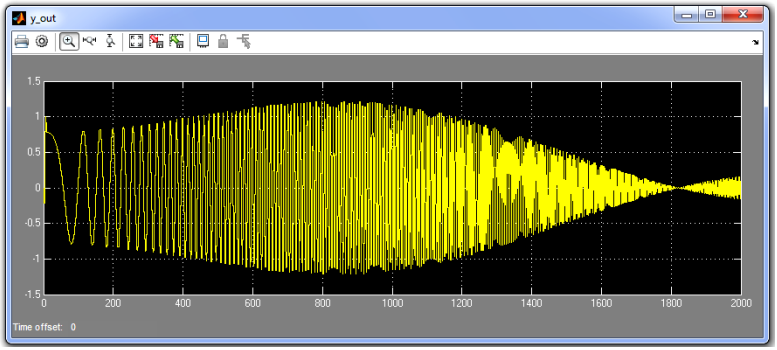
```
%#codegen
function [y_out, delayed_xout] = fir8(x_in, h_in1, h_in2, h_in3, h_in4)
% Symmetric FIR Filter
% HDL specific fimath
hdl_fm = fimath(...
'RoundMode', 'floor',...
```

```
'OverflowMode', 'wrap',...
'ProductMode', 'FullPrecision', 'ProductWordLength', 32,...
'SumMode', 'FullPrecision', 'SumWordLength', 32,...
'CastBeforeSum', true);
% declare and initialize the delay registers
persistent ud1 ud2 ud3 ud4 ud5 ud6 ud7 ud8;
if isempty(ud1)
   ud1 = fi(0, 1, 16, 10, hdl_fm);
   ud2 = fi(0, 1, 16, 10, hdl_fm);
   ud3 = fi(0, 1, 16, 10, hdl_fm);
   ud4 = fi(0, 1, 16, 10, hdl_fm);
   ud5 = fi(0, 1, 16, 10, hdl_fm);
   ud6 = fi(0, 1, 16, 10, hdl_fm);
   ud7 = fi(0, 1, 16, 10, hdl_fm);
   ud8 = fi(0, 1, 16, 10, hdl_fm);
end
% access the previous value of states/registers
a1 = fi(ud1 + ud8, 1, 17, 10, hdl_fm);
a2 = fi(ud2 + ud7, 1, 17, 10, hdl_fm);
a3 = fi(ud3 + ud6, 1, 17, 10, hdl_fm);
a4 = fi(ud4 + ud5, 1, 17, 10, hdl_fm);
% multiplier chain
m1 = fi((h_in1*a1), 1, 33, 20, hdl_fm);
m2 = fi((h_in2*a2), 1, 33, 20, hdl_fm);
m3 = fi((h_in3*a3), 1, 33, 20, hdl_fm);
m4 = fi((h_in4*a4), 1, 33, 20, hdl_fm);
% adder chain
a5 = fi(m1 + m2, 1, 34, 20, hdl_fm);
a6 = fi(m3 + m4, 1, 34, 20, hdl_fm);
% filtered output
y_out = fi(a5 + a6, 1, 35, 20, hdl_fm);
% delayout input signal
delayed_xout = ud8;
% update the delay line
ud8 = ud7;  ud7 = ud6; ud6 = ud5;
ud5 = ud4;
ud4 = ud3;
ud3 = ud2;
ud2 = ud1;
ud1 = fi(x_in, 1, 16, 10, hdl_fm);
end
```

Пример 1. М-файл симметричного КИХ-фильтра на восемь отводов с использованием fi-объектов



а)



б)

Рис. 1.9. Результаты имитационного моделирования:
а) исходный сигнал; б) профильтрованный сигнал

Рассмотрим функциональное моделирование КИХ-фильтра с использованием симулятора ModelSim-Altera Starter Edition. На рис. 1.10 показано моделирование фильтра с использованием тестбенча, полученного с помощью приложения Simulink HDL Coder из имитационной модели симметричного КИХ-фильтра на восемь отводов на основе M-файлов и fi-объектов.

Используя полученный VHDL-код, разработаем функциональную модель в САПР ПЛИС Quartus II (рис. 1.11). Входной сигнал сформируем с помощью векторного редактора (рис. 1.12) в соответствии с примером 4. Проект размещен в

ПЛИС EP2C5AF256A7 серии Cyclone II. На рис. 1.13 показано распределение задействованных ресурсов проекта по кристаллу ПЛИС EP2C5AF256A7. Используются восемь аппаратных умножителей размерностью операндов 9×9. Что равносильно использованию четырех умножителей размерностью операндов 18×18 (табл. 1.3).

Таблица 1.3

Общие сведения по числу задействованных ресурсов ПЛИС Cyclone II EP2C5AF256A7

Логические элементы (Logic Cells, LC)	Триггеры логических элементов (Dedicated logic registers)	Аппаратные умножители размерностью операндов 9×9 (Embedded Multiplier 9-bit elements)	Рабочая частота в наихудшем случае Fmax, МГц
238/4608 (5%)	128/4608 (3%)	8/26 (31%) или 4 умножителя размерностью 18x18	380

Сравнивая рис. 1.10 и рис. 1.12, видим, что функциональная модель КИХ-фильтра на восемь отводов, построенная с использованием кода языка VHDL, извлеченного в автоматическом режиме с помощью приложения Simulink HDL Coder из имитационной модели симметричного КИХ-фильтра на восемь отводов на основе M-файлов и fi-объектов в САПР ПЛИС Quartus II версии 13.0, работает корректно.

В состав HDL Coder входит инструмент под названием HDL Workflow Advisor (помощник по работе с HDL), который автоматизирует программирование ПЛИС фирм Xilinx и Altera. Можно контролировать HDL-архитектуру и реализацию, выделять критические пути и генерировать отчеты об использовании аппаратных ресурсов.

На рис. 1.14 показан программный инструмент HDL Workflow Advisor расширения Simulink. Более подробную

информацию можно получить на сайте m*atlab*.ru. Программный инструмент HDL Workflow Advisor дублирует пункты меню Code/HDL Code/Generate HDL и Code/HDL Code/Option, но позволяет работать на более качественном уровне.

Генерация кода происходит в несколько этапов. Первый этап заключается в выборе стиля проектирования – реализация проекта в базисе заказных БИС/ПЛИС без привязки к конкретному производителю или реализация проекта с выбором отладочной платы и серии ПЛИС фирм Xilinx или Altera. Для проектов в базисе ПЛИС Workflow Advisor обеспечивает такие возможности оптимизации, как площадь-скорость, распределение памяти RAM, конвейеризация, совместное использование ресурсов и развертывание циклов.

Например, в разделе Set Target выбирается отладочная плата Altera DE2-115 Development and Education Board на ПЛИС серии Cyclone IV EP4CE115. В случае выбора стиля проектирования FPGA-in-the-Loop (замкнутый цикл) отлаживать проект возможно непосредственно из Simulink. Проверку на поддержку отладочной платы Workflow Advisor можно осуществить на втором этапе подготовки модели для генерации кода Prepare Model For HDL Code Generation в разделе Chek FPGA-in-the-Loop Compatibility. На третьем этапе осуществляется генерация кода.

Система M*atlab*/Simulink с приложением Simulink HDL Coder может быть эффективно использована для ускорения процесса разработки квантованных КИХ-фильтров в базисе ПЛИС. Автоматически извлеченный VHDL-код фильтра из описания Simulink-модели приводит к использованию встроенных ЦОС-блоков в ПЛИС серии Cyclone II, обеспечивая разработку высокопроизводительных КИХ-фильтров.

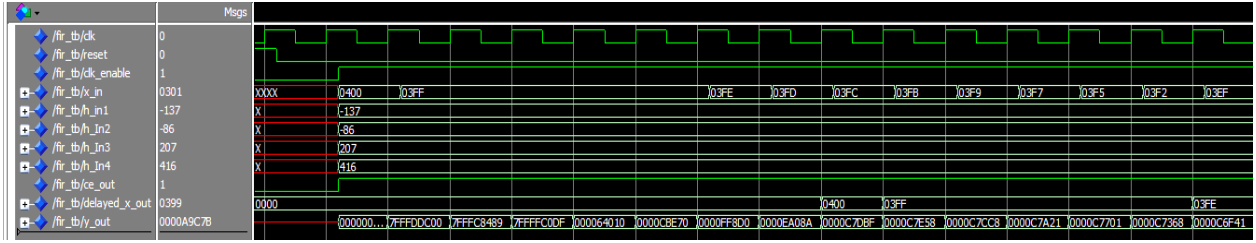


Рис. 1.10. Функциональное моделирование фильтра с использованием тестбенча, полученного с помощью приложения Simulink HDL Coder из имитационной модели симметричного КИХ-фильтра на восемь отводов на основе М-файлов и fi-объектов

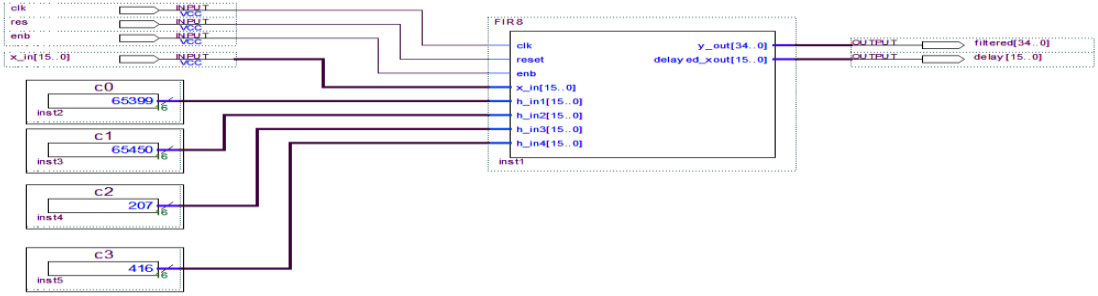


Рис. 1.11. Функциональная модель КИХ-фильтра на восемь отводов, построенная с использованием кода языка VHDL, извлеченного в автоматическом режиме с помощью приложения Simulink HDL Coder из имитационной модели симметричного КИХ-фильтра на восемь отводов на основе М-файлов и fi-объектов в САПР ПЛИС Quartus II версии 13.0

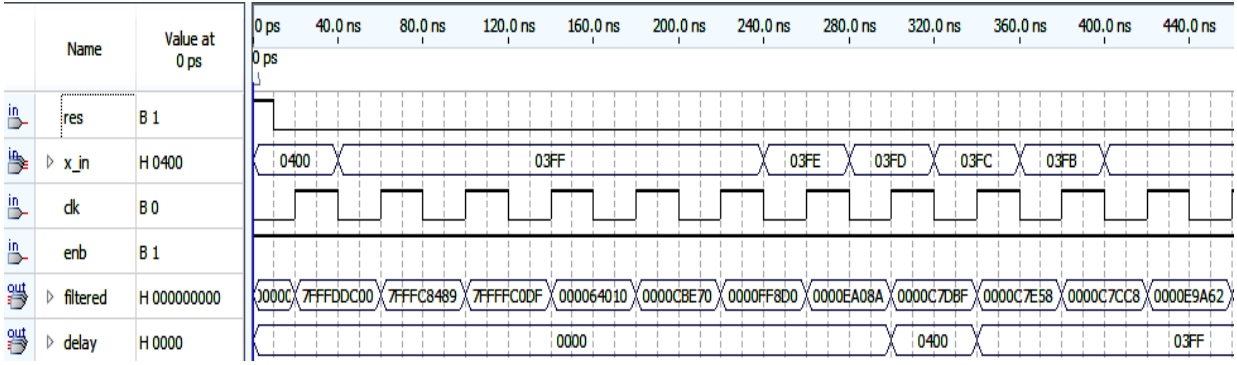


Рис. 1.12. Функциональное моделирование КИХ-фильтра с использованием кода языка VHDL, извлеченного в автоматическом режиме с помощью приложения Simulink HDL Coder из имитационной модели симметричного КИХ-фильтра на восемь отводов на основе М-файлов и fi-объектов

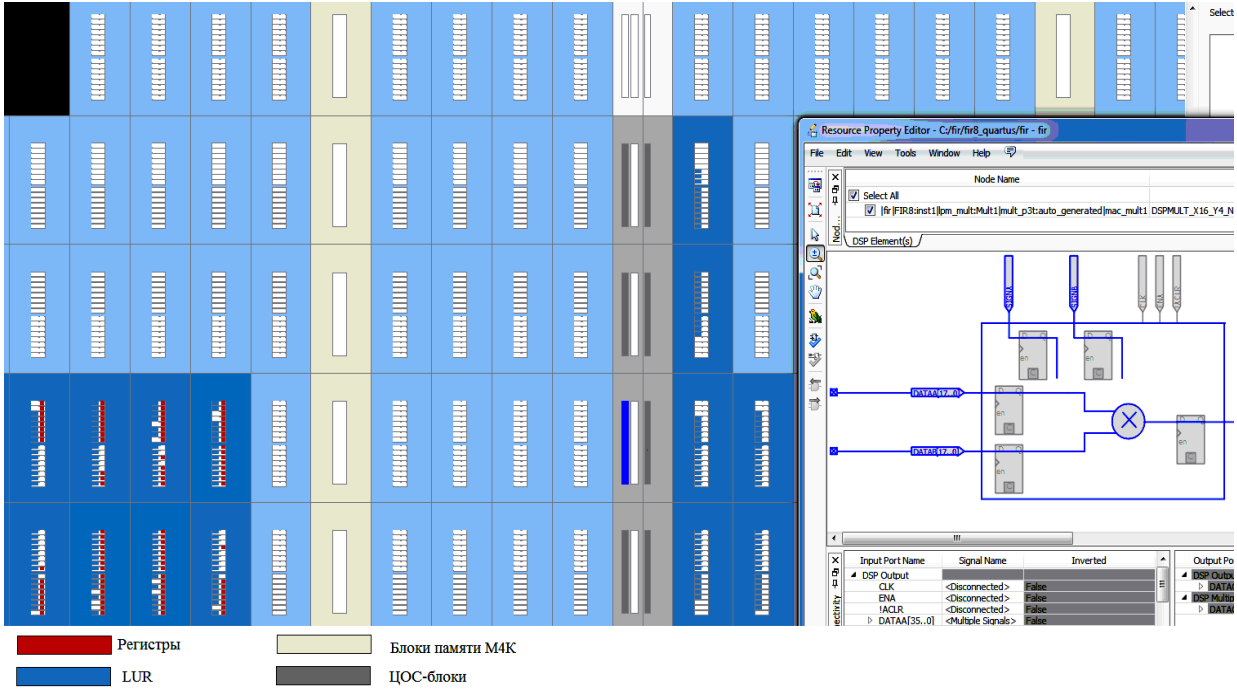


Рис. 1.13. Распределение задействованных ресурсов проекта по кристаллу ПЛИС EP2C5AF256A7

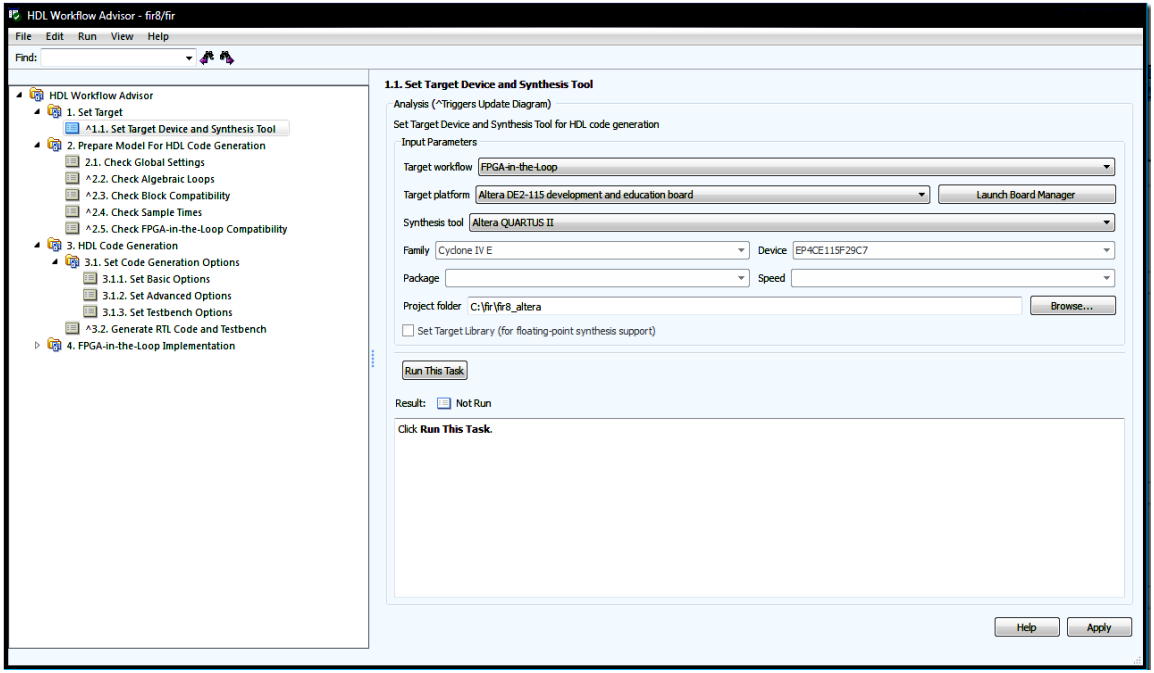


Рис. 1.14. Программный инструмент Workflow Advisor пакета расширения Simulink

1.5. Представление коэффициентов КИХ-фильтра в формате с фиксированной запятой

В системе визуально-имитационного моделирования Matlab/Simulink для реализации алгоритмов ЦОС в ПЛИС могут быть использованы пакеты расширения как от Xilinx System Generator так и от Altera DSP Builder. Программные пакеты расширения Altera DSP Builder и System Generator IDS системы визуально-имитационного моделирования Matlab/Simulink обеспечивают высокоуровневое оптимизированное VHDL-представление проектов с автоматическим компилированием в ПЛИС Xilinx и Altera с последующим созданием испытательных стендов. Однако эти пакеты являются не дружественными по отношению друг к другу, т.к. в них заложены архитектурные особенности ПЛИС разных производителей, но базируются на общих принципах цифровой обработки сигналов используемых в системе Matlab/Simulink.

Рассмотрим методики представления коэффициентов КИХ-фильтра в формате с фиксированной запятой с использованием пакета Xilinx System Generator IDS и Altera DSP Builder.

В качестве примера рассмотрим последовательный КИХ-фильтр с использованием MAC-блока с числом отводов 43 в системе Matlab/Simulink с применением библиотек System Generator IDS. Модель называется mac_df2t_soln.mdl ее можно найти в папке ISE\sysgen\examples\mac_fir.

Модель основана на функциональном блоке n-tap Dual Port Memory MAC FIR Filter из библиотеки Xilinx Reference Blockset/DSP пакета расширения Matlab/Simulink. Для хранения коэффициентов фильтра и входных отсчетов используется блочная память ПЛИС Xilinx (двухпортовое ОЗУ, функциональный блок n-tap Dual Port Memory). Параметры спецификации фильтра следующие: единицы

измерения Гц; частота взятия отчетов $F_S = 44100$ Гц (44.1 кГц); порядок фильтра – автоматический выбор (Minimum Order); граница полосы пропускания $f_{Pass} = 6000$ Гц (6 кГц); граница полосы задерживания (подавления) $f_{Stop} = 7725$ Гц (7.725 кГц); неравномерность АЧХ в полосе пропускания A_{Pass} ($R_p = 1$ дБ); минимальное затухание в полосе задерживания A_{Stop} ($R_S = 48$ дБ); коэффициент плотности сетки частот 16 (Density Factor). Осуществим синтез фильтров с равномерными пульсациями АЧХ в полосах пропускания и задерживания методом Ремеза (Equiripple). По заданной технической спецификации синтезируется фильтр с порядком $n = 42$ (переключатель установлен на минимальный порядок) или 43 коэффициента. Для расчетов коэффициентов КИХ-фильтра используем графическую среду для синтеза и анализа цифровых фильтров FDATool.

КИХ-фильтр с именем MAC Based FIR, приведенный в справочной системе Xilinx System Generator является параметризованным. В System Generator используется специфичный блок, вызывающий среду FDATool (блок помечен фирменным значком X). В отличие от оригинального инструмента FDATool в нем нельзя задавать параметры квантования фильтра, т.к. отсутствуют опции для квантования set quantization parameters. Операция квантования – это представление коэффициентов фильтра фиксированным набором битов.

С помощью специальной функции xlfda_numerator('FDATool') происходит расчет коэффициентов (рис. 1.15). Максимальное значение коэффициента составляет 0.3022, а минимальное -0.067, поэтому коэффициенты фильтра не выходят из диапазона [-1, 1]. Параметры квантования задаются с помощью специальной маски для MAC FIR. Входные данные представляются в формате с фиксированной

запятой FIX_10_8, а коэффициенты как FIX_12_12 (рис. 1.16). Обычно коэффициенты фильтра представляются с помощью двухэлементного вектора M.N (нотация Qm.n), где M – общее число двоичных разрядов, используемое для представления чисел; N – число разрядов дробной части, например 12.12. В Xilinx System Generator принято обозначение FIX_12_12 (рис. 1.16), что обеспечивает приведение коэффициентов к диапазону [-0.5, 0.4998].

Таким образом, в рассмотренном примере процесс квантования входных отсчетов и коэффициентов фильтра осуществляется в автоматическом режиме. Операцию перехода от дробных значений коэффициентов фильтра к целочисленным значениям называют еще нормализацией.

Фрагменты значений выходного сигнала и коэффициентов фильтра в форматах double, FIX_10_8 и FIX_12_12 приведены в таблице.

Вычислим коэффициенты фильтра с помощью функции xlfda_numerator('FDATool') в командной строке системы Matlab и умножим на масштабный коэффициент 4096 (2^12) а затем округлим коэффициенты с помощью функции round. Таким образом, перевод коэффициентов фильтра в формат с фиксированной запятой в простейшем случае означает договоренность умножения коэффициентов на масштабный коэффициент с последующим округлением.

```
y=xlfda_numerator('FDATool');
fir_coef=round(y*4096);
```

–8	–41	–57	–58	–24	26	59	43
–17	–77	–80	–7	97	140	60	–113
–249	–193	124	613	1058	1238	1058	613
124	–193	–249	–113	60	140	97	–7
–80	–77	–17	43	59	26	–24	–58
–57	–41	–8					

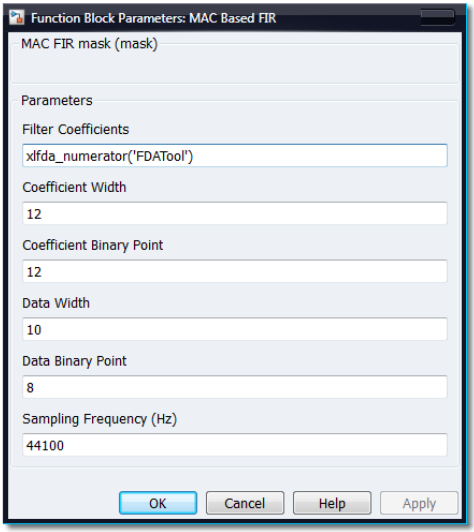


Рис. 1.15. Маска для задания параметров квантования КИХ-фильтра в Xilinx System Generator

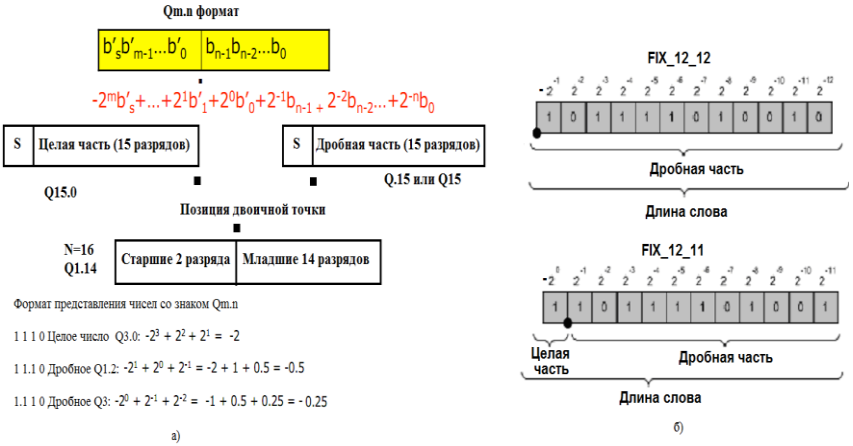


Рис. 1.16. Представление чисел в формате с фиксированной запятой: а) формат Qm.n; б) формат FIX в System Generator

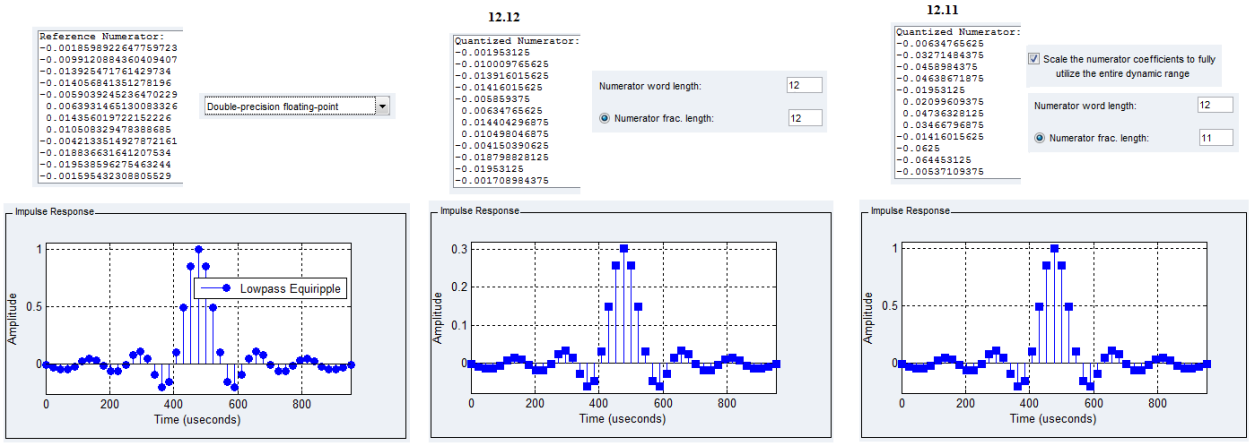
Фрагменты значений выходного сигнала и коэффициентов фильтра в форматах double и FIX при прохождении по структуре фильтра единичного импульса

Значения отклика в формате double (округлены)	Значения отклика в формате FIX_10_8 (значения отклика * масшт. коэф. 256)	Коэффициенты фильтра в формате double (округлены)	Коэффициенты фильтра в формате FIX_12_12 (коэф.фильтра * масшт. коэф. 4096)
–0,00195;	–1	–0.0019	–8
–0,01196;	–3	–0.099	–41
–0,02588;	–7	–0.0139	–57
–0,0400;	–10	–0.0141	–58

Вызов среды FDATool осуществим в командной строке системы Matlab. Посмотрим коэффициенты КИХ-фильтра без учета квантования, выбирается арифметика фильтра Double-precision floating-point (двойная точность для вычислений с плавающей запятой). На рис. 1.17а показаны опорные значения числителя (Reference Numerator) передаточной функции фильтра в формате с плавающей запятой (арифметика фильтра Double-precision floating-point).

Рассмотрим, как с помощью специальной опции для квантования set quantization parameters среды FDATool можно перейти к квантованным коэффициентам фильтра (рис. 1.18). В поле Filter arithmetic выбираем закладку Fixed-point (арифметика фильтра с фиксированной запятой).

Далее настроим параметры квантования так, чтобы они соответствовали модели mac_df2t_soln.mdl Xilinx System Generator, в которой используется формат FIX12.12. Зададим дробную часть числа (Numerator frac. Lenth: 12). Остальные настраиваемые значения во вкладке Coefficients не должны использоваться (без масштабирования). Формат 12.12 (рис. 1.17б) соответствует опции Best-precision fraction lengths (длина дробной части при наилучшей точности представления коэффициентов). На рис. 1.18 показано влияние представления коэффициентов КИХ-фильтра в формате с фиксированной запятой 12.12 на его АЧХ.



а) б) в)

Рис.1.17. Коэффициенты (часть) КИХ-фильтра без учета квантования, выбирается арифметика фильтра Double-precision floating-point (а); б) квантованные, формат 12.12 (приводятся к диапазону [-0.5 0.5)); в) квантованные, формат 12.11 (приводятся к диапазону [-1 1))

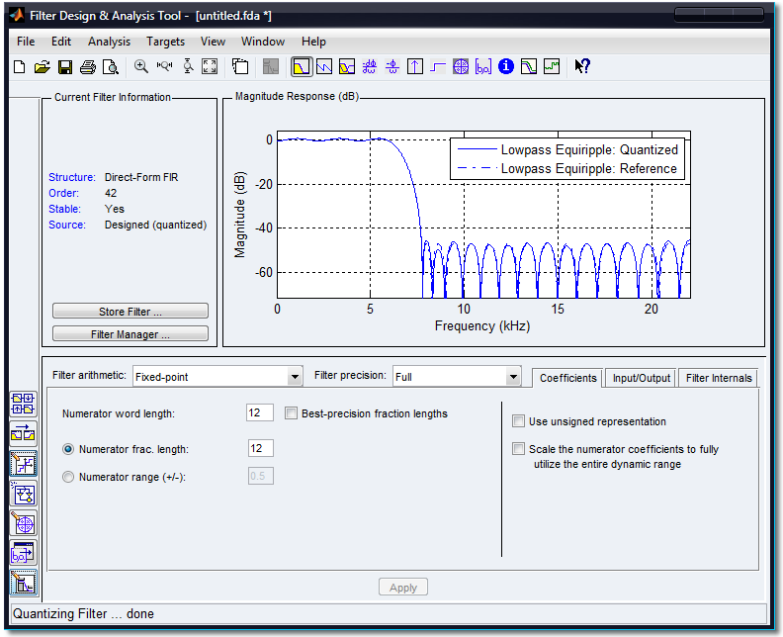


Рис.1.18. Влияние представления коэффициентов фильтра в формате с фиксированной запятой 12.12 на его АЧХ

Для сравнения, двумя способами выгрузим коэффициенты фильтра. Непосредственно в VHDL-файл и в COE-файл с помощью меню Targets Generate HDL и XILINX Cofficient (.COE) File. Ниже приводится фрагмент VHDL-кода фильтра, извлеченный в автоматическом режиме с предварительно заданными параметрами для арифметики с фиксированной запятой в формате 12.12 (пример 1). В системе M*atlab* коэффициенты фильтра в формате с фиксированной запятой при экспорте в VHDL-файл обозначаются как sfix12_En12. В VHDL-коде показано, что коэффициенты фильтра с помощью преобразования типов переводятся в дополнительный код. Например, десятичное целое число со знаком -8, заданное как константа, переводится в 12-разрядное число в дополнительном коде с помощью функции изменения типов to_signed.

```
-- Filter Structure  : Direct-Form FIR
-- Filter Length     : 43
-- Stable            : Yes
-- Linear Phase      : Yes (Type 1)
-- Arithmetic        : fixed
-- Numerator         : s12,12 -> [-5.000000e-01 5.000000e-01)
-- Input             : s10,8 -> [-2 2)
-- Filter Internals  : Full Precision
--   Output          : s23,20 -> [-4 4)  (auto determined)
--   Product         : s21,20 -> [-1 1)  (auto determined)
--   Accumulator     : s23,20 -> [-4 4)  (auto determined)
--   Round Mode      : No rounding
--   Overflow Mode   : No overflow
  -- Constants
CONSTANT coeff1 : signed(11 DOWNTO 0) := to_signed(-8, 12); -- sfix12_En12
CONSTANT coeff2: signed(11 DOWNTO 0) := to_signed(-41, 12); -- sfix12_En12
CONSTANT coeff3  : signed(11 DOWNTO 0) := to_signed(-57, 12); -- sfix12_En12
CONSTANT coeff4: signed(11 DOWNTO 0) := to_signed(-58, 12); -- sfix12_En12
CONSTANT coeff5: signed(11 DOWNTO 0) := to_signed(-24, 12); -- sfix12_En12
CONSTANT coeff6: signed(11 DOWNTO 0) := to_signed(26, 12); -- sfix12_En12
CONSTANT coeff7: signed(11 DOWNTO 0) := to_signed(59, 12); -- sfix12_En12
CONSTANT coeff8: signed(11 DOWNTO 0) := to_signed(43, 12); -- sfix12_En12
CONSTANT coeff9: signed(11 DOWNTO 0) := to_signed(-17, 12); -- sfix12_En12
```

```
CONSTANT coeff10: signed(11 DOWNTO 0) := to_signed(-77, 12); -- sfix12_En12
CONSTANT coeff11: signed(11 DOWNTO 0) := to_signed(-80, 12); -- sfix12_En12
CONSTANT coeff12: signed(11 DOWNTO 0) := to_signed(-7, 12); -- sfix12_En12
CONSTANT coeff13: signed(11 DOWNTO 0) := to_signed(97, 12); -- sfix12_En12
```

Пример 1. Используемая арифметика и фрагмент VHDL-кода коэффициентов фильтра в формате 12.12

Дополнительно выгрузим коэффициенты фильтра для Xilinx System Generator (пример 2).

```
; XILINX CORE Generator(tm)Distributed Arithmetic FIR filter coefficient (.COE) File
; Generated by MATLAB(R) 8.3 and the DSP System Toolbox 8.6.
Radix = 16;
Coefficient_Width = 12;
CoefData =
ff8,fd7,fc7,fc6,fe8,01a,03b,02b,fef,fb3,fb0,ff9,061,08c,03c,f8f,f07,f3f,07c,265,
422,4d6,422,265,07c,f3f,f07,f8f,03c,08c,061,ff9,fb0,fb3,fef,02b,03b,01a,fe8,fc6,
fc7,fd7,ff8;
```

Пример 2. Коэффициенты фильтра представлены в шестнадцатеричном формате с 12-разрядной точностью (coe-файл)

Сравнивая два примера убеждаемся в том, что коэффициенты в coe-файле совпадают с коэффициентами указанными как целые десятичные числа со знаком в VHDL-файле. С той лишь разницей, что coe-файл удобно использовать в генераторе параметризированных ядер XILINX CORE Generator. А полученный VHDL-код может быть использован в любом САПР ПЛИС. Для проверки разделим полученные коэффициенты в формате 12.12 на максимальное положительное значение 4096 и получим импульсную характеристику фильтра.

Предположим, что предварительно неизвестны значения коэффициентов фильтра, и они могут оказаться больше единицы по абсолютному значению. Представим

коэффициенты КИХ-фильтра на 43 отвода в формате с фиксированной запятой 12.11, несмотря на то, что в исходной модели от Xilinx System Generator используется формат FIX12.12.

Дробная часть числа в формате 12.11 (Numerator frac. Lenth) составляет 11 разрядов. Выбирается опция: Scale the numerator coefficients to fuly utilize the dynamic range (масштабирование коэффициентов числителя для того, чтобы полностью использовать динамический диапазон). Опции Best-precision fraction lengths и Use unsigned representation (беззнаковое представление коэффициентов) не используются. Формат входных отсчетов задается в формате 8.8 (ранее было 10.8). После представления коэффициентов в формате с фиксированной запятой 12.11 они приводятся к диапазону [-1, 1) (рис. 1.18в).

Опять для сравнения двумя способами выгрузим коэффициенты фильтра. Непосредственно в VHDL-файл и в COE-файл (пример 3 и 4). Ниже приводится фрагмент VHDL-кода фильтра, извлеченный в автоматическом режиме с предварительно заданными параметрами для арифметики с фиксированной запятой в формате 12.11. Сравнивая два примера убеждаемся в том, что коэффициенты в формате 12.11 в сое-файле совпадают с коэффициентами в VHDL-файле.

```
-- Constants
CONSTANT coeff1: signed(11 DOWNTO 0) := to_signed(-13, 12); -- sfix12_En11
CONSTANT coeff2: signed(11 DOWNTO 0) := to_signed(-67, 12); -- sfix12_En11
CONSTANT coeff3: signed(11 DOWNTO 0) := to_signed(-94, 12); -- sfix12_En11
CONSTANT coeff4: signed(11 DOWNTO 0) := to_signed(-95, 12); -- sfix12_En11
CONSTANT coeff5: signed(11 DOWNTO 0) := to_signed(-40, 12); -- sfix12_En11
CONSTANT coeff6: signed(11 DOWNTO 0) := to_signed(43, 12); -- sfix12_En11
CONSTANT coeff7: signed(11 DOWNTO 0) := to_signed(97, 12); -- sfix12_En11
CONSTANT coeff8: signed(11 DOWNTO 0) := to_signed(71, 12); -- sfix12_En11
CONSTANT coeff9: signed(11 DOWNTO 0) := to_signed(-29, 12); -- sfix12_En11
CONSTANT coeff1: signed(11 DOWNTO 0) := to_signed(-128, 12); -- sfix12_En11
CONSTANT coeff11: signed(11 DOWNTO 0) := to_signed(-132, 12); -- sfix12_En11
CONSTANT coeff12: signed(11 DOWNTO 0) := to_signed(-11, 12); -- sfix12_En11
CONSTANT coeff13: signed(11 DOWNTO 0) := to_signed(160, 12); -- sfix12_En11
```

```
CONSTANT coeff14: signed(11 DOWNTO 0) := to_signed(231, 12); -- sfix12_En11
CONSTANT coeff15: signed(11 DOWNTO 0) := to_signed(99, 12); -- sfix12_En11
CONSTANT coeff16: signed(11 DOWNTO 0) := to_signed(-187, 12); -- sfix12_En11
CONSTANT coeff17: signed(11 DOWNTO 0) := to_signed(-412, 12); -- sfix12_En11
```

Пример 3. Фрагмент VHDL-кода фильтра, извлеченный в автоматическом режиме с предварительно заданными параметрами для арифметики с фиксированной запятой в формате 12.11

```
; XILINX CORE Generator(tm)Distributed Arithmetic FIR filter coefficient
(.COE) File
; Generated by MATLAB(R) 8.3 and the DSP System Toolbox 8.6.
Radix = 16;
Coefficient_Width = 12;
CoefData =
ff3,fbd,fa2,fa1,fd8,02b,061,047,fe3,f80,f7c,ff5,0a0,0e7,063,f45,e64,ec1,0cd,3f6,
6d7,7ff,6d7,3f6,
0cd,ec1,e64,f45,063,0e7,0a0,ff5,f7c,f80,fe3,047,061,02b,fd8,fa1,fa2,fbd,ff3;
```

Пример 4. Коэффициенты фильтра в шестнадцатеричном формате (coe-файл)

Произведем выгрузку m-файла (кода модели фильтра на языке m-файлов) из среды FDATool для расчета коэффициентов фильтра в формате Double-precision floating-point с помощью меню File/Generate MATLAB Code/Filter Design Function (пример 5). Полученный код из файла необходимо скопировать в окно команд (Command Window) для его отработки.

```
Fs = 44100;  % Sampling Frequency
Fpass = 6000;           % Passband Frequency
Fstop = 7725;           % Stopband Frequency
Dpass = 0.057501127785;   % Passband Ripple
Dstop = 0.0039810717055;  % Stopband Attenuation
dens  = 16;
% Calculate the order from the parameters using FIRPMORD.
```

```
[N, Fo, Ao, W] = firpmord([Fpass, Fstop]/(Fs/2), [1 0], [Dpass, Dstop]);
% Calculate the coefficients using the FIRPM function.
b  = firpm(N, Fo, Ao, W, {dens});
Hd = dfilt.dffir(b);
FlCoef = b;
fvtool(Hd,'Analysis','info');figure(gcf)
```

Пример 5. m-файл КИХ-фильтра, созданный с помощью функции firpmord и выгружаемый из среды FDATool без нормализации коэффициентов

Для перевода коэффициентов КИХ-фильтра в формат 12.12 умножим максимально возможное число для 12-разрядного кода на коэффициенты фильтра в формате с плавающей запятой и округлим до ближайшего целого:

```
CoefBitWidth=12;
setka_max = 2^CoefBitWidth;
FpCoef = round(setka_max*FlCoef);
```

Осуществим нормализацию коэффициентов с помощью скрипта модели mac_df2t_soln.mdl (точность представления входных отчетов в этой модели 8 разрядов) из справочной системы Altera DSP Builder и сравним полученные значения коэффициентов со значениями нормализованных коэффициентов, полученными с помощью среды FDATool.

Пример находится по адресу: StFilteringLab/LowPass_SerialFilter. Его можно найти в папке: DesignExamples\Demos\Filters\FilteringLab\.

Приведем коэффициенты фильтра к полному динамическому диапазону по примеру Altera DSP Builder. В этом скрипте имеется возможность задавать точность представления входных отчетов (8 бит) и точность представления коэффициентов (12 бит). Минимальное отрицательное значение при точности представления

коэффициентов 12 бит для дополнительного кода составит 2048 (setka_min=-2^(CoefBitWidth - 1)), а максимальное положительное – 2047 (setka_max = 2^(CoefBitWidth - 1) - 1). Далее найдем минимальное и максимальное значения коэффициентов фильтра FlCoef: -0.0607 и 0.3022. Определим масштабный коэффициент (ScalingFactor1): 6.7739e+03 и умножим на него коэффициенты фильтра с последующим округлением с помощью функции fix. Полученные значения и будут представлять собой коэффициенты КИХ-фильтра в формате с фиксированной запятой.

```
InputBitWidth = 8;
CoefBitWidth = 12;
setka_max = 2^(CoefBitWidth - 1) - 1;
setka_min = -2^(CoefBitWidth - 1);
max_coef=max(FlCoef);
min_coef=min(FlCoef);
ScalingFactor1 = min(setka_max/max(FlCoef),
setka_min/min(FlCoef));
FpCoef1 = fix(ScalingFactor1 * FlCoef);
```

Проверяем, совпадают ли коэффициенты фильтра FpCoef_round, приведенные к полному динамическому диапазону по примеру Altera DSP Builder с нормализованными коэффициентами, полученными с помощью среды FDATool. Для полного совпадения необходимо в примере функцию fix (усечение дробной части числа) заменить на round (округление до ближайшего целого).

```
FpCoef_round=round(ScalingFactor1 * FlCoef);
```

Обнаруживается совпадение. Использование скрипта из модели mac_df2t_soln.mdl Altera DSP Builder приводит к представлению коэффициентов в формате 12.11 с опцией Scale

the numerator coefficients to fuly utilize the dynamic range среды FDATool:

–13 –67 –94 –95 –40 43 97 71 –29 128 –132 –11 160 231 99 –187 –411 –319 205 1013 1750 2047 1750 1013 205 –319 –411 –187 99 231 160 –11 –132 –128 –29 71 97 43 –40 –95 –94 –67 –13.

Для проверки разделим полученные коэффициенты в формате 12.11 на максимальное положительное значение 2047 для 12-разрядного числа в дополнительном коде и получим импульсную характеристику фильтра как на рис. 1.18в.

По рекомендациям Xilinx предлагается способ представления коэффициентов в формате с фиксированной запятой с учетом только нормировки максимального положительного числа (2047) на значение максимального по модулю коэффициента (0.3022) приводящий к такому же результату.

```
ScalingFactor = setka_max/max(abs(FlCoef));
FpCoef = round(setka_max*(FlCoef/max(abs(FlCoef))));
```

Еще раз извлечем m-файл. Но теперь переход к представлению коэффициентов фильтра в формате с фиксированной запятой осуществим с помощью среды FDATool. Создадим квантованный КИХ-фильтр (пример 6), пропустим через него ЛЧМ-сигнал и посмотрим результат фильтрации.

```
% Equiripple Lowpass filter designed using the FIRPM function.
% All frequency values are in Hz.
Fs = 44100;  % Sampling Frequency
Fpass = 6000;           % Passband Frequency
Fstop = 7725;           % Stopband Frequency
Dpass = 0.057501127785;   % Passband Ripple
```

```
Dstop = 0.0039810717055;  % Stopband Attenuation
dens  = 16;               % Density Factor

% Calculate the order from the parameters using FIRPMORD.
[N, Fo, Ao, W] = firpmord([Fpass, Fstop]/(Fs/2), [1 0], [Dpass,
Dstop]);
% Calculate the coefficients using the FIRPM function.
b  = firpm(N, Fo, Ao, W, {dens});
Hd = dfilt.dffir(b);
% Set the arithmetic property.
set(Hd, 'Arithmetic', 'fixed', ...
   'CoeffWordLength', 12, ...
   'CoeffAutoScale', false, ...
   'NumFracLength', 11, ...
   'Signed',        true, ...
   'InputWordLength', 8, ...
   'inputFracLength', 8, ...
   'FilterInternals',  'FullPrecision');
normalize(Hd);
```

Пример 6. m-файл КИХ-фильтра, созданный с помощью функции firpmord и выгружаемый из среды FDATool с нормализацией коэффициентов (формат 12.11)

Далее конструируем ЛЧМ-сигнал и пропускаем его через синтезированный фильтр с целью убедиться в его работоспособности (пример 7):

```
fs=1e3;
t=0:1/fs:3;
f0=1+[t*200];
s1=cos(2.*pi.*t.*f0);
plot(s1);
signal_filtered = filter(Hd, s1);
```

plot(signal_filtered);
Пример 7. ЛЧМ-сигнал и его фильтрация

Итак, если использовать в среде FDATool формат 12.11 с опцией Scale the numerator coefficients to fuly utilize the dynamic range, то получим коэффициенты в формате с фиксированной запятой (а сами коэффициенты при этом будут приведены к диапазону [-1, 1)), совпадающие с коэффициентами фильтра после нормировки, согласно скрипту модели mac_df2t_soln.mdl из справочной системы Altera DSP Builder.

Максимальное положительное значение для 12-разрядного числа в дополнительном коде составит 2047 и определяется по формуле setka_max = 2^(CoefBitWidth - 1) - 1. Масштабный коэффициент (ScalingFactor1) в этом случае составляет 6774.

При представлении коэффициентов в формате 12.12 (приведение к полному динамическому диапазону не требуется, т.к. коэффициенты изначально укладываются в диапазон [-0.5, 0.5)), параметр setka_max составит 4096 и определяется по формуле setka_max = 2^CoefBitWidth. При переводе коэффициентов в формат 12.12 параметр setka_max и будет являться масштабным коэффициентом.

2. ПРОЕКТИРОВАНИЕ ПАРАЛЛЕЛЬНЫХ КИХ-ФИЛЬТРОВ В БАЗИСЕ ПЛИС

На рис. 2.1 показаны структуры фильтров, характерные для реализации в базисе сигнальных цифровых процессоров, а на рис. 2.2 показаны структуры фильтров, характерные для реализации в базисе ПЛИС. В качестве матричных умножителей могут быть использованы параллельные векторные умножители. На рис. 2.3 показан 2-разрядный векторный умножитель с использованием двух идентичных таблиц перекодировки LUT1 и LUT2 для формирования частичных произведений P1(n) и P2(n), которые необходимо сложить с учетом их веса. Каждая LUT образована из четырех LUT логических элементов (ЛЭ) ПЛИС. Результат вычисления P2(n) необходимо сдвинуть на один разряд влево. Такой умножитель может быть использован для структуры фильтра четыре отвода два бита при 2-разрядном представлении коэффициентов. В случае если число отводов останется постоянным (например четыре в случае симметрии коэффициентов фильтра), а разрядность входного сигнала, подлежащего фильтрации, и коэффициентов фильтра составит восемь бит, то уже потребуется восемь LUT, каждая из которых будет содержать восемь LUT ЛЭ. При этом увеличивается и число многоразрядных сумматоров и операций сдвига.

Параллельные КИХ-фильтры, реализованные в базисе ПЛИС, обладая наивысшим быстродействием, позволяют получать результат фильтрации, например, для КИХ-фильтра со структурой 120 отводов 12 бит уже после первого синхроимпульса, последовательные – через 12, а фильтр в базисе ЦОС-процессоров – через 120 синхроимпульсов.

Рисунок 2.5 показывает использование параллельного векторного умножителя четырех 8-разрядных сигналов на четыре 8-разрядные константы в составе КИХ-фильтра на 8 отводов с симметричными коэффициентами. Симметричность коэффициентов позволяет использовать пресумматоры на

выходах линии задержки, что и обеспечит формирование четырех 8-разрядных сигналов.

Рисунок 2.6а демонстрирует число задействованных конфигурируемых логических блоков (КЛБ) для реализации параллельного векторного умножителя в базисе ПЛИС серии XC4000. Для симметричного КИХ-фильтра со структурой 8 отводов 8 бит с точностью представления коэффициентов 8 бит потребуется 122 КЛБ ПЛИС серии XC4000 фирмы Xilinx, при этом обеспечивается быстродействие 50–70 MSPS.

В случае последовательной структуры (рис. 2.6б) применяется одна - единственная LUT для вычисления частичных произведений (рис. 2.7). Такой фильтр обрабатывает только один разряд входного сигнала в течение такта. Последовательно вычисляемые частичные произведения накапливаются в масштабирующем аккумуляторе. После N для несимметричного и N+1 тактов синхроимпульсов для симметричного фильтра на выходе появляется результат, где N – разрядность входного сигнала подлежащего фильтрации. Для обеспечения правильной работы фильтра требуется управляющий автомат. Производительность фильтра определяется как fclk/N для несимметричного и как fclk/N+1 для симметричного фильтра. Рисунок 2.6б показывает, что с ростом числа отводов производительность фильтра остается постоянной, в то время как у фильтра на базе ЦОС-процессора с использованием MAC-блоков начинает резко падать при числе отводов более 16.

Умножители сигналов играют ключевую роль в проектировании высокопроизводительных цифровых фильтров.

Покажем различные варианты реализации КИХ-фильтров с использованием умножителей на мегафункциях ALTMULT_ACCUM, ALTMULT_ADD и ALTMEMMULT САПР Quartus II компании Altera в базисе ПЛИС, а затем сосредоточим внимание на реализации умножения методом правого сдвига с накоплением, применяемого для разработки масштабирующего аккумулятора.

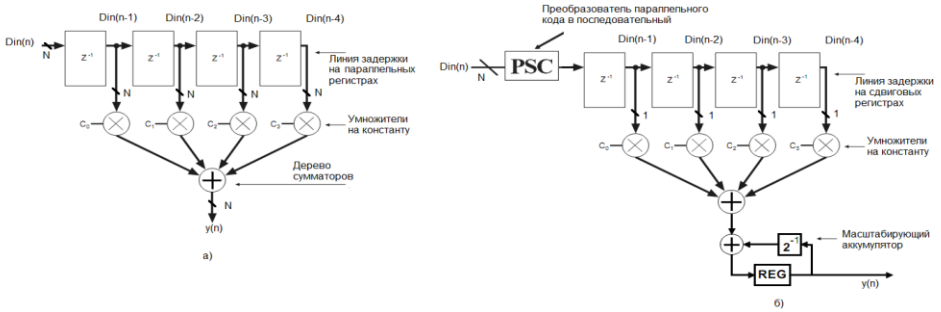


Рис. 2.1. Параллельный (а) и последовательный фильтры (б) на четыре отвода для реализации в базисе цифровых сигнальных процессоров

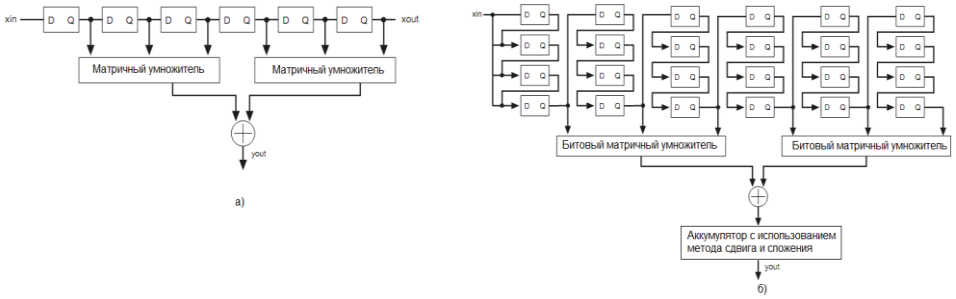


Рис. 2.2. Обобщенное представление структур КИХ-фильтров:
а) параллельных; б) последовательных

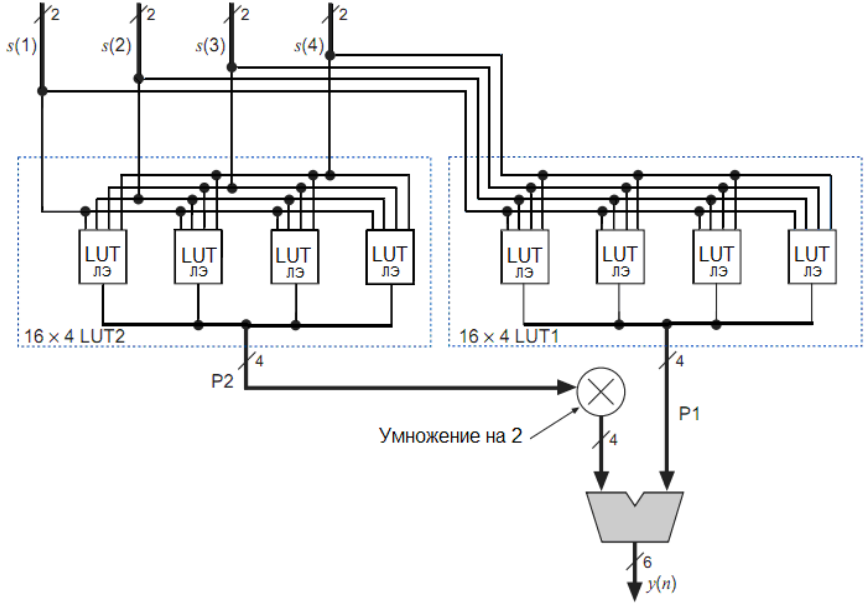


Рис. 2.3. Параллельный векторный умножитель четырех 2-разрядных сигналов на четыре 2-разрядные константы с использованием LUT ЛЭ в ПЛИС серии FLEX

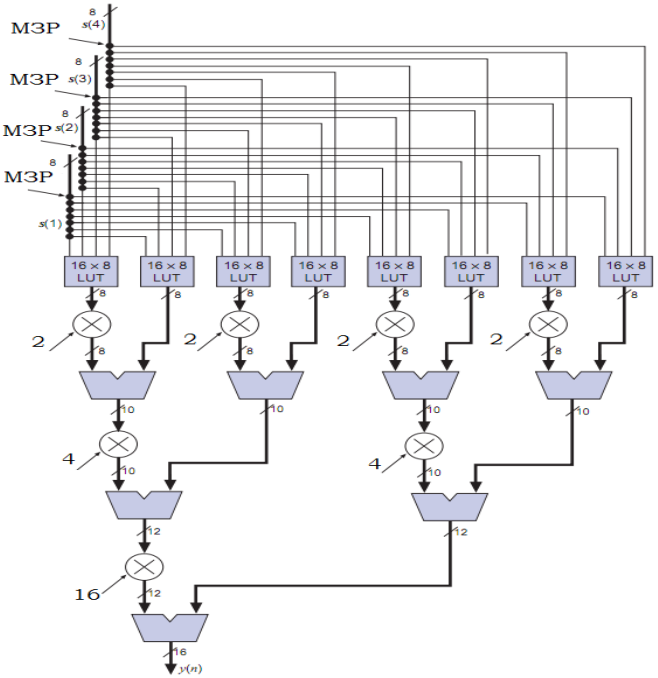


Рис. 2.4. Параллельный векторный умножитель четырех 8-разрядных сигналов на четыре 8-разрядные константы с использованием LUT ЛЭ в ПЛИС серии FLEX

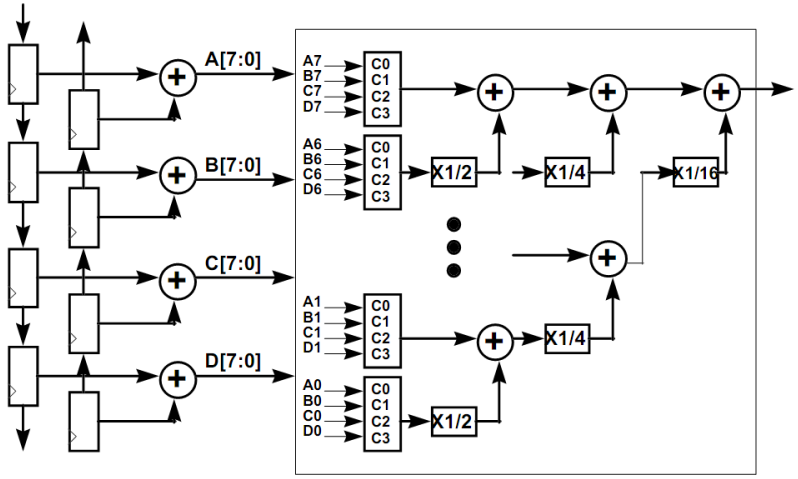
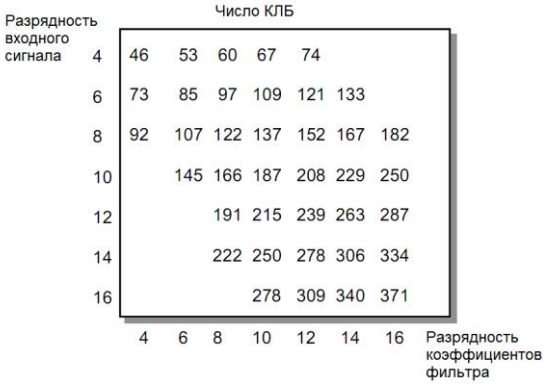
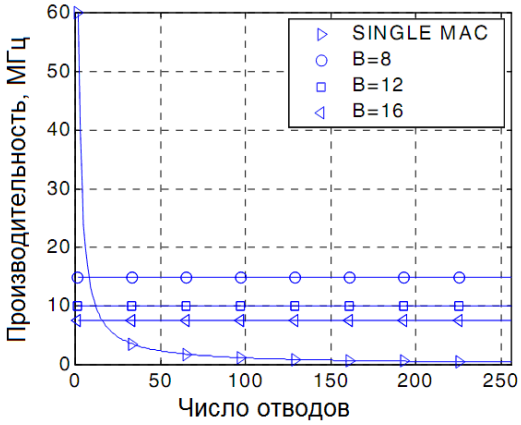


Рис. 2.5. Использование параллельного векторного умножителя четырех 8-разрядных сигналов на четыре 8-разрядные константы в составе КИХ-фильтра на восемь отводов с симметричными коэффициентами в базисе ПЛИС XC4000, построенного с использованием параллельной распределенной арифметики



а)



б)

Рис. 2.6. Число задействованных ресурсов для реализации параллельного векторного умножителя в базисе ПЛИС XC4000 (а) и производительность фильтра в зависимости от числа отводов при частоте тактирования 120 МГц (б)

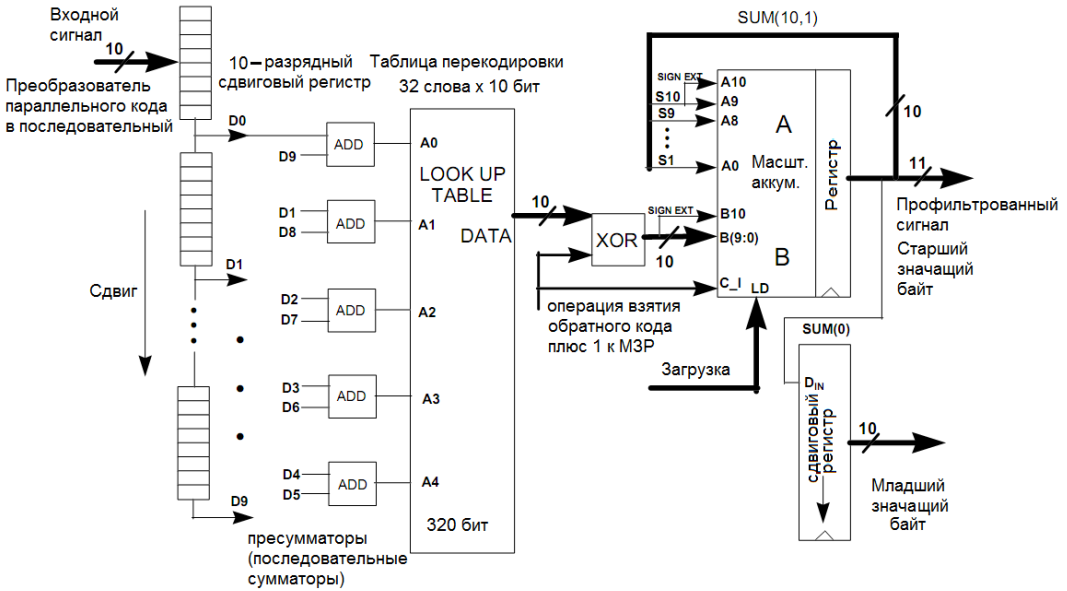


Рис. 2.7. Симметричный КИХ-фильтр со структурой 10 отводов 10 бит с точностью представления коэффициентов 10 бит с использованием последовательной распределенной арифметики

Рассмотрим уравнение КИХ-фильтра (нерекурсивного цифрового фильтра с конечно-импульсной характеристикой), которое представляется как арифметическая сумма произведений:

$$y = \sum_{k=0}^{K-1} C_k \cdot x_k \ , \qquad (2.1)$$

где $y-$ отклик цепи; $x_k -$ k-я входная переменная; $C_k -$весовой коэффициент k-й входной переменной, который является постоянным для всех n; K – число отводов фильтра.

В качестве простейшего примера рассмотрим три варианта проектирования параллельного КИХ-фильтра на четыре отвода: $y = C_0x_0 + C_1x_1 + C_2x_2 + C_3x_3$ с использованием мегафункций САПР ПЛИС Quartus II, объединенных общей идеей использования умножителей цифровых сигналов и «дерева сумматоров». Предположим, что коэффициенты фильтра целочисленные со знаком известны и равны: $C_0 = -2$, $C_1 = -1$, $C_2 = 7$ и $C_3 = 6$. На вход КИХ-фильтра поступают входные отсчеты –5, 3, 1 и 0. Правильные значения на выходе фильтра: 10, –1, –40, –10, 26, 6 и т. д., т. е. согласно формуле, $y = C_0x_0 + C_1x_1 + C_2x_2 + C_3x_3$.

Первый вариант. Параллельная реализация КИХ-фильтра на четыре отвода с использованием четырех блоков умножения с накоплением. В проекте используются четыре мегафункции ALTMULT_ACCUM (рис. 2.8). Каждый блок использует один умножитель и один сумматор-аккумулятор. Для параллельной реализации фильтра на четыре отвода требуются четыре блока и три дополнительных однотипных многоразрядных сумматора, связанных по принципу «дерево сумматоров». Для того чтобы фильтр работал корректно (в этом случае сумматор-аккумулятор превращается в обычный сумматор), необходимо осуществлять синхронную загрузку каждого произведения в каждый сумматор-аккумулятор каждого блока, для этого используется дополнительный вход мегафункции accum_sload.

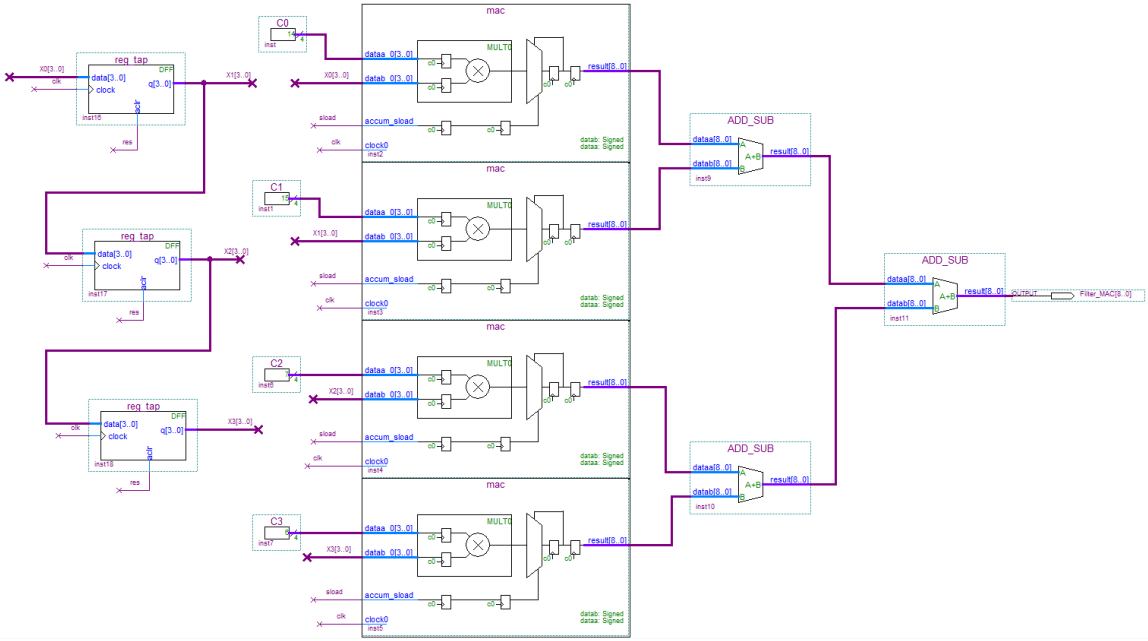


Рис. 2.8. Параллельная реализация КИХ-фильтра на четыре отвода с использованием четырех блоков в САПР ПЛИС Quartus II (мегафункция ALTMULT_ACCUM)

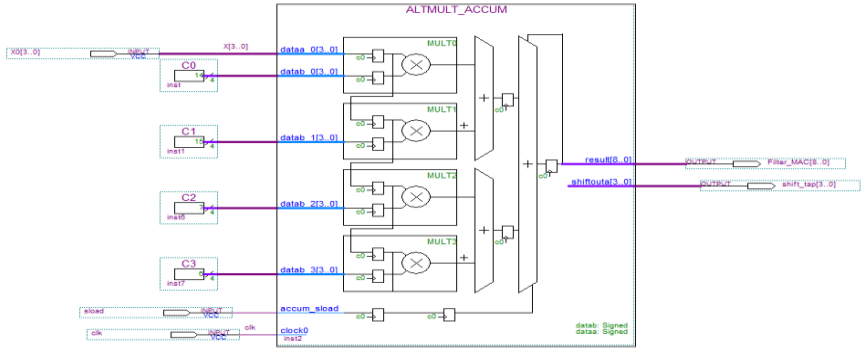


Рис. 2.9. Параллельная реализация КИХ-фильтра на четыре отвода с использованием четырех перемножителей в блоке (мегафункция ALTMULT_ACCUM, линия задержки построена на внутренних регистрах перемножителей MULT0-MULT3)

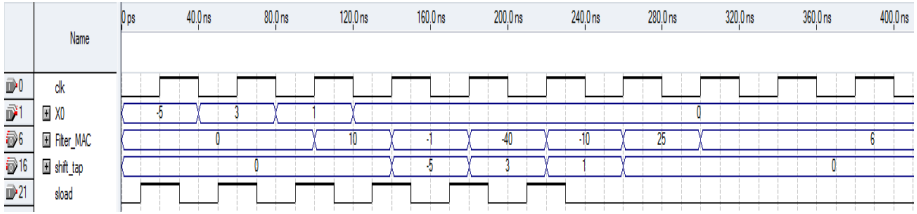


Рис. 2.10. Временные диаграммы работы фильтра на четыре отвода с использованием мегафункции ALTMULT_ACCUM

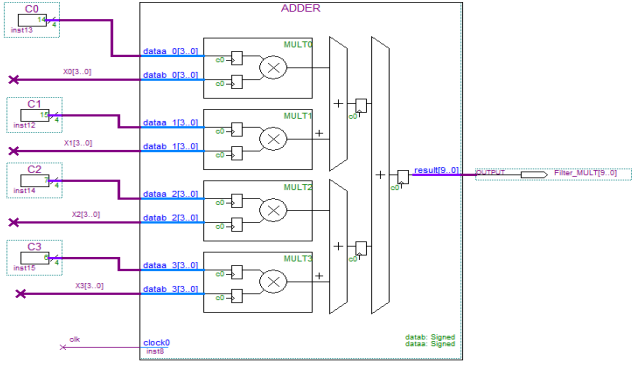


Рис. 2.11. Параллельная реализация КИХ-фильтра на четыре отвода с использованием четырех перемножителей в блоке (мегафункция ALTMULT_ADD, линия задержки такая же, как и на рис. 2.13)

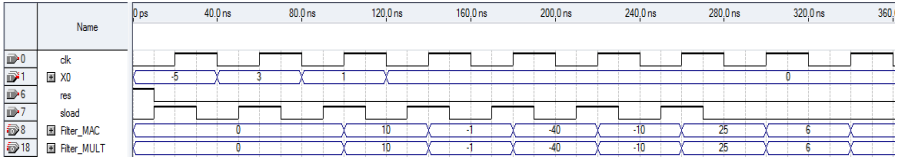


Рис. 2.12. Временные диаграммы работы параллельных фильтров на четыре отвода с использованием мегафункции ALTMULT_ACCUM и ALTMULT_ADD

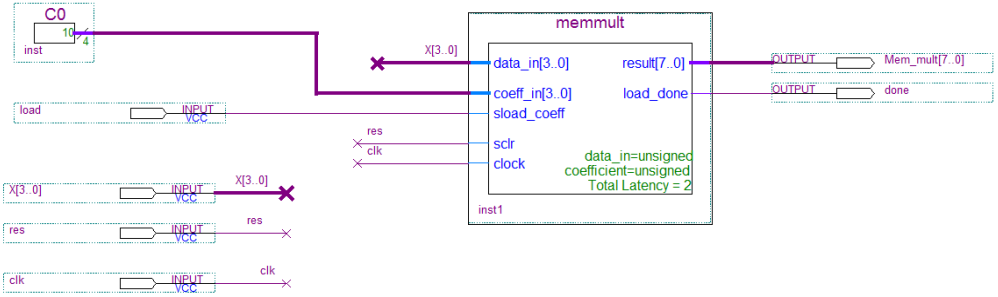


Рис. 2.13. Умножение 11 на 10 с помощью мегафункции ALTMEMMULT

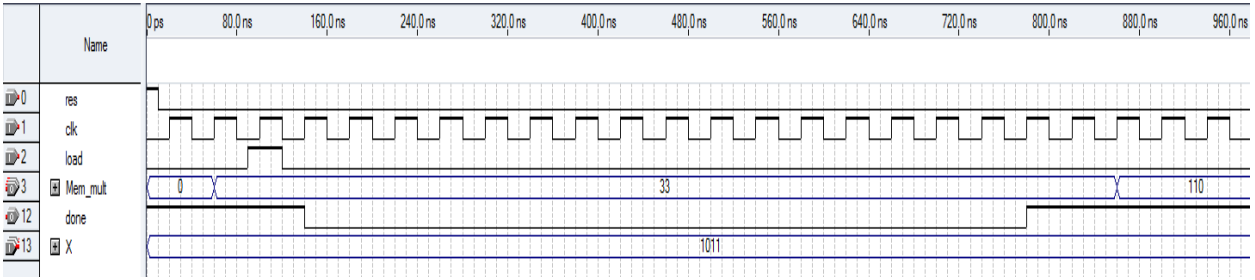


Рис. 2.14. Временные диаграммы умножения 11 на 10. По умолчанию в мегафункцию загружена константа 3. Результат 110

На рис. 2.8 также показана внешняя линия задержки на четыре отвода из трех 4-разрядных регистров, тактируемых фронтом синхросигнала. Коэффициенты фильтра представляются в двоичном виде с учетом знака числа и загружаются с помощью мегафункции LPM_CONSTANT.

Значительно упростить разработку проекта КИХ-фильтра позволяет иное использование мегафункции ALTMULT_ACCUM. Фактически это одна мегафункция (блок с четырьмя умножителями), в которой линия задержки организована на внутренних регистрах, располагающихся на входах перемножителей. В мегафункции используются встроенные два сумматора и один сумматор-аккумулятор (рис. 2.9). В мегафункции ALTMULT_ACCUM используются три встроенных сумматора. Профильтрованные значения показаны на рис. 2.10.

Второй вариант. Параллельная реализация КИХ-фильтра на четыре отвода с использованием четырех умножителей в блоке на мегафункции ALTMULT_ADD (функция умножения и сложения) в САПР ПЛИС Quartus II показана на рис. 2.11. Сравнивая временные диаграммы, видим, что профильтрованные значения на выходе у двух фильтров, построенных на разных мегафункциях, совпадают. Временные диаграммы работы фильтров, показанные на рис. 2.12, не отличаются от диаграмм на рис. 2.10.

Третий вариант. Рассмотрим умножение десятичного числа 11 на 10 на примере мегафункции ALTMEMMULT (рис. 2.13). Мегафункция ALTMEMMULT (программный умножитель) предназначена для умножения числа на константу, которая хранится в блочной памяти ПЛИС (M512, M4K, M9K и MLAB-блоки), обеспечивая наивысшее быстродействие, лимитируемое латентностью. Однако константу можно загрузить и из внешнего порта.

Считаем, что десятичное число 10 – это константа и загружается из внешнего порта. По умолчанию в мегафункцию загружена, например, константа 3. Латентность мегафункции -2, т. е. доступность результата умножения числа на константу, если константа хранится в памяти мегафункции, возможно уже

после 2 синхроимпульсов (высокий уровень сигнала done, соответствующий порту load_done). Число 3, загруженное в мегафункцию по умолчанию и умноженное на число 11, с входного порта data_in[3..0] дает результат 33. Далее, синхронный сигнал загрузки load (порт sload_coeff) разрешает загрузку числа 10 в перемножитель. Низкий уровень сигнала done в течение 16 тактов синхрочастоты говорит о том, что идет процесс умножения. И лишь спустя 2 синхроимпульса при высоком уровне сигнала done на выходе появляется требуемое число 110. Таким образом, процесс умножения составляет 20 синхроимпульсов от момента появления сигнала load (рис. 2.14).

Применяя мегафункцию ALTMEMMULT, разработаем параллельную реализацию КИХ-фильтра на четыре отвода с использованием четырех перемножителей (4 блока по 1 умножителю в каждом) в САПР ПЛИС Quartus II и дерева сумматоров (рис. 2.15). Внешняя линия задержки состоит не из трех регистров, как в первых двух вариантах, а из четырех регистров. Дополнительно требуются, как и в первом варианте, три однотипных многоразрядных сумматора.

Латентность каждого умножителя равна двум тактам синхроимпульса. В каждый умножитель по умолчанию загружено число 0. Временные диаграммы работы фильтра на четыре отвода с использованием мегафункции ALTMEMMULT показаны на рис. 2.16. Коэффициенты фильтра $C_0 = -2$, $C_1 = -1$, $C_2 = 7$ и $C_3 = 6$ загружаются из внешнего порта. Для этих целей используется мегафункция LPM_CONSTANT.

Рассмотрим вариант, когда коэффициенты фильтра загружаются из блочной памяти в ПЛИС. В мегафункции ALTMEMMULT коэффициенты представляются как целочисленные значения со знаком (рис. 2.17). Временные диаграммы работы фильтра на четыре отвода с использованием мегафункции ALTMEMMULT показаны на рис. 2.18. Сравнивая рис. 2.18 и 2.16 видим, что быстродействие фильтра в этом случае значительно

увеличивается за счет хранения коэффициентов в блочной памяти. Фильтр на мегафункции ALTMULT_ACCUM (вариант 1) является самым затратным, так как требует 16 аппаратных умножителей, три дополнительных сумматора и внешнюю линию задержки.

Наиболее оптимальным по числу используемых ресурсов ПЛИС является модификация варианта 1 (рис. 2.9), которая позволяет построить параллельный КИХ-фильтр на четыре отвода с использованием всего лишь одного блока со встроенными умножителями в количестве четырех штук, двумя сумматорами, сумматором-аккумулятором и линией задержки. Мегафункции ALTMULT_ADD (рис. 2.11) также позволяют построить параллельный КИХ-фильтр с использованием всего лишь одного блока со встроенными умножителями и сумматорами. Использование внешней линии задержки из трех регистров приводит к незначительному увеличению ресурсов и не сказывается на быстродействии. Экономия ресурсов ПЛИС в первом модифицированном и во втором вариантах достигается за счет использования встроенных четырех аппаратных умножителей размерностью 18x18.

Фильтр на мегафункции ALTMEMMULT с загрузкой коэффициентов из внешнего порта обладает пониженным быстродействием (рис. 2.15). Использование же блочной памяти (рис. 2.17) для хранения коэффициентов фильтра внутри ПЛИС значительно упрощает процесс разработки и не приводит к существенному увеличению ресурсов за счет использования внешней линии задержки на четырех регистрах, дополнительных трех однотипных многоразрядных сумматоров и не снижает быстродействие проекта.

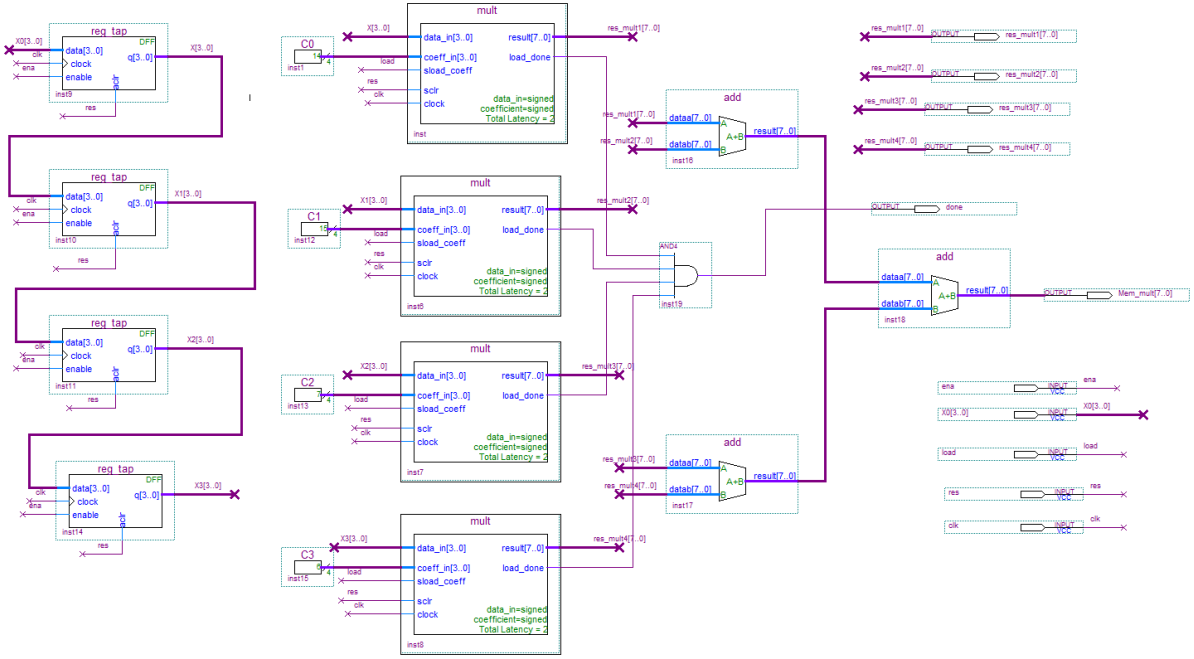


Рис. 2.15. Параллельная реализация КИХ-фильтра на четыре отвода с использованием четырех умножителей в САПР ПЛИС Quartus II (мегафункция ALTMEMMULT, линия задержки построена на четырех регистрах, коэффициенты фильтра загружаются из внешнего порта)

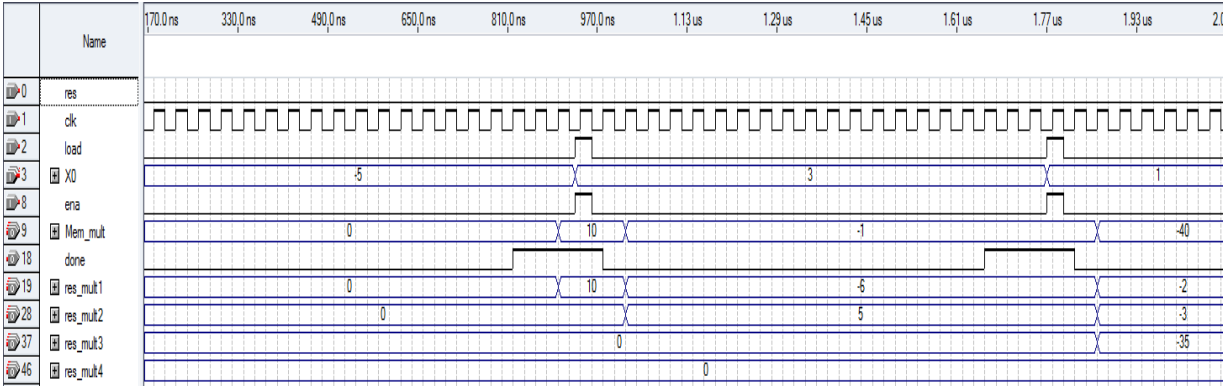


Рис. 2.16. Временные диаграммы работы фильтра на четыре отвода с использованием мегафункции ALTMEMMULT (коэффициенты фильтра загружаются из внешнего порта)

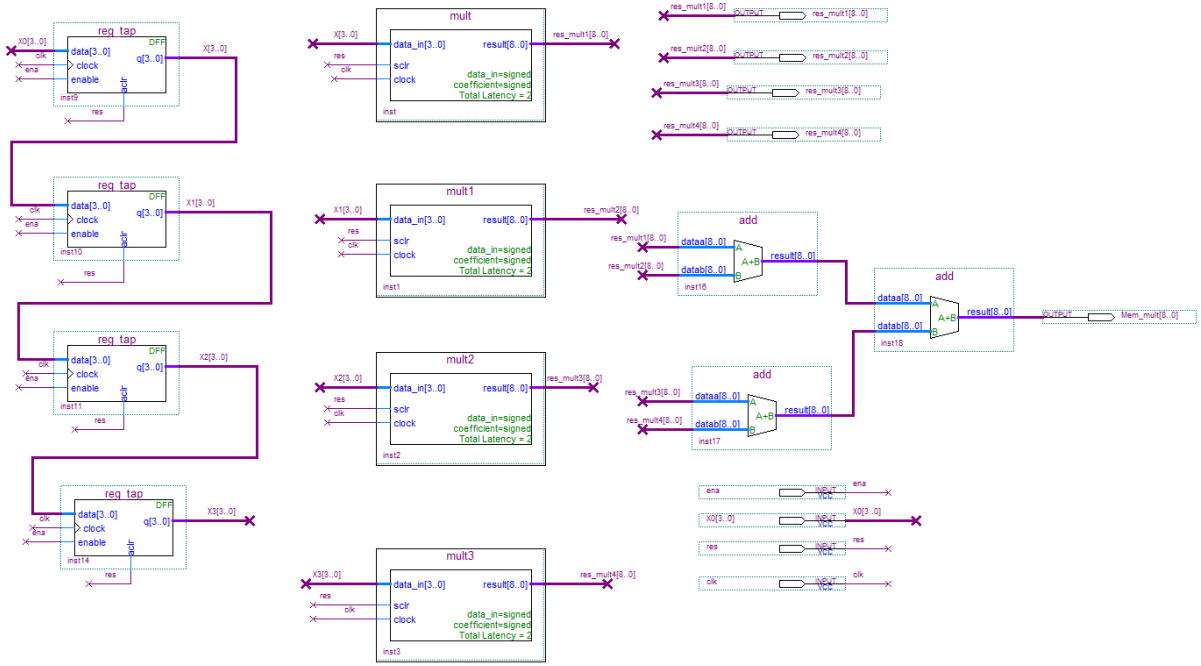


Рис. 2.17. Параллельная реализация КИХ-фильтра на четыре отвода с использованием четырех умножителей в САПР ПЛИС Quartus II (мегафункция ALTMEMMULT, линия задержки построена на четырех регистрах, коэффициенты фильтра загружаются из блочной памяти)

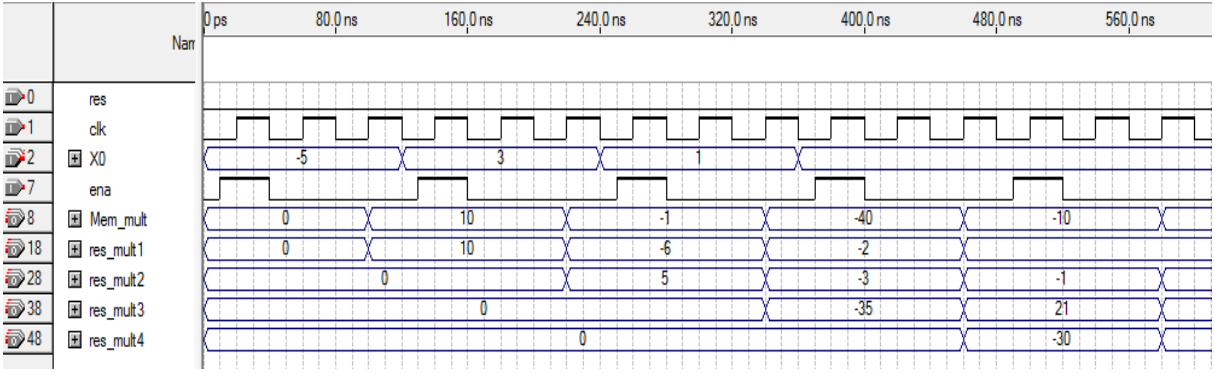


Рис. 2.18. Временные диаграммы работы фильтра на четыре отвода с использованием мегафункции ALTMEMMULT (коэффициенты фильтра загружаются из блочной памяти ПЛИС)

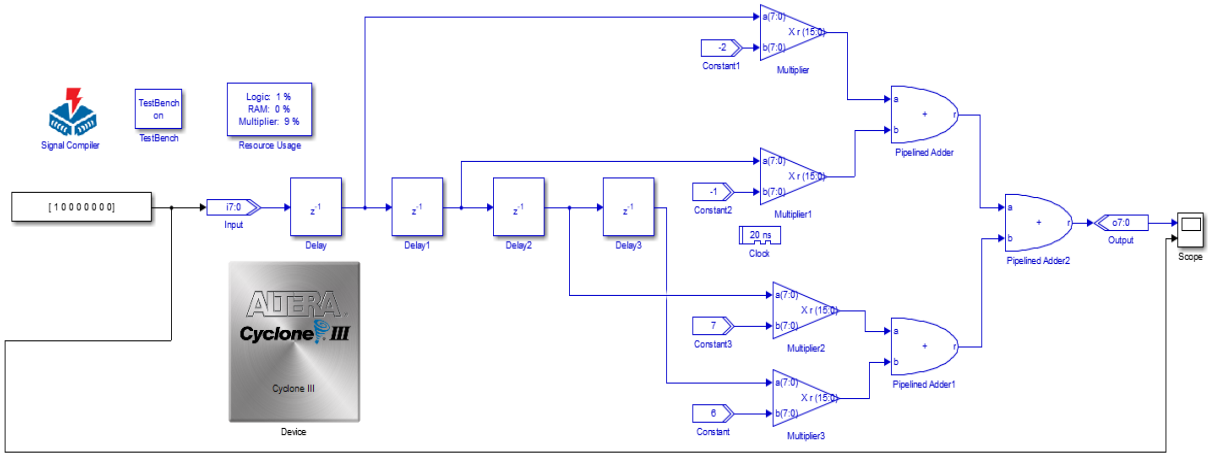


Рис. 2.19. Имитационная модель параллельного КИХ-фильтра на четыре отвода для реализации в базисе ПЛИС Altera серии Cyclone III на основе линии задержки из функциональных блоков Delay AlteraBlockset

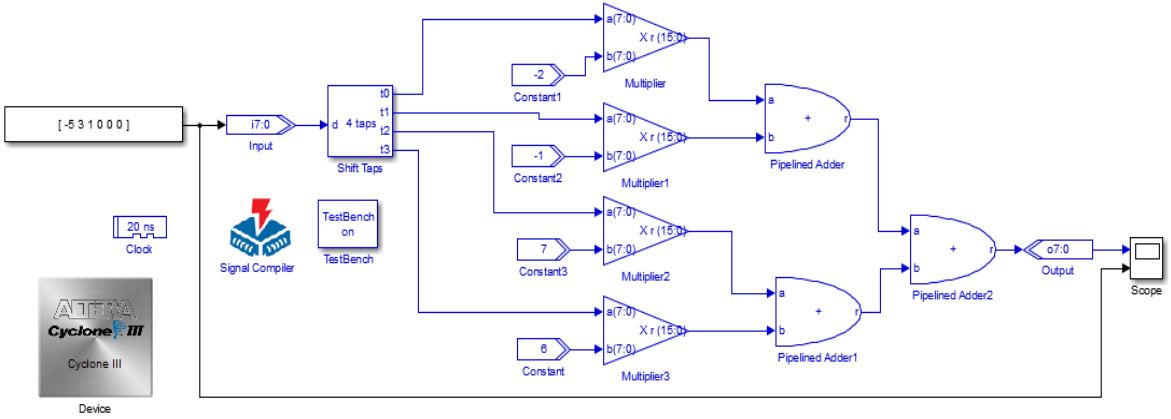


Рис. 2.20. Имитационная модель параллельного КИХ-фильтра на четыре отвода, линия задержки на функциональном блоке Shift Taps

Рассмотрим разработку параллельного КИХ-фильтра на четыре отвода в Altera DSP Builder: $y = C_0x_0 + C_1x_1 + C_2x_2 + C_3x_3$. Рассмотрим случай, когда на вход фильтра поступает сигнал 1, 0, 0, 0, 0, 0 и 0 т.д., представляющий собой единичную дельта-функцию, позволяющую увидеть на выходе фильтра его коэффициенты (импульсная характеристика). Правильные значения на выходе фильтра: –2, –1, 7, 6, 0, 0 и 0 т.д.

На рис. 2.19 представлена имитационная модель параллельного КИХ-фильтра на четыре отвода для реализации в базисе ПЛИС Altera серии Cyclone III на основе линии задержки из функциональных блоков Delay AlteraBlockset. Для функционального моделирования задается синхросигнал с периодом 20 нс и временной шаг симуляции в Matlab/Simulink 0.25 с, интервал дискретизации входного сигнала 1 с. В отличие от последовательного КИХ-фильтра на 4 отвода для параллельного фильтра временной шаг симуляции в Matlab/Simulink задается 1 с.

На рис. 2.20 показана имитационная модель параллельного КИХ-фильтра на четыре отвода, линия задержки на функциональном блоке Shift Taps, а на рис.18 – имитационное моделирование.

В данном разделе рассмотрены различные варианты проектирования параллельных КИХ-фильтров с использованием мегафункций САПР ПЛИС Quartus II компании Altera.

Показано, что КИХ-фильтр на мегафункции ALTMULT_ACCUM с использованием четырех блоков умножения с накоплением является самым затратным. Наиболее оптимальным по числу используемых ресурсов ПЛИС является его модификация, позволяющая построить параллельный КИХ-фильтр на 4 отвода с использованием всего лишь одного блока со встроенными перемножителями в

количестве четырех штук, двумя сумматорами, сумматором-аккумулятором и линией задержки.

Мегафункции ALTMULT_ADD позволяют построить параллельный КИХ-фильтр с использованием всего лишь одного блока со встроенными умножителями и сумматорами, выполняющего функцию умножения и сложения. Использование внешней линии задержки из трех регистров приводит к незначительному увеличению ресурсов и не сказывается на быстродействии. Экономия ресурсов ПЛИС в первом модифицированном и во втором вариантах достигается за счет использования встроенных четырех аппаратных умножителей размерностью 18×18.

Фильтр на мегафункции ALTMEMMULT с загрузкой коэффициентов из внешнего порта обладает пониженным быстродействием. Использование же блочной памяти (рис. 2.17) для хранения коэффициентов фильтра внутри ПЛИС значительно упрощает процесс разработки и не приводит к существенному увеличению ресурсов за счет использования внешней линии задержки на четырех регистрах, дополнительных трех однотипных многоразрядных сумматоров и не снижает быстродействие проекта.

В заключении отметим, что пакет расширения Altera DSP Builder системы визуально-имитационного моделирования Matlab/Simulink является мощным инструментом по разработке устройств цифровой обработки сигналов.

3. ПРОЕКТИРОВАНИЕ ПОСЛЕДОВАТЕЛЬНЫХ КИХ-ФИЛЬТРОВ В БАЗИСЕ ПЛИС

3.1. Проектирование последовательных КИХ-фильтров в САПР ПЛИС Quartus II

Параллельная структура позволяет вычислять результат фильтрации за один такт синхроимпульса. Параллельные фильтры обеспечивают наивысшую производительность и требуют большое количество аппаратных ресурсов ПЛИС. Механизм конвейерной обработки позволяет создавать фильтры с частотой от 120 до 300 МГц и выше.

На рис. 3.1 показана структура последовательного КИХ-фильтра на четыре отвода с использованием одного блока умножения и накопления (MAC-блока). Такие КИХ-фильтры, использующие в своей структуре блоки умножения и накопления, получили название в зарубежной литературе MAC FIR Filter (MAC КИХ-фильтры).

В структуре простейшего (single) MAC КИХ-фильтра используется один умножитель с аккумулятором по сравнению с полным параллельным фильтром. Этот компромисс снижает аппаратные затраты в N раз, но также снижает и пропускную способность фильтра на тот же фактор. В MAC КИХ-фильтрах пропускная способность выборки обратно пропорциональна количеству его отводов, т.е. при увеличении длины фильтра частота дискретизации системы пропорционально уменьшается. MAC-блок может быть реализован как на аппаратных ЦОС-блоках, так и на логических ресурсах ПЛИС.

Отказаться от использования аппаратных умножителей ЦОС-блоков позволяет распределенная арифметика. В этом случае частота дискретизации КИХ-фильтра уже не будет зависеть от его длины. Например, в мегаядре (MegaCore) Altera® FIR Compiler и в генераторе параметризированных ядер Xilinx CORE Generator реализована поддержка

параллельной и последовательной распределенной арифметики.

В случае низкой частоты дискретизации сигнала и большого количества коэффициентов КИХ-фильтра последовательная структура MAC КИХ-фильтра является оптимальной, а использование двухпортовой блочной памяти ПЛИС в конфигурации смешанного режима позволяет реализовать линию задержки на ОЗУ (циклический буфер входных данных) и ПЗУ для хранения коэффициентов фильтра.

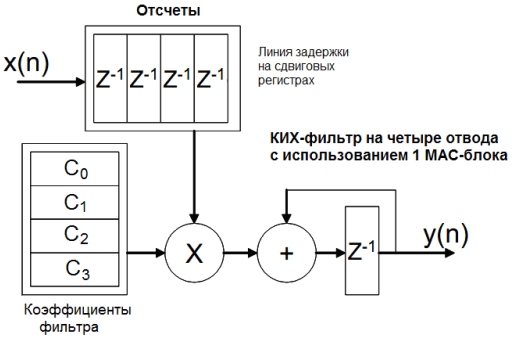


Рис. 3.1. КИХ-фильтр на четыре отвода
с использованием одного MAC-блока

На рис. 3.2 показан КИХ-фильтр с использованием ЦОС-блока DSP48 ПЛИС Xilinx Virtex-4 для реализации операций умножения и накопления и блочной памяти ПЛИС (Dual-Port Block RAM). Данные записываются и считываются из порта A (режим ОЗУ), а коэффициенты считываются только с порта B (режим ПЗУ). В отличие от структуры параллельного КИХ-фильтра требуется наличие управляющего автомата (блок Control), который обеспечивает необходимую адресную логику для организации циклического буфера ОЗУ для порта A (линия задержки) и B (ПЗУ), а также вырабатывает сигналы

разрешения загрузки данных и коэффициентов в MAC-блок и захвата в регистр результата в случае если результат накопления не может быть немедленно использован в последующей обработке. Также необходим учет латентности работы основных узлов фильтра, так для формирования правильных значений на выходе MAC-блока и их последующего захвата регистром результата необходимо сигнал WE управляющего автомата задержать на 4 такта синхроимпульса.

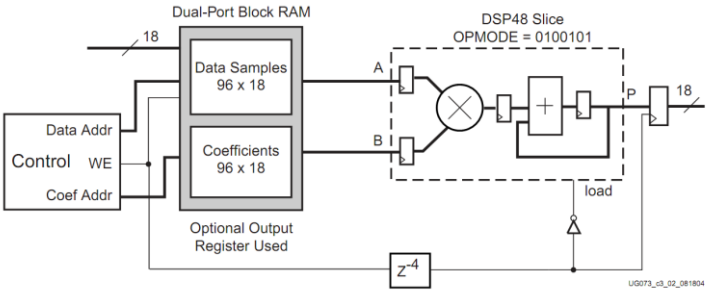


Рис. 3.2. КИХ-фильтр с использованием ЦОС-блока DSP49 ПЛИС Virtex-4

Рассмотрим проектирование последовательных КИХ-фильтров в САПР ПЛИС Quartus II. Последовательные КИХ-фильтры, использующие в своей основе блоки умножения и накопления, получили название в зарубежной литературе MAC FIR Filter (в нашем случае 1 MAC-фильтр).

Проект последовательного КИХ-фильтра на четыре отвода представлен в САПР Quartus II Version 5.1 (Build 170 10/3/2005) и до настоящего времени включен в перечень демонстрационных примеров более современных версий САПР, например Quartus II Version 13.1. Для САПР Quartus II ver 9.0 его можно найти по адресу altera\90\qdesigns\fir_filter, а для версии 13.0: altera\13.1\quartus\qdesigns\fir_filter. Линия задержки, ПЗУ для хранения коэффициентов и управляющий

автомат разработаны на языке Verilog, а умножитель – с применением мегафункции lpm_mult. Аккумулятор представляет иерархическое описание на языке Verilog, нижним уровнем которого является Verilog-файл, сгенерированный мегафункцией LPM_ADD_SUB. Умножитель в проекте представлен в виде символа, а линия задержки, коээфициенты и управляющий автомат в виде блок-схем.

Адаптируем данный пример под задачу проектирования последовательного КИХ-фильтра на четыре отвода $y = C_0x_0 + C_1x_1 + C_2x_2 + C_3x_3$ со следующими коэффициентами $C_0 = -2$, $C_1 = -1$, $C_2 = 7$ и $C_3 = 6$.

На рис. 3.3 показан иерархический проект последовательного КИХ-фильтра на четыре отвода в САПР Quartus II версии 13.0, а на рис. 3.4 – структурная схема проекта, представленная на уровне регистровых передач и полученная с помощью RTL-viewer. Входной сигнал и коэффициенты фильтра представлены в дополнительном коде с 8- и 4-битной точностью, поэтому умножитель и аккумулятор также настроены для выполнения операций над числами со знаком.

Проект использует два синхросигнала clk и clkx2. Синхросигнал clk используется для тактирования линии задержки и внутреннего регистра аккумулятора фильтра, а clkx2 – для тактирования работы регистра результата. Период синхросигнала clk в два раза больше clkx2. Умножитель и сумматор аккумулятора настроены для работы в асинхронных режимах.

Пример 1 демонстрирует Verilog-код линии задержки на четыре отвода. На рис. 3.5 показана структурная схема линии задержки на базе регистров. Пример 2 демонстрирует Verilog-код ПЗУ для хранения коэффициентов фильтра, а на рис. 3.6 показана структурная схема ПЗУ на базе дешифратора. Пример 3 демонстрирует verilog-код управляющего автомата

КИХ-фильтра. На рис. 3.7 показана структурная схема управляющего автомата (а); диаграмма состояний (б); таблица переходов и таблица кодирований состояний. Используется кодирование с одним активным или горячим состоянием (one hot encoding, OHE), т. е. в каждый конкретный момент времени активным (hot) может быть только один триггер состояния.

Метод OHE применительно к ПЛИС FPGA дает возможность строить конечные автоматы, которые в общем случае требуют меньших ресурсов и отличаются более высокими скоростными показателями, чем аналогичные конечные автоматы с двоичным кодированием состояний.

На рис. 3.8 показана структурная схема умножителя размерностью операндов 8×4, настроенная на асинхронный режим работы (clock=0). Умножитель проектируется на базе мегафункции LPM_MULT. Точность вычисления произведения 12-разрядов. Для реализации умножителя будем использовать аппаратные умножители ПЛИС. Всего аппаратных умножителей в ПЛИС EP3C5F256C6 с размерностью операндов 9×9 – 46 шт.

На рисунке 3.9 показан сумматор/вычитатель на базе мегафункции LPM_ADD_SUB. Сумматор/вычитатель способен работать как в асинхронном, так и в синхронном режиме. По умолчанию предполагается, что входные сигналы сумматора data[11..0] и datab[11..0] представлены в дополнительном коде.

Пример 4 демонстрирует Verilog-код аккумулятора КИХ-фильтра, а на рис. 3.10 показана структурная схема аккумулятора, соответствующая этому коду. Аккумулятор (функциональный блок acc) выполнен на основе асинхронного накапливающего сумматора, на базе мегафункции LPM_ADD_SUB, регистра и мультиплексора для принудительной установки входа сумматора datab[11..0] в ноль. Пример 5 демонстрирует фрагмент verilog-кода функции, используемой в примере 4.

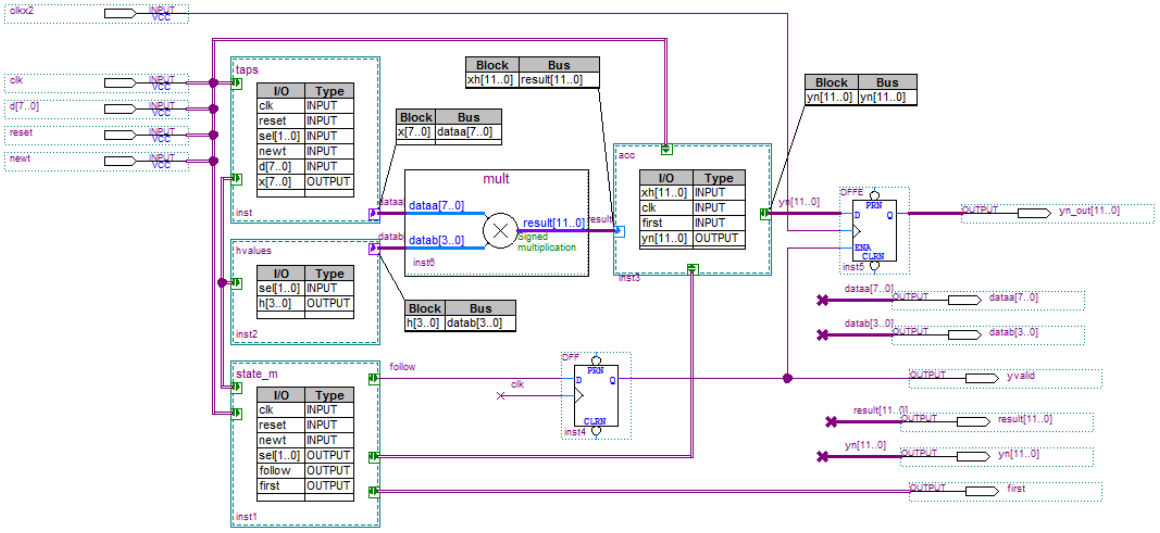


Рис. 3.3. Иерархический проект последовательного КИХ-фильтра на четыре отвода

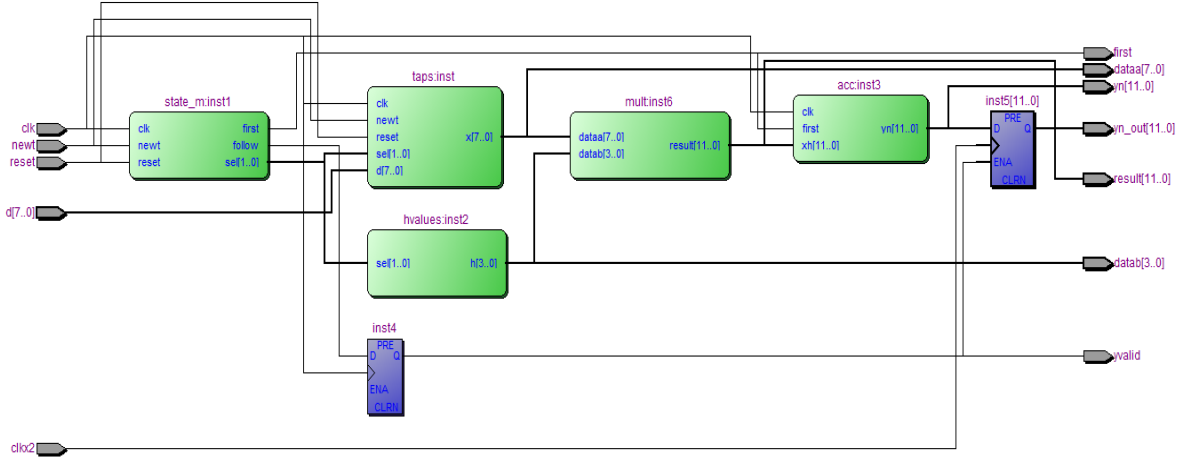


Рис. 3.4. Структурная схема проекта, полученная с помощью RTL-viewer

```
module taps
(       clk, reset, sel, newt, d, x);
        input clk;
        input reset;
        input [1:0] sel;
        input newt;
        input [7:0] d;
        output [7:0] x;
        reg [7:0] x, xn, xn_1, xn_2, xn_3;
// Register element
always @(posedge clk or posedge reset)
begin
if (reset)
        begin
                xn = 8'b00000000;
                xn_1 = 8'b00000000;
                xn_2 = 8'b00000000;
                xn_3 = 8'b00000000;
        end
else if (newt)
        begin
                xn_3 = xn_2;
                xn_2 = xn_1;
                xn_1 = xn;
                xn = d;
        end
end
// Mux element
always @(sel or xn or xn_1 or xn_2 or xn_3)
case (sel)
2'b 00: x = xn;
2'b 01: x = xn_1;
2'b 10: x = xn_2;
2'b 11: x = xn_3;
default: x = 8'bXXXXXXXX;
endcase
endmodule
```

Пример 1. Verilog-код линии задержки

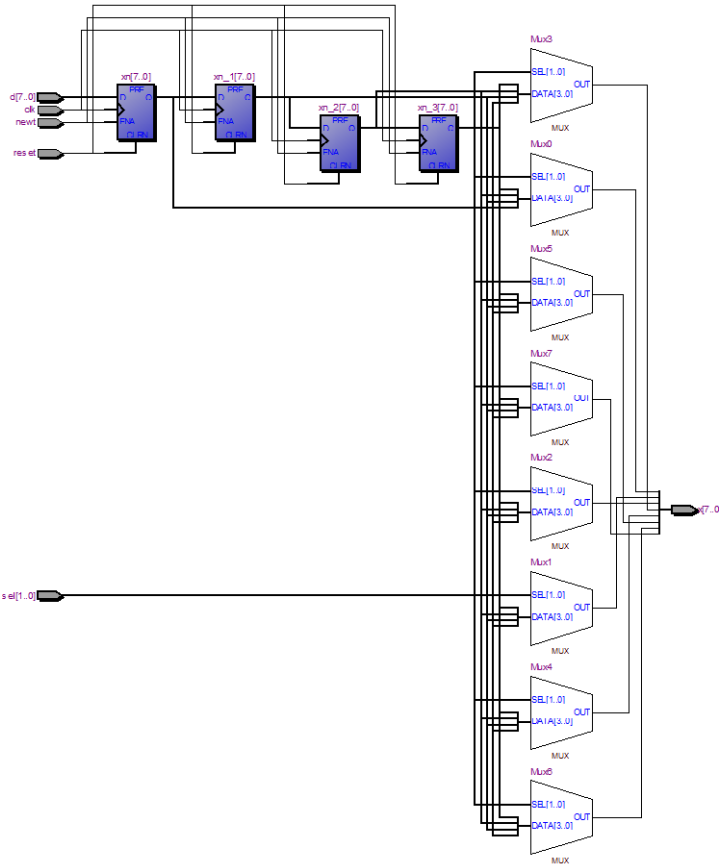


Рис. 3.5. Структурная схема линии задержки КИХ-фильтра

```
module hvalues
(
        sel, h
);
        input [1:0] sel;
        output [3:0] h;
reg [3:0]h;
always @(sel)
case (sel)
        2'b 00 : h = 4'b 1110;
```

```
            2'b 01 : h = 4'b 1111;
            2'b 10 : h = 4'b 0111;
            2'b 11 : h = 4'b 0110;
    endcase
    endmodule
```

Пример 2. Verilog-код ПЗУ для хранения коэффициентов фильтра

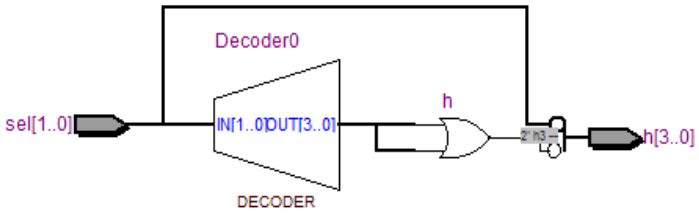


Рис. 3.6. Структурная схема ПЗУ

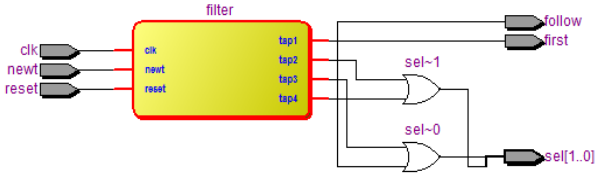
```
module state_m
(
        clk, reset, newt, sel, follow, first
);
        input clk, reset, newt;
        output follow, first;
        output [1:0]sel;
        reg follow, first;
        reg [1:0] sel;
        reg [2:0] filter;
parameter idle = 0, tap1 = 1, tap2 = 2, tap3 = 3, tap4 = 4;
always
begin
        case (filter)
                idle: begin
                                sel = 2'b0;
                                follow = 1'b0;
                                first = 1'b0;
                        end
                tap1: begin
                                sel = 2'b0;
                                follow = 1'b0;
                                first = 1'b1;
                        end
                tap2: begin
                                sel = 2'b01;
                                follow = 1'b0;
```

```
                                first = 1'b0;
                        end
                tap3: begin
                                sel = 2'b10;
                                follow = 1'b0;
                                first = 1'b0;
                        end
                tap4: begin
                                sel = 2'b11;
                                follow = 1'b1;
                                first = 1'b0;
                         end

                default : begin
                                        sel = 2'b00;
                                        follow = 1'b0;
                                        first = 1'b0;
                                        end
        endcase
end
always @(posedge clk or posedge reset)
begin
        if (reset)
                filter = idle;
        else
                case (filter)
                        idle: begin
                                if (newt)
                                        filter = tap1;
                                  end

                        tap1: begin
                                        filter = tap2;
                                   end
                        tap2: begin
                                        filter = tap3;
                                   end
                        tap3: begin
                                        filter = tap4;
                                   end
                        tap4: begin
                                        if (newt)
                                                filter = tap1;
                                        else
                                                filter = idle;
                                    end
                endcase
end
endmodule
```

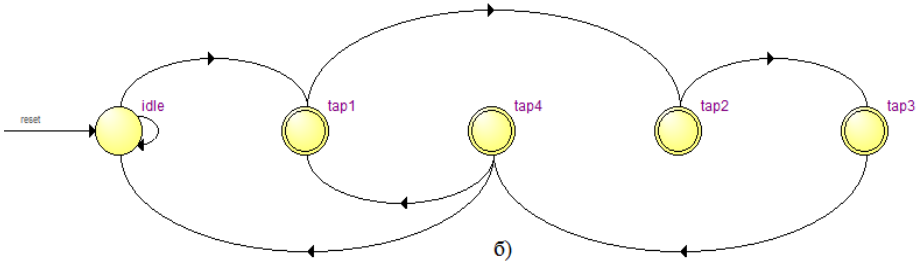
Пример 3. Verilog-код управляющего автомата КИХ-фильтра



а)

	Source State	Destination State	Condition
1	idle	idle	(!newt)
2	idle	tap1	(newt)
3	tap1	tap2	
4	tap2	tap3	
5	tap3	tap4	
6	tap4	idle	(!newt)
7	tap4	tap1	(newt)

Transitions | Encoding



б)

	Name	tap3	tap2	tap1	idle	tap4
1	idle	0	0	0	0	0
2	tap1	0	0	1	1	0
3	tap2	0	1	0	1	0
4	tap3	1	0	0	1	0
5	tap4	0	0	0	1	1

в)

Рис. 3.7. Структурная схема управляющего автомата (а); диаграмма состояний (б); таблица переходов и таблица кодирований состояний

Рис. 3.8. Структурная схема умножителя размерностью операндов 8×4 на базе мегафункции LPM_MULT

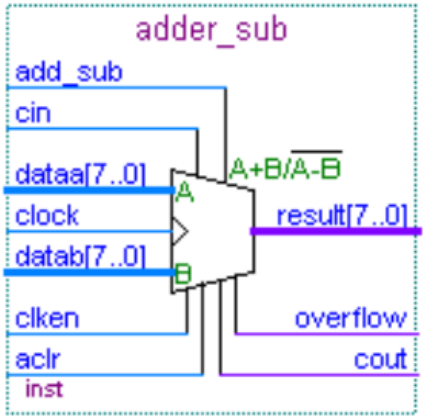


Рис. 3.9. Сумматор/вычитатель на базе мегафункции LPM_ADD_SUB

На рис. 3.11 и 3.12 показано функциональное моделирование импульсной характеристики КИХ-фильтра на четыре отвода и прохождение сигнала по структуре фильтра. На вход фильтра поступает сигнал –5, 3, 1, 0, 0 и 0 т.д. Правильные значения на выходе фильтра: 10, –1, –40, –10, 25, 6 и 0 т.д.

```
module acc
(
        xh, clk, first, yn
);
        input [11:0] xh;
        input clk;
        input first;
        output [11:0] yn;
reg [11:0]yn;
reg [11:0] ynm, result, a_in;
wire [11:0] inter;
// Describe Multiplexer
always @(first or result)
begin
        case (first)
                1'b 0: ynm = result;
                1'b 1: ynm = 12'b00000000000;
        endcase
end
always @(posedge clk)
begin
        result = inter;
end
always @(xh)
begin
        a_in[11:0] = (xh);
        //a_in[12] = 0;
end
always @(result)
        begin
                yn[11:0] = result[11:0];
```

```
        end
    accum  inst_13(.dataa(a_in), .datab(ynm), .result(inter));
    endmodule
```

Пример 4. Verilog-код аккумулятора КИХ-фильтра

```
    module accum (
        dataa,
        datab,
        result);

        input    [11:0]  dataa;
        input    [11:0]  datab;
        output   [11:0]  result;
        wire [11:0] sub_wire0;
        wire [11:0] result = sub_wire0[11:0];
        lpm_add_sub   lpm_add_sub_component (
                                .dataa (dataa),
                                .datab (datab),
                                .result (sub_wire0)
                                // synopsys translate_off
                                ,
                                .cout (),
                                .cin (),
                                .add_sub (),
                                .overflow (),
                                .clock (),
                                .clken (),
                                .aclr ()
                                // synopsys translate_on
                                );
        defparam
lpm_add_sub_component.lpm_direction = "ADD",
lpm_add_sub_component.lpm_hint =
"ONE_INPUT_IS_CONSTANT=NO,CIN_USED=NO",
lpm_add_sub_component.lpm_type = "LPM_ADD_SUB",
lpm_add_sub_component.lpm_width = 12;
    endmodule
```

Пример 5. Фрагмент Verilog-кода функции accum (накапливающий сумматор) на базе мегафункции lpm_add_sub

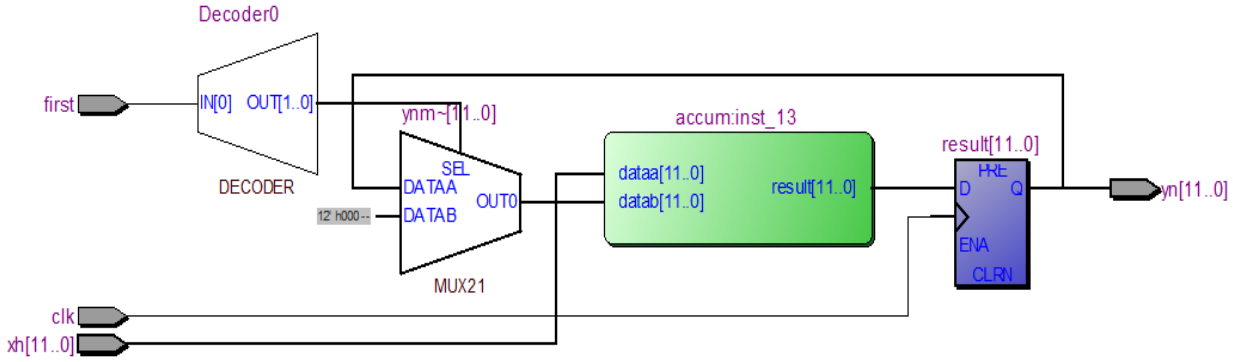


Рис. 3.10. Структурная схема аккумулятора на мегафункции LPM_ADD_SUB

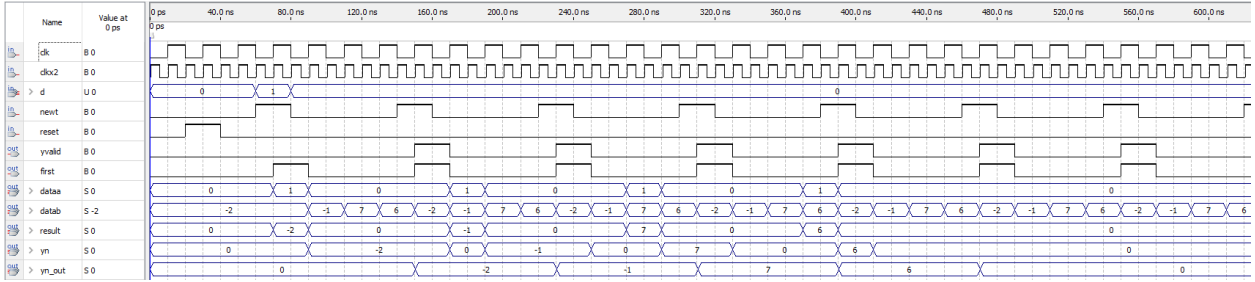


Рис. 3.11. Функциональное моделирование импульсной характеристики КИХ-фильтра на четыре отвода

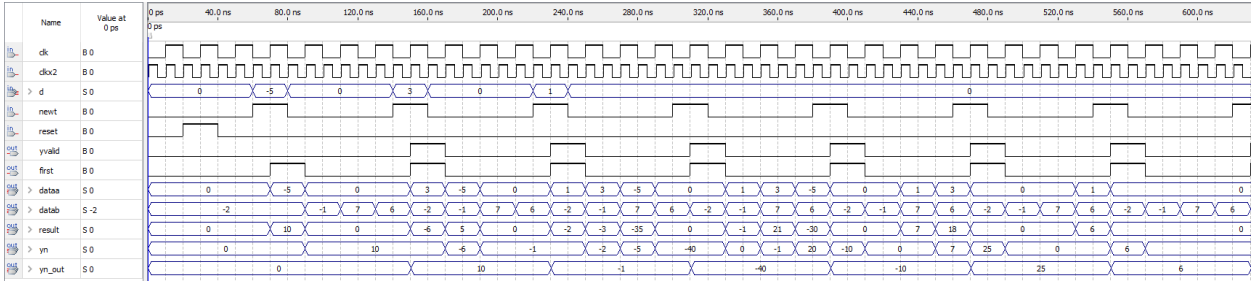


Рис. 3.12. Функциональное моделирование прохождения сигнала по структуре КИХ-фильтра на четыре отвода. На вход фильтра поступает сигнал –5, 3, 1, 0, 0 и 0 т. д. Правильные значения на выходе фильтра: 10, –1, –40, –10, 25, 6 и 0 т. д.

Рассмотрим разнообразие схемных решений проекта последовательного КИХ-фильтра в базисе ПЛИС Altera серии Cyclone IV EP4CGX22CF19C6 на основе демонстрационного примера fir_filter САПР Quartus II в условиях ограничения аппаратных ресурсов. Несмотря на то, что последовательный фильтр требует один умножитель и аккумулятор, при его реализации в составе сложного проекта в базисе отечественных ПЛИС могут возникнуть определенные трудности, например, недостаточное количество блоков встроенной памяти или ЦОС-блоков, при этом тактовая частота отечественных ПЛИС не превышает 250 МГц. Поэтому в работе показано как можно преодолеть эти ограничения, используя разнообразие схемных решений.

Вариант 1. Последовательный КИХ-фильтр на четыре отвода с использованием мегафункции умножения LPM_MULT (асинхронный) и накопления ALTACCUMULATE. Адаптируем оригинальный проект fir_filter к задаче проектирования последовательного КИХ-фильтра на четыре отвода $y = C_0x_0 + C_1x_1 + C_2x_2 + C_3x_3$ со следующими коэффициентами $C_0 = -2$, $C_1 = -1$, $C_2 = 7$ и $C_3 = 6$ в базисе ПЛИС Cyclone IV EP4CGX22CF19C6 (содержит 80 умножителей с размерностью операндов 9×9). Входной сигнал и коэффициенты фильтра представим с 8-разрядной точностью.

Линия задержки (taps.v), ПЗУ для хранения коэффициентов (hvalues.v), управляющий автомат (state_m.v) оригинальный Verilog-код. Умножитель настроен на работу с числами со знаком в асинхронном режиме. Во всех рассматриваемых вариантах реализации КИХ-фильтра умножитель строится на аппаратном умножителе ЦОС-блока, без использования логических ресурсов ПЛИС.

На рис. 3.13 показан проект последовательного КИХ-фильтра на четыре отвода, а на рис. 3.14 демонстрируется функциональное моделирование прохождения сигнала по структуре КИХ-фильтра.

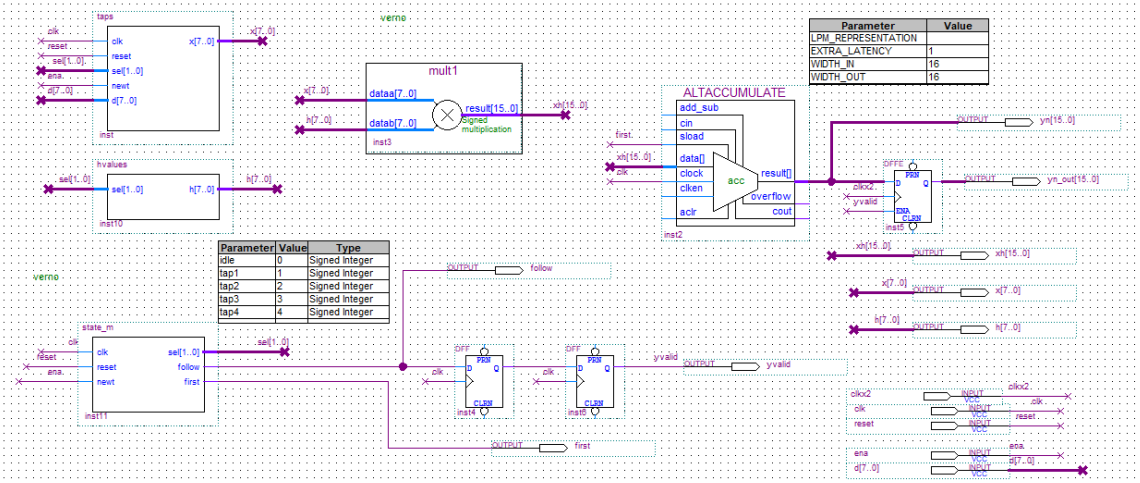


Рис. 3.13. Проект последовательного КИХ-фильтра на четыре отвода с использованием мегафункции умножения LPM_MULT (асинхронный режим) и накопления ALTACCUMULATE

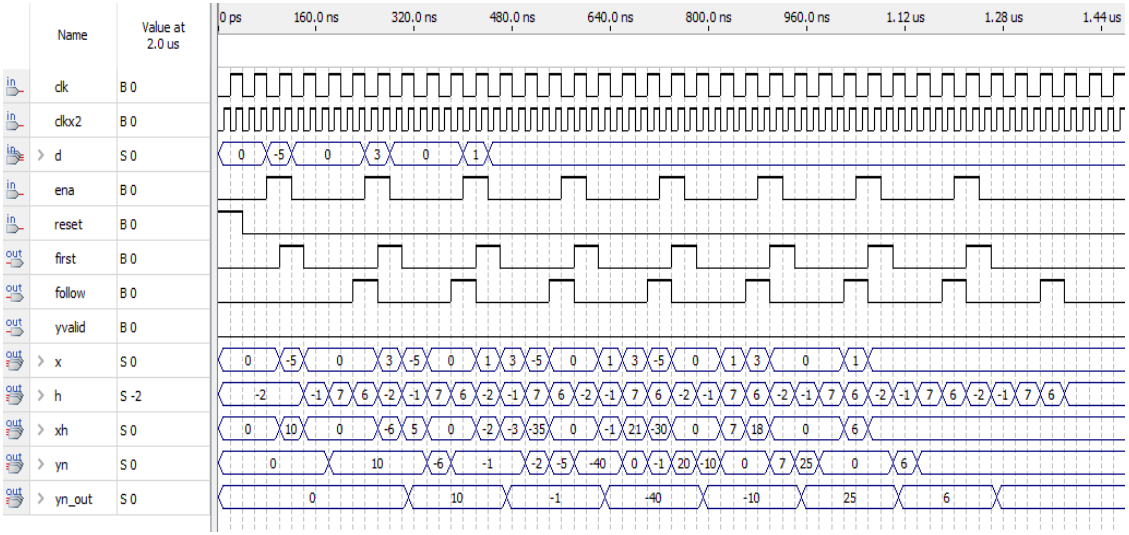


Рис. 3.14. Функциональное моделирование прохождения сигнала по структуре КИХ-фильтра на четыре отвода (асинхронный режим работы умножителя)

На вход фильтра поступает сигнал –5, 3, 1, 0, 0 и 0 т. д. Правильные значения на выходе фильтра: 10, –1, –40, –10, 26, 6 и 0 т. д. Аккумулятор работает в синхронном режиме, и его латентность составляет 1 такт синхросигнала. Для согласования работы управляющего автомата и аккумулятора необходимо выходной сигнал first автомата подключить к входу сигнала синхронной загрузки аккумулятора sload.

Результат умножения и вычисления суммы произведений представляется с 16-разрядной точностью. Для получения правильных значений профильтрованного сигнала на выходе yn_out[15..0] необходимо выходной сигнал follow управляющего автомата задержать на два такта синхросигнала и подключить его к входу разрешения тактирования ena регистра результата.

Вариант 2. Последовательный КИХ-фильтр на четыре отвода с использованием мегафункции умножения LPM_MULT (синхронный режим) и накопления ALTACCUMULATE. Умножитель работает в синхронном режиме (рис. 3.15). Все остальные блоки без изменений. Латентность умножителя и аккумулятора составляет один такт синхросигнала. Для согласования работы управляющего автомата, умножителя и аккумулятора необходимо выходной сигнал first автомата задержать на один такт синхросигнала с помощью двухтактного триггера, выход которого необходимо подключить к входу сигнала синхронной загрузки аккумулятора sload. А сигнал follow задержать на два такта с помощью двух триггеров.

Вариант 3. Последовательный КИХ-фильтр на четыре отвода с использованием мегафункции умножения и накопления ALTMULT_ACCUM. Рассмотрим вариант, когда умножитель и аккумулятор реализованы на мегафункции умножения и накопления ALTMULT_ACCUM без регистров на входах и выходах умножителя (рис. 3.17).

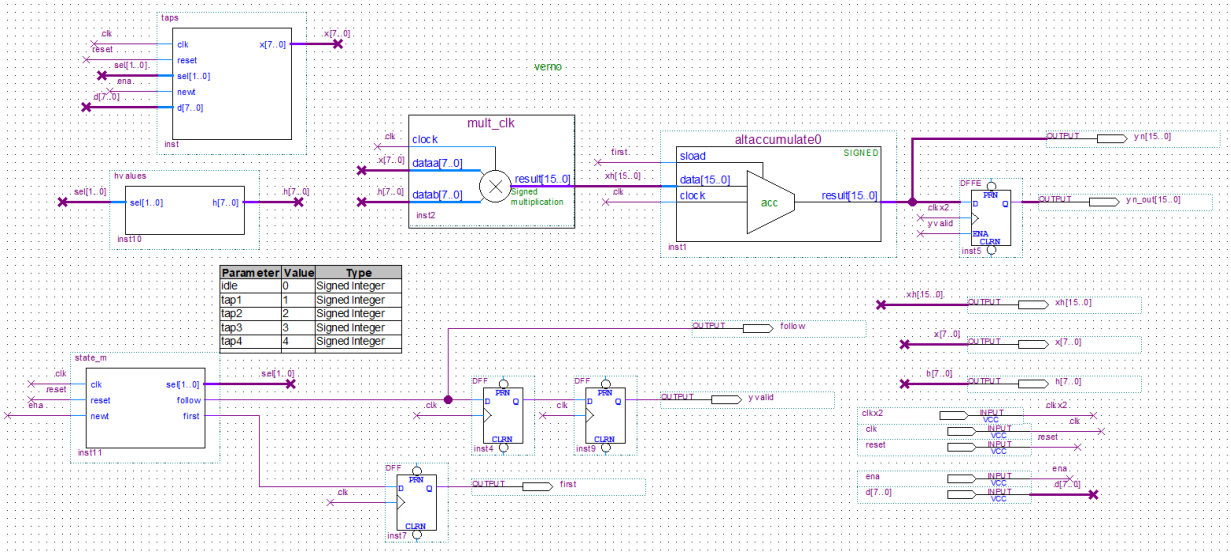


Рис. 3.15. Проект последовательного КИХ-фильтра на четыре отвода с использованием мегафункции умножения LPM_MULT (синхронный режим) и накопления ALTACCUMULATE

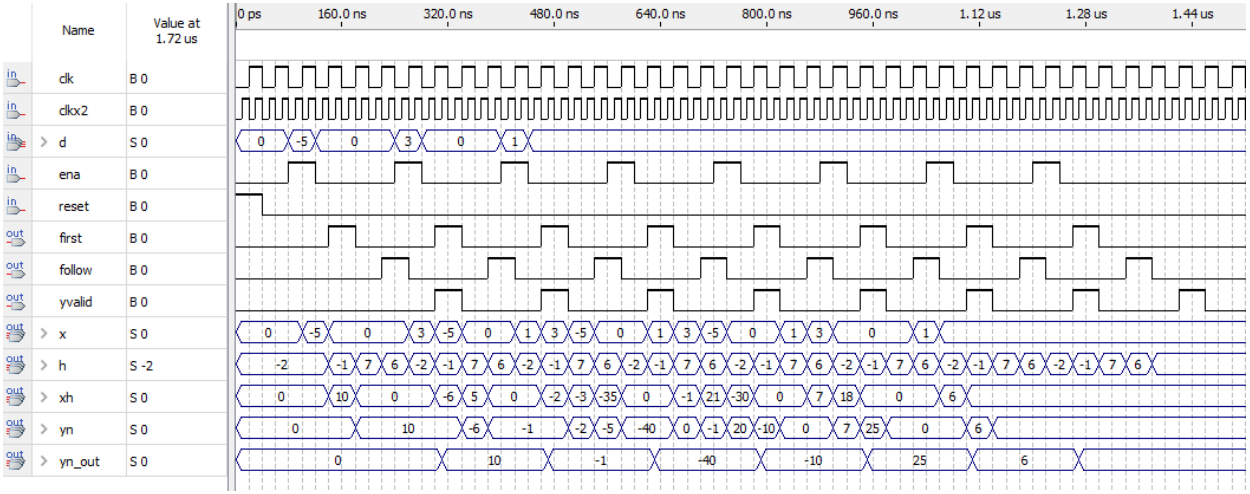


Рис. 3.16. Функциональное моделирование прохождения сигнала по структуре КИХ-фильтра на четыре отвода (умножитель в синхронном режиме)

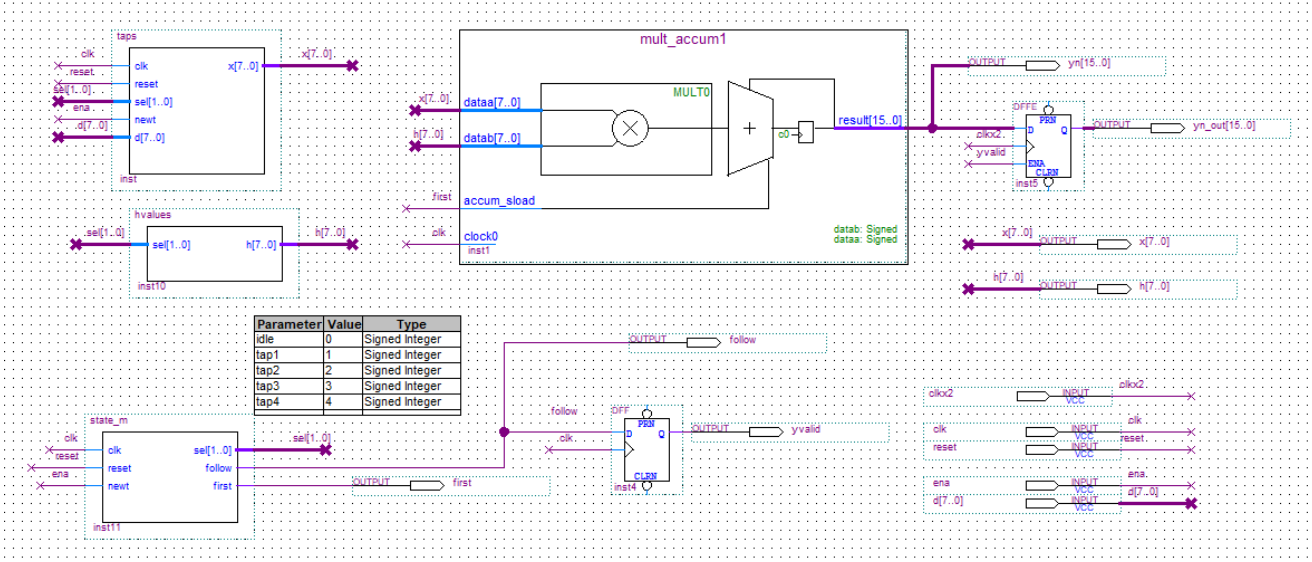


Рис. 3.17. Проект последовательного КИХ-фильтра на четыре отвода с использованием MAC-блока (мегафункция ALTMULT_ACCUM)

Линия задержки, ПЗУ, управляющий автомат представлены оригинальным Verilog-кодом. Выходной сигнал автомата first необходимо подключить к входу синхронной загрузки аккумулятора accum_sload. Для получения правильных значений профильтрованного сигнала на выходе yn_out[15..0] необходимо выходной сигнал follow задержать на один такт синхросигнала, т.к. умножитель в мегафункции ALTMULT_ACCUM работает в асинхронном режиме.

Вариант 4. Последовательный КИХ-фильтр на четыре отвода с использованием управляющего автомата, разработанного с помощью редактора Sate Flow. На основе оригинального Verilog-кода управляющего автомата (state_m.v) восстановим его графическое представление с помощью редактора State Flow (рис. 3.18). И извлечем в автоматическом режиме из его описания VHDL-код (SM_fir.vhd). Все остальные блоки и учет латентности как в варианте 3. Каких-либо дополнительных согласований работы блоков по латентности не требуется. А выходной сигнал follow необходимо задержать на один такт синхросигнала. Интересен вариант 4 представлением адресного сигнала sel[1:0]. В графическом схемном редакторе используется иное представление sel[1..0]. На рис. 3.19 показан проект последовательного КИХ-фильтра с использованием VHDL-кода, извлеченного из графического представления. Рис. 3.20 подтверждает правильность работы представленного варианта.

Вариант 5. Представляет собой модификацию варианта 4. Покажем, как можно реализовать аккумулятор без использования мегафункции ALTACCUMULATE (рис. 3.21 и рис. 3.22). Для этого необходимо использовать аккумулятор на основе синхронного сумматора на мегафункции lpm_add_sub и шинного мультиплексора 2 в 1 на мегафункции lpm_mux в обратной связи. На один из входов мультиплексора необходимо подключить константу – логический ноль. Выходной сигнал follow необходимо задержать на один такт синхронизации.

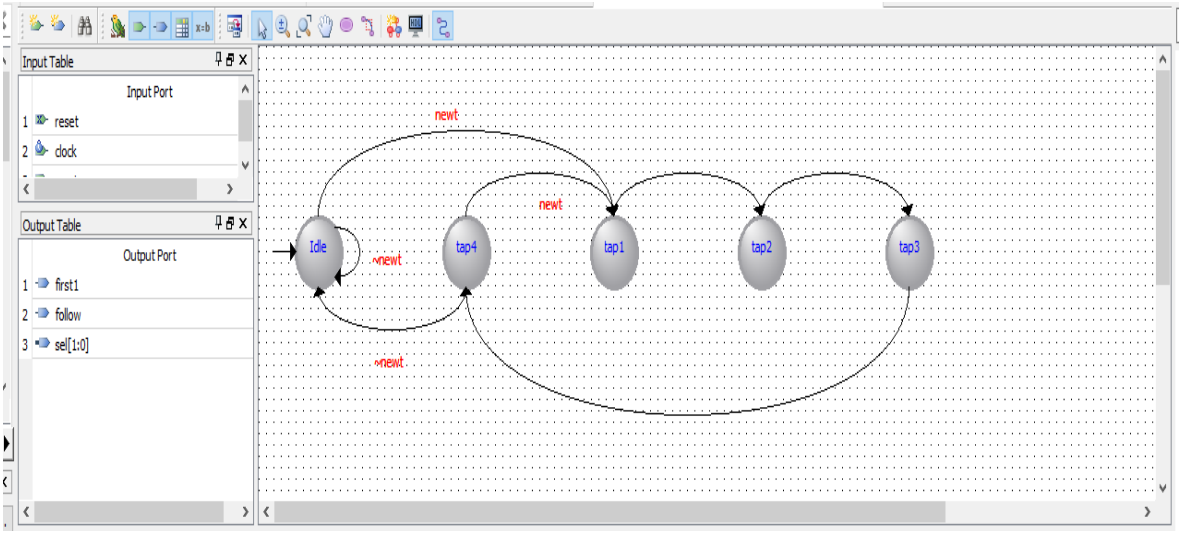


Рис. 3.18. Графическое представление управляющего автомата на пять состояний, восстановленное из оригинального Verilog-кода

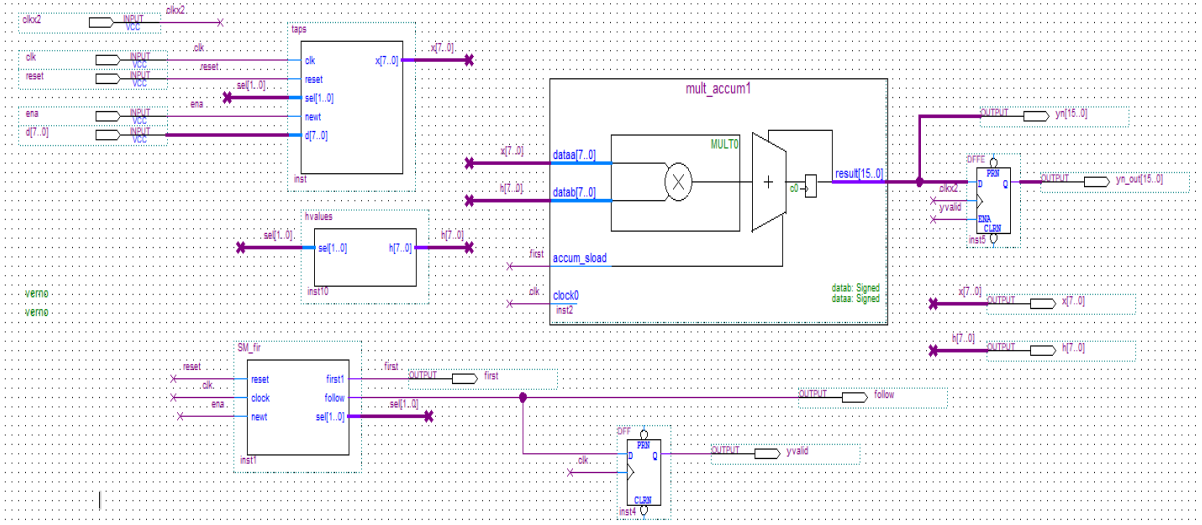


Рис. 3.19. Проект последовательного КИХ-фильтра на четыре отвода с использованием VHDL-кода, извлеченного из графического представления управляющего автомата

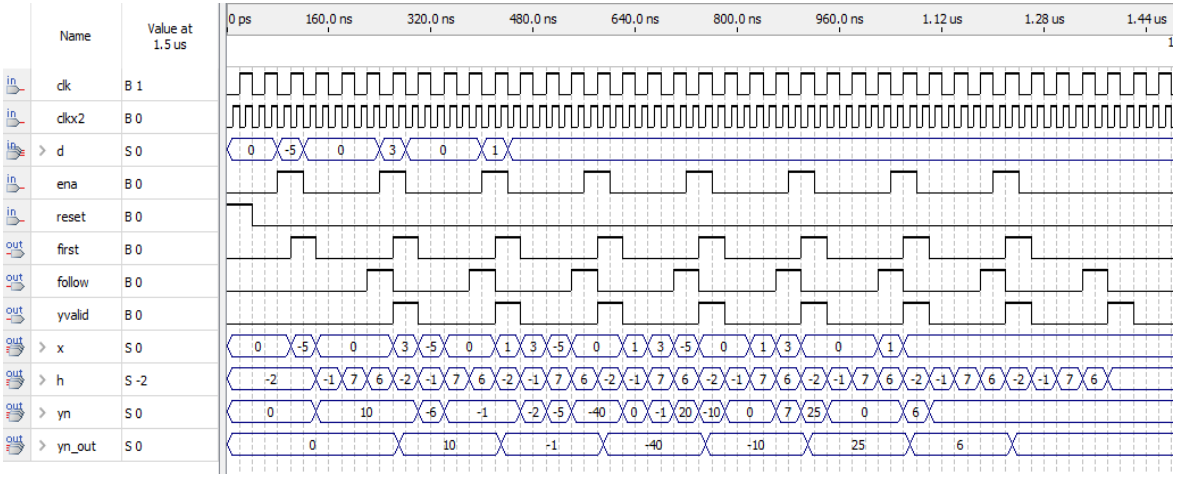


Рис. 3.20. Функциональное моделирование прохождения сигнала по структуре КИХ-фильтра на четыре отвода (управляющий автомат представлен VHDL-кодом)

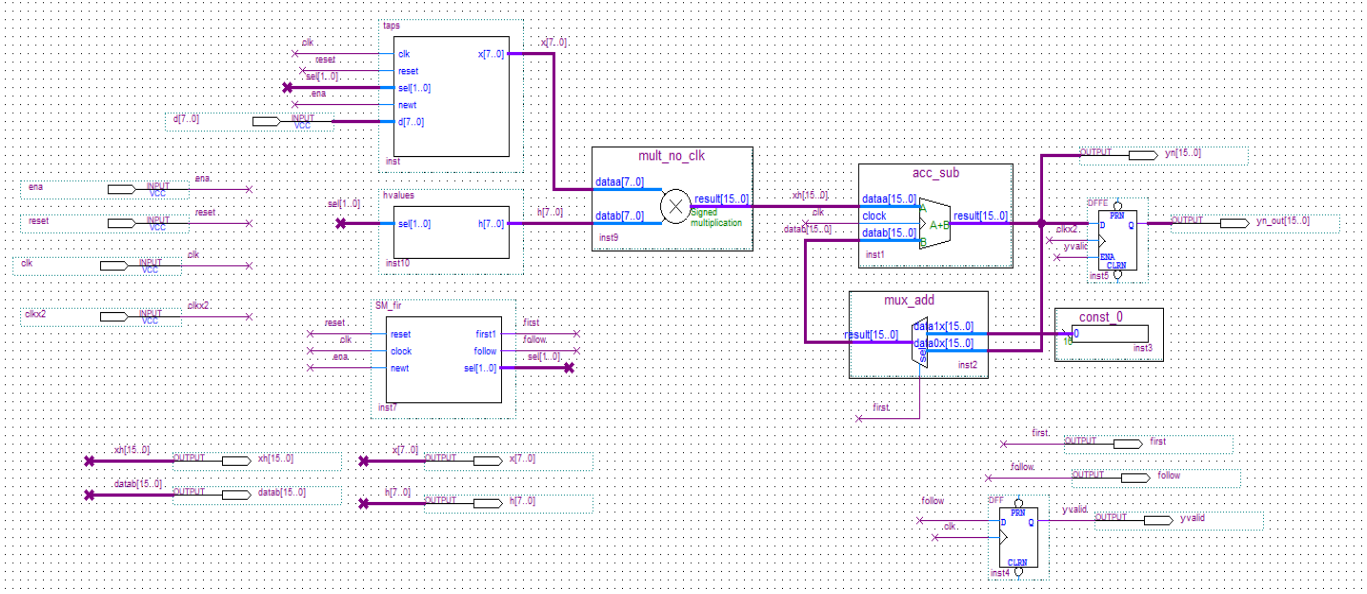


Рис. 3.21. Проект последовательного КИХ-фильтра на четыре отвода с использованием аккумулятора на основе синхронного сумматора на мегафункции lpm_add_sub и шинного мультиплексора 2 в 1 на мегафункции lpm_mux в обратной связи

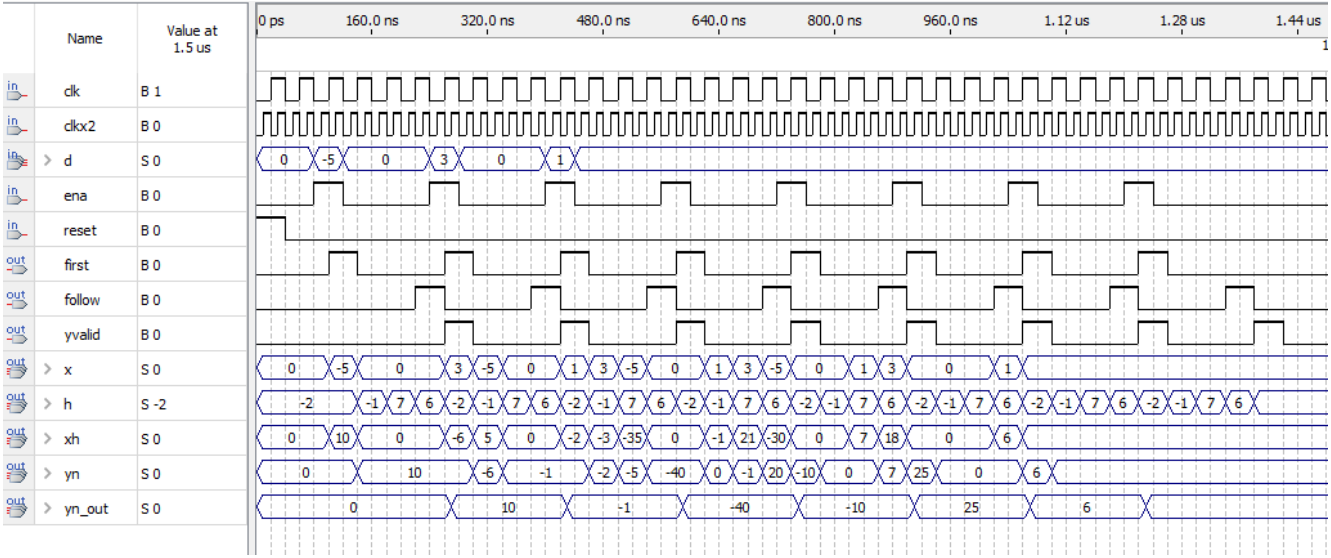


Рис. 3.22. Функциональное моделирование прохождения сигнала по структуре КИХ-фильтра с использованием аккумулятора на основе синхронного сумматора на мегафункции lpm_add_sub и шинного мультиплексора 2 в 1 на мегафункции lpm_mux в обратной связи

Вариант 6. Представляет собой вариант 4, с той лишь разницей, что линия задержки организована на регистрах (рис. 3.23). От оригинального проекта fir_filter используется лишь асинхронное ПЗУ, представленное Verilog-кодом. Отводы линии задержки коммутируются на один из входов умножителя с помощью шинного мультиплексора 4 в 1. Так же, как и в вариантах 1, 3–5, в которых умножитель работает в асинхронном режиме, сигнал first напрямую подключается к входу accum_sload мегафункции ALTMULT_ACCUM, а сигнал follow задерживается на два такта синхронизации. Результаты моделирования представлены на рис. 3.24.

Ниже рассматриваемые проекты используют одночастотную синхронизацию (**варианты 7-9**).

Вариант 7. Линию задержки также можно реализовать на основе двухпортовой памяти с использованием мегафункции altshift_taps (Shift Register (RAM-based)) со следующими установками: число отводов – 4; дистанция между отводами – 4 (рис. 3.25). Коммутация отводов линии задержки осуществляется с помощью шинного мультиплексора четыре в один, на адресный вход которого подключается выход управляющего автомата sel[1..0]. В проекте используется одночастотная синхронизация и все блоки оригинального проекта fir_filter заменены.

ПЗУ для хранения коэффициентов фильтра реализуется на мегафункции ROM: 1 port с инициализацией. Мегафункция настроена так, что к адресной и выходной шинам данных подключены дополнительные регистры для организации конвейеризации. Такая настройка приводит к тому, что ПЗУ работает в синхронном режиме.

Мегафункция умножения и накопления ALTMULT_ACCUM представлена с конвейеризующими регистрами на входах и на выходах умножителя, с двумя регистрами на входе синхронной загрузки accum_sload.

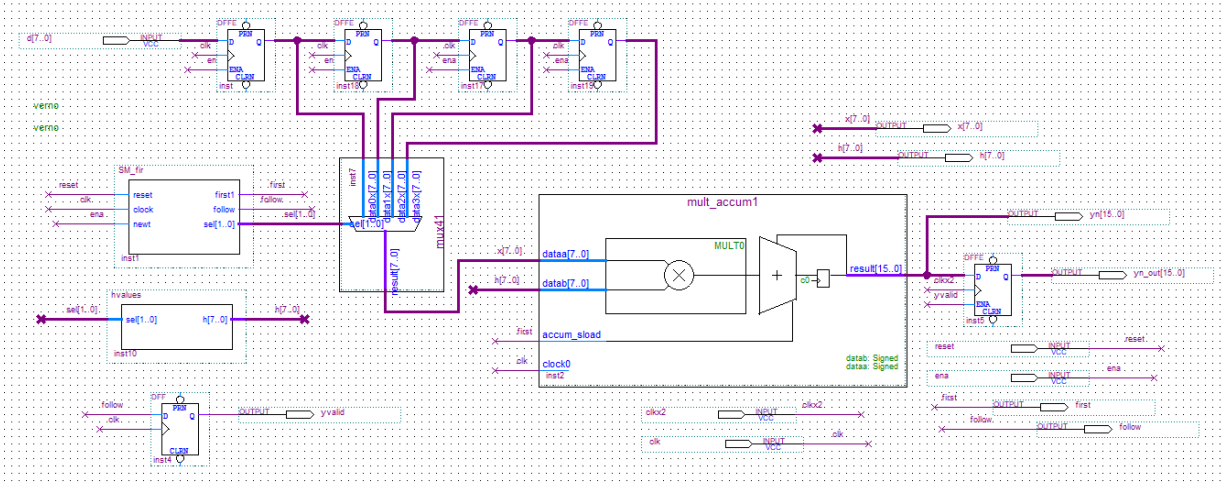


Рис. 3.23. Проект последовательного КИХ-фильтра на четыре отвода с использованием линии задержки на регистрах с мультиплексором, управляющий автомат (state flow), MAC-блок (мегафункция), асинхронное ПЗУ (оригинальный Verilog-код), двухчастотная синхронизация

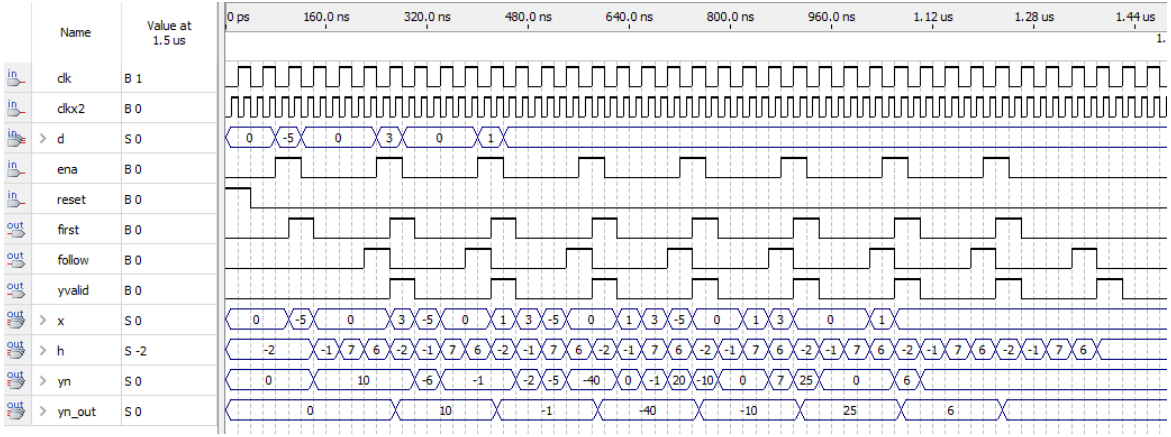


Рис. 3.24. Функциональное моделирование прохождения сигнала по структуре КИХ-фильтра с использованием линии задержки на регистрах

Учет латентности может быть осуществлен следующим образом. Сигнал first управляющего автомата напрямую подключается к входу accum_sload мегафункции ALTMULT_ACCUM. Сигнал follow нужно задержать на три такта синхросигнала, число триггеров определяется по числу уровней конвейеризации. В данных настройках мегафункции умножения и накопления используются три уровня. К выходной шине данных ПЗУ необходимо подключить два регистра и один регистр на вход линии задержки. В этом случае сигналы x[7..0] (номер отвода линии задержки зависит от значения на адресной шине мультиплексора) и h[7..0] (коэффициенты) перемножаются корректно. На рис. 3.26 показаны временные диаграммы.

Вариант 8. Проект отличается от варианта 7 тем, что в качестве управляющего автомата для генерации адресов используется 2-разрядный счетчик на 4 состояния (мегафункция lpm_counter) для адресации к мультиплексору и ПЗУ (рис. 3.27). А в качестве сигнала first задержанный на один такт синхронизации служит выход cout счетчика (сигнал сквозного переноса возникает, когда на выходах счетчика q[1..0] устанавливается число 3). Использование асинхронного ПЗУ приводит к тому, что для правильного умножения необходимо коэффициенты задержать на 4 такта с помощью 4 регистров. Функциональное моделирование прохождения сигнала по структуре КИХ-фильтра показано на рис. 3.28.

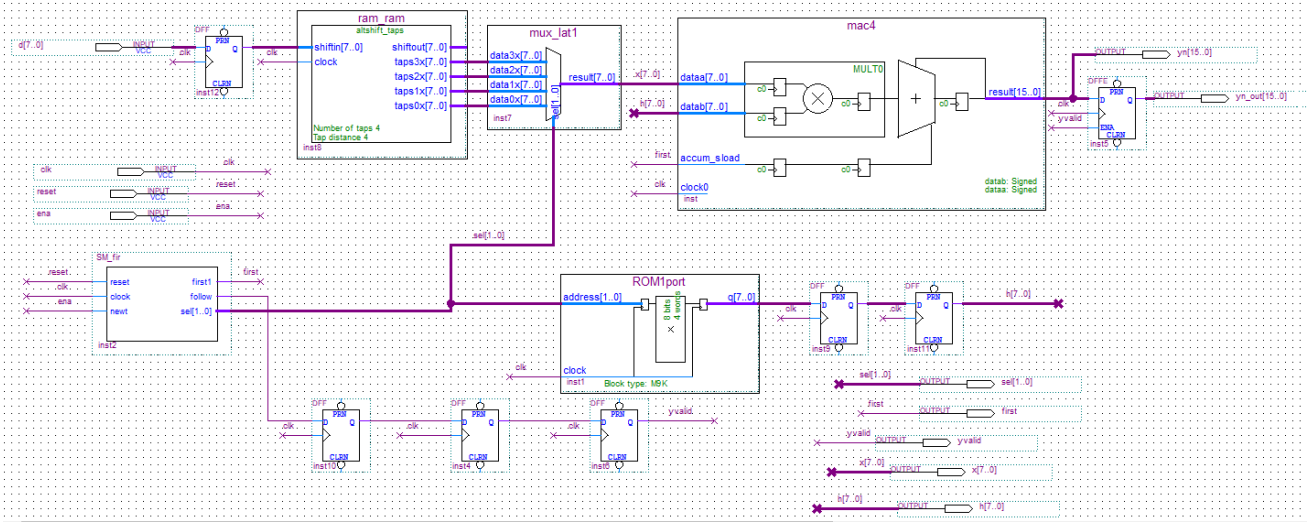


Рис. 3.25. Управляющий автомат (state flow), линия задержки на двухпортовой памяти (мегафункция Shift Register (RAM-based)), MAC-блок (мегафункция умножения и накопления ALTMULT_ACCUM), синхронное ПЗУ (мегафункция)

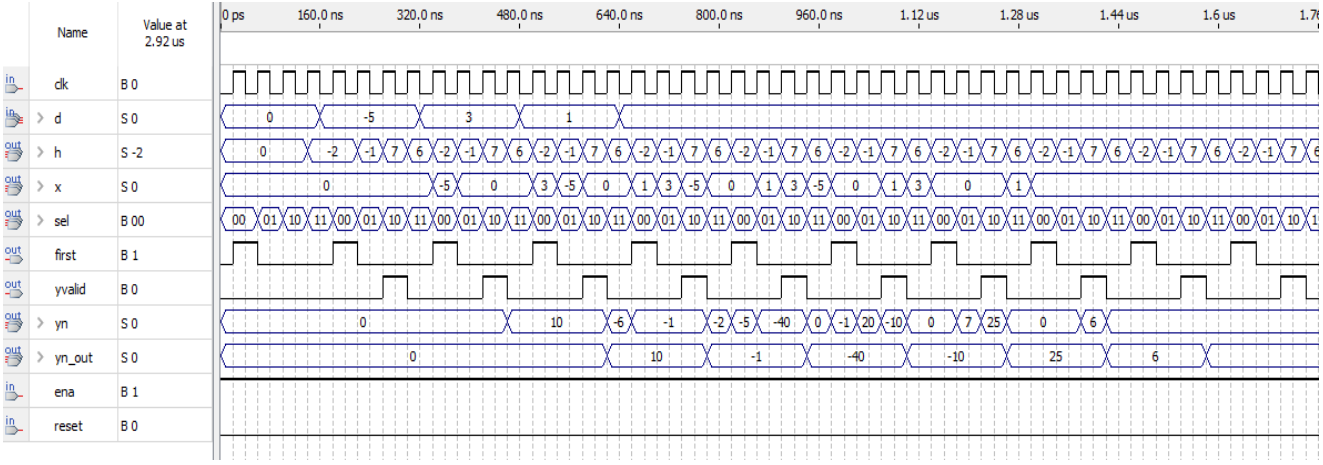


Рис. 3.26. Функциональное моделирование прохождения сигнала по структуре КИХ-фильтра с использованием линии задержки на двухпортовой памяти (мегафункция Shift Register (RAM-based))

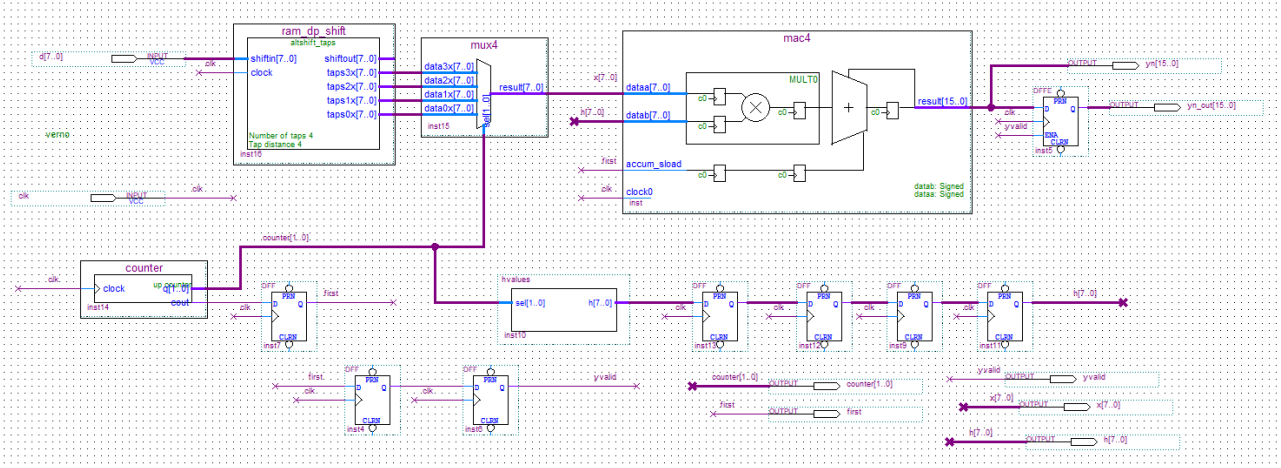


Рис. 3.27. Линия задержки на двухпортовой памяти (мегафункция Shift Register (RAM-based)), управляющий автомат – счетчик (мегафункция lpm_counter), асинхронное ПЗУ (оригинальный Verilog-код), MAC-блок (мегафункция умножения и накопления ALTMULT_ACCUM)

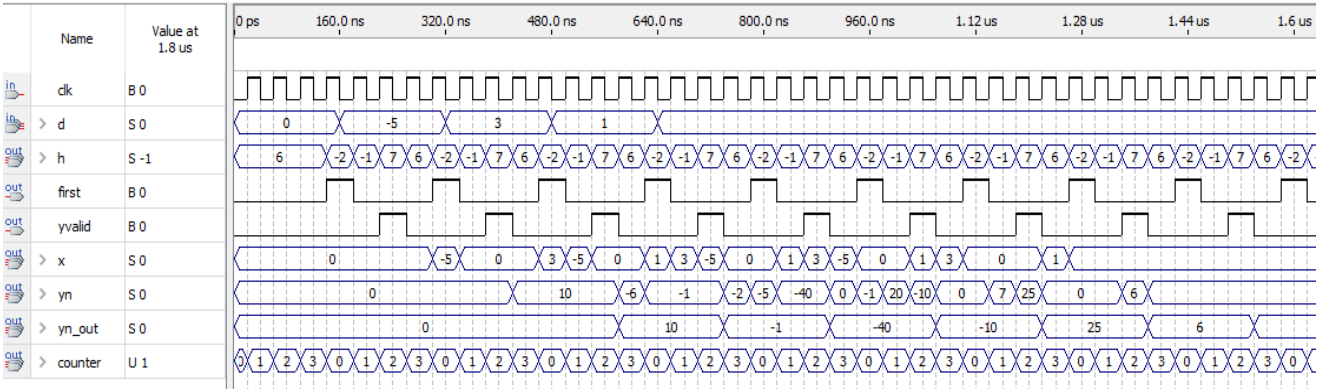


Рис. 3.28. Функциональное моделирование прохождения сигнала по структуре КИХ-фильтра с использованием в качестве управляющего автомата 2-разрядного счетчика на мегафункции lpm_counter

Вариант 9. **Последовательный КИХ-фильтр на четыре отвода с использованием линии задержки на двухпортовой памяти Simple Dual Port в режиме old memory contents appear, синхронное ПЗУ реализуется на мегафункции ROM: 1 port, с инициализацией.** Используется мегафункция умножения и накопления ALTMULT_ACCUM с регистрами на входах и на выходах умножителя, с регистром на входе синхронной загрузки accum_sload и дополнительным регистром на выходе сумматора (рис. 3.29).

В этом варианте линия задержки реализуется в виде циклического буфера (такое решение характерно для ЦОС-процессоров, например, вычислительный процессор микроконтроллера STM32F4 на ядре Cortex-M4) в ОЗУ, когда новые значения записываются в память на место старых. Блоки памяти ПЛИС фирмы Altera для отображения значений на выходной шине q в случае одновременного чтения и записи по одинаковому адресу могут работать в трех режимах new_data, old_data и don't_care. При выборе режима old_data на выходной шине q отображаются старые данные, хранящиеся в ОЗУ по конкретному адресу, прежде чем новые данные будут записаны по этому же адресу в память.

В режиме don't_care при одновременном чтении и записи по одинаковому адресу новые данные записываются в память, а на выходной шине q считываемые значения отображаются в виде символа 'x' (unknown values).

В режиме new_data новые данные записываются в память и одновременно отображаются на выходе блока памяти. Выбор режимов new_data, old_data и don't_care зависит от типа блока памяти ПЛИС. В данном проекте используются блоки памяти типа M9K.

Как и в примере 8 сигнал first получен задержкой сигнала с выхода cout. А сигнал yvalid получен задержкой сигнала first на 2 такта. Рисунок 3.30 подтверждает правильность работы КИХ-фильтра.

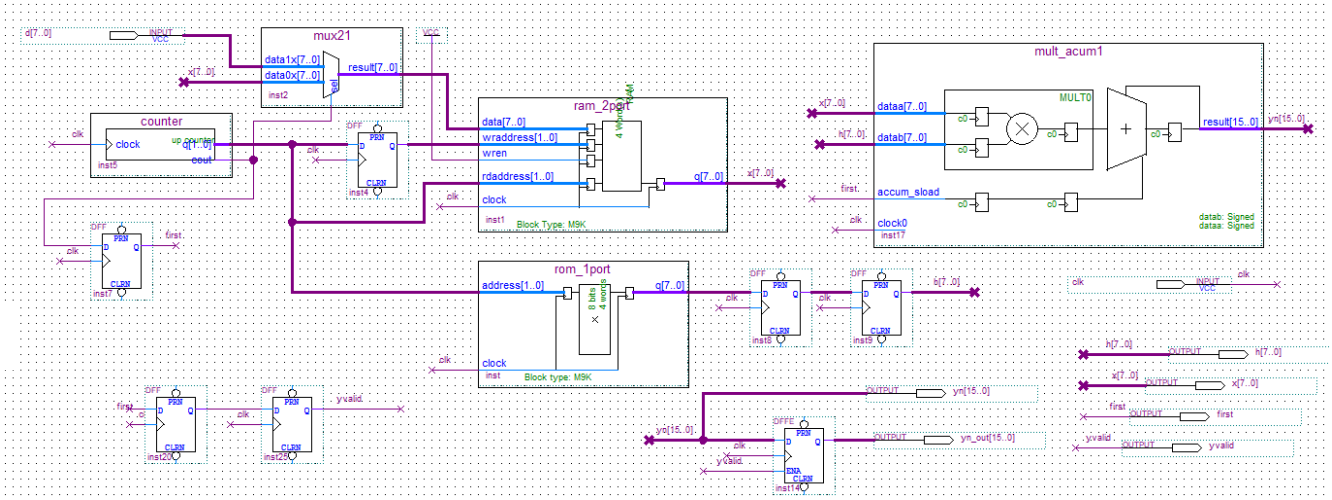


Рис. 3.29. Функциональная модель последовательного КИХ-фильтра на четыре отвода с использованием линии задержки на основе двухпортовой памяти, мегафункция RAM: 2-PORT и синхронного ПЗУ, мегафункции ROM: 1 port

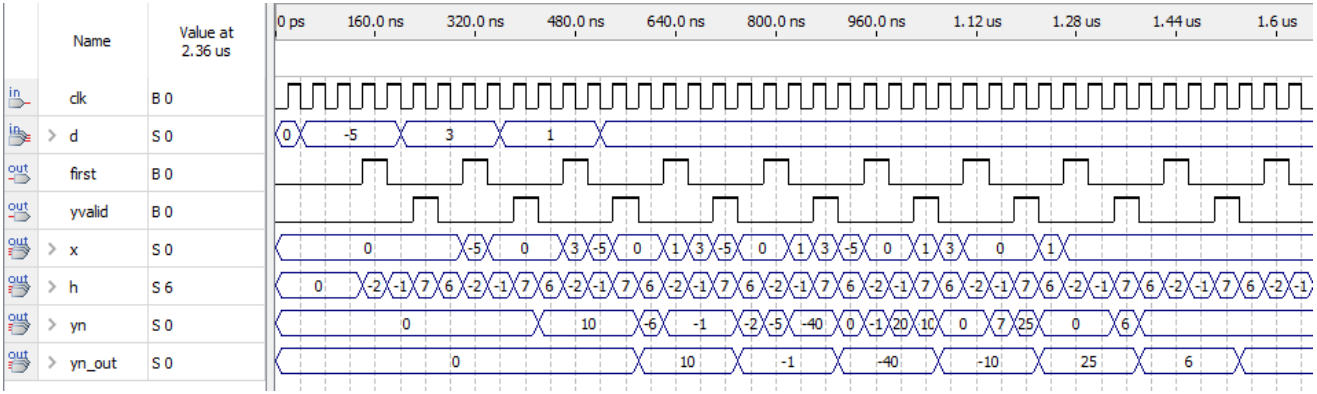


Рис. 3.30. Функциональное моделирование прохождения сигнала по структуре КИХ-фильтра с использованием линии задержки на основе двухпортовой памяти, мегафункция RAM: 2-PORT и синхронного ПЗУ, мегафункции ROM: 1 port

В табл. 3.1 представлены результаты размещения проектов в базис ПЛИС Cyclone IV EP4CGX22CF19C6 и оценки быстродействия (максимальная частота переключения триггеров, Fmax), полученные с помощью TimeQuest без учета временных ограничений (без sdc-файла). По объему проекты 1 – 9 занимают менее 1% логических и аппаратных ресурсов ПЛИС.

Для вариантов 1 – 6 с двухчастотной (clk и clkx2) и для вариантов 7 – 9 с одночастотной (clk) синхронизацией без использования заранее созданного пользователем scd-файла САПР Quartus II автоматически запускает Classic Timing Analyzer и в качестве требования к синхронизации по умолчанию применяет назначение FMAX_REQUIREMENT – 1000 МГц (типовая полоса обрабатываемого радиосигнала или производительность, характерная для ЦОС-процессоров: частота 1 ГГц/4 такта = 250 MSPS).

На рис. 3.31 показан отчет с отрицательными запасами по времени (Slack) по параметру Minimum Pulse Width ‘clk’ модели Slow при условиях 1200 мВ и 0 ^{0}C для варианта 3 при работе на заданной частоте 1000 МГц. Следовательно, необходимо ограничить частоты синхронизации до появления положительных запасов по времени.

Для вариантов 1 – 6 с двухчастотной синхронизацией (clk и clkx2) используем следующий scd-файл с частотами 100 МГц (период 10 нс) для порта clk и 200 МГц (период 5 нс) для порта clkx2.

Create_clock –name {clk} –period 10.000 –waveform { 0.000 2.000 } [get_ports {clk}]

create_clock –name {clkx2} –period 5.000 –waveform { 0.000 2.000 } [get_ports {clkx2}]

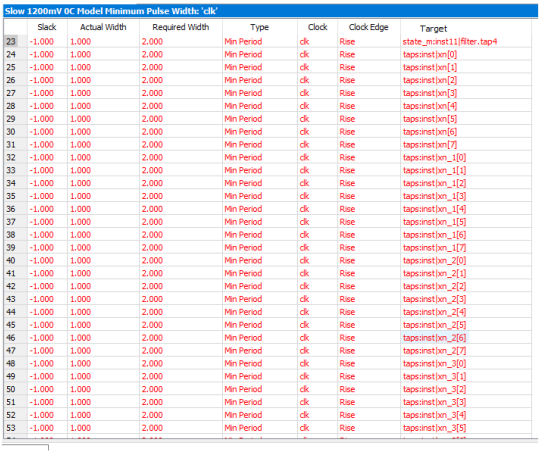
Тестирование для вариантов 1 – 6 с частотами 100 и 200 МГц показало наличие положительных запасов по параметру

Minimum Pulse Width ‘clk’. В качестве примера на рис. 3.32 показан отчет с положительными запасами по параметру Minimum Pulse Width ‘clk’ модели Slow при условиях 1200 мВ и $0\ ^0C$ для варианта 3.

Таблица 3.1

Задействовано логических и аппаратных ресурсов (1 умножитель с размерностью операндов 9×9) ПЛИС Cyclone IV EP4CGX22CF19C6

Варианты	Логических элементов/ регистров	Оценка быстродействия для модели Slow при условиях 1200 мВ и $0\ ^0C$, Fmax, МГц / Fmax в наихудшем случае, МГц при clk и clkx2 1000 МГц	Оценка быстродействия для модели Slow при условиях 1200 мВ и $85\ ^0C$, Fmax, МГц / Fmax в наихудшем случае, МГц при clk и clkx2 1000 МГц
Оригинальный проект fir_filter	93/77	273/250	242/242
1	99/96	180/180	162/180
2	72/72	206/206	185/185
3	70/70	145/145	129/129
4	72/70	144/144	129/129
5	86/70	142/142	128/128
6	75/70	145/145	130/130
7	85/70 96 бит встроенной памяти	348/250 (295/250 при clk 100 МГц)	337/250 (267/250 при clk 100 МГц
8	62/46, 96 бит встроенной памяти	328/250 (305/250 при clk 100 МГц)	296/250 (278/250 при clk 100 МГц)
9	63/55, 96 бит встроенной памяти	348/250 (398/250 при clk 100 МГц)	343/250 (359/250 при clk 100 МГц)



Slow 1200mV 0C Model Minimum Pulse Width: 'clk'

	Slack	Actual Width	Required Width	Type	Clock	Clock Edge	Target
23	-1.000	1.000	2.000	Min Period	clk	Rise	state_m:inst11|filter.tap4
24	-1.000	1.000	2.000	Min Period	clk	Rise	taps:inst|xn[0]
25	-1.000	1.000	2.000	Min Period	clk	Rise	taps:inst|xn[1]
26	-1.000	1.000	2.000	Min Period	clk	Rise	taps:inst|xn[2]
27	-1.000	1.000	2.000	Min Period	clk	Rise	taps:inst|xn[3]
28	-1.000	1.000	2.000	Min Period	clk	Rise	taps:inst|xn[4]
29	-1.000	1.000	2.000	Min Period	clk	Rise	taps:inst|xn[5]
30	-1.000	1.000	2.000	Min Period	clk	Rise	taps:inst|xn[6]
31	-1.000	1.000	2.000	Min Period	clk	Rise	taps:inst|xn[7]
32	-1.000	1.000	2.000	Min Period	clk	Rise	taps:inst|xn_1[0]
33	-1.000	1.000	2.000	Min Period	clk	Rise	taps:inst|xn_1[1]
34	-1.000	1.000	2.000	Min Period	clk	Rise	taps:inst|xn_1[2]
35	-1.000	1.000	2.000	Min Period	clk	Rise	taps:inst|xn_1[3]
36	-1.000	1.000	2.000	Min Period	clk	Rise	taps:inst|xn_1[4]
37	-1.000	1.000	2.000	Min Period	clk	Rise	taps:inst|xn_1[5]
38	-1.000	1.000	2.000	Min Period	clk	Rise	taps:inst|xn_1[6]
39	-1.000	1.000	2.000	Min Period	clk	Rise	taps:inst|xn_1[7]
40	-1.000	1.000	2.000	Min Period	clk	Rise	taps:inst|xn_2[0]
41	-1.000	1.000	2.000	Min Period	clk	Rise	taps:inst|xn_2[1]
42	-1.000	1.000	2.000	Min Period	clk	Rise	taps:inst|xn_2[2]
43	-1.000	1.000	2.000	Min Period	clk	Rise	taps:inst|xn_2[3]
44	-1.000	1.000	2.000	Min Period	clk	Rise	taps:inst|xn_2[4]
45	-1.000	1.000	2.000	Min Period	clk	Rise	taps:inst|xn_2[5]
46	-1.000	1.000	2.000	Min Period	clk	Rise	taps:inst|xn_2[6]
47	-1.000	1.000	2.000	Min Period	clk	Rise	taps:inst|xn_2[7]
48	-1.000	1.000	2.000	Min Period	clk	Rise	taps:inst|xn_3[0]
49	-1.000	1.000	2.000	Min Period	clk	Rise	taps:inst|xn_3[1]
50	-1.000	1.000	2.000	Min Period	clk	Rise	taps:inst|xn_3[2]
51	-1.000	1.000	2.000	Min Period	clk	Rise	taps:inst|xn_3[3]
52	-1.000	1.000	2.000	Min Period	clk	Rise	taps:inst|xn_3[4]
53	-1.000	1.000	2.000	Min Period	clk	Rise	taps:inst|xn_3[5]

Рис. 3.31. Отрицательные запасы по параметру Minimum Pulse Width ‘clk’ модели Slow при условиях 1200 мВ и 0 ^{0}C для варианта 3

Однако для двухчастотной синхронизации, при использовании sdc-файла, оценки Fmax приложение для временного анализа TimeQuest уже не вычисляет в отличие от одночастотной. В табл. 3.1 для вариантов 7 – 9 дополнительно показаны оценки быстродействия для одночастотной синхронизации:

create_clock -name {clk} -period 10.000 -waveform { 0.000 2.000 } [get_ports {clk}]

По своей структуре последовательные МАС КИХ-фильтры в базисах ПЛИС Xilinx и Altera схожи, отличаются лишь разной латентностью работы основных узлов фильтра реализуемых на блоках встроенной памяти и ЦОС-блоках. Отличительной особенностью от параллельных КИХ-фильтров является наличие управляющего автомата, который адресуется к линии задержки и к ПЗУ для хранения коэффициентов фильтра, а также управляет МАС-блоком и регистром результата.

Slow 1200mV 0C Model Minimum Pulse Width: 'clk'

	Slack	Actual Width	Required Width	Type	Clock	Clock Edge	Target
1	1.792	2.008	0.216	High Pulse Width	clk	Rise	taps:inst|xn[0]
2	1.792	2.008	0.216	High Pulse Width	clk	Rise	taps:inst|xn[1]
3	1.792	2.008	0.216	High Pulse Width	clk	Rise	taps:inst|xn[2]
4	1.792	2.008	0.216	High Pulse Width	clk	Rise	taps:inst|xn[3]
5	1.792	2.008	0.216	High Pulse Width	clk	Rise	taps:inst|xn[4]
6	1.792	2.008	0.216	High Pulse Width	clk	Rise	taps:inst|xn[5]
7	1.792	2.008	0.216	High Pulse Width	clk	Rise	taps:inst|xn[6]
8	1.792	2.008	0.216	High Pulse Width	clk	Rise	taps:inst|xn[7]
9	1.792	2.008	0.216	High Pulse Width	clk	Rise	taps:inst|xn_1[0]
10	1.792	2.008	0.216	High Pulse Width	clk	Rise	taps:inst|xn_1[1]
11	1.792	2.008	0.216	High Pulse Width	clk	Rise	taps:inst|xn_1[2]
12	1.792	2.008	0.216	High Pulse Width	clk	Rise	taps:inst|xn_1[3]
13	1.792	2.008	0.216	High Pulse Width	clk	Rise	taps:inst|xn_1[4]
14	1.792	2.008	0.216	High Pulse Width	clk	Rise	taps:inst|xn_1[5]
15	1.792	2.008	0.216	High Pulse Width	clk	Rise	taps:inst|xn_1[6]
16	1.792	2.008	0.216	High Pulse Width	clk	Rise	taps:inst|xn_1[7]
17	1.792	2.008	0.216	High Pulse Width	clk	Rise	taps:inst|xn_2[0]
18	1.792	2.008	0.216	High Pulse Width	clk	Rise	taps:inst|xn_2[1]
19	1.792	2.008	0.216	High Pulse Width	clk	Rise	taps:inst|xn_2[2]
20	1.792	2.008	0.216	High Pulse Width	clk	Rise	taps:inst|xn_2[3]
21	1.792	2.008	0.216	High Pulse Width	clk	Rise	taps:inst|xn_2[4]
22	1.792	2.008	0.216	High Pulse Width	clk	Rise	taps:inst|xn_2[5]
23	1.792	2.008	0.216	High Pulse Width	clk	Rise	taps:inst|xn_2[6]
24	1.792	2.008	0.216	High Pulse Width	clk	Rise	taps:inst|xn_2[7]
25	1.792	2.008	0.216	High Pulse Width	clk	Rise	taps:inst|xn_3[0]
26	1.792	2.008	0.216	High Pulse Width	clk	Rise	taps:inst|xn_3[1]
27	1.792	2.008	0.216	High Pulse Width	clk	Rise	taps:inst|xn_3[2]
28	1.792	2.008	0.216	High Pulse Width	clk	Rise	taps:inst|xn_3[3]
29	1.792	2.008	0.216	High Pulse Width	clk	Rise	taps:inst|xn_3[4]
30	1.792	2.008	0.216	High Pulse Width	clk	Rise	taps:inst|xn_3[5]

Рис. 3.32. Положительные запасы по параметру Minimum Pulse Width ‘clk’ модели Slow при условиях 1200 мВ и 0 ^{0}C для варианта 3

В разделе представлено 9 вариантов реализации проекта последовательного КИХ-фильтра от простого до сложного с использованием циклического буфера в двухпортовой памяти ОЗУ, синхронного ПЗУ и конвейризованного МАС-блока. Проекты задействуют примерно одинаковое количество логических и аппаратных ресурсов ПЛИС Cyclone IV EP4CGX22CF19C6 (менее 1%), но имеют разную оценку быстродействия. Наивысшее быстродействие показывают проекты 7, 8 и 9 с использованием конвейеризации и блочной памяти ПЛИС для организации линии задержки. Для двух- (clk=100 и clkx2=200 МГц) и одночастотной (clk=100 МГц) синхронизаций при использовании файлов временных ограничений обеспечены положительные запасы на рассматриваемых частотах.

3.2. Проектирование последовательных КИХ-фильтров в системе визуально-имитационного моделирования M*atlab*/Simulink с использованием Altera DSP Builder и MAC-блоков

Пакет Altera DSP Builder работает в связке с САПР Quartus II по аналогии с пакетом System Generator IDS и САПР ПЛИС ISE фирмы Xilinx. Программные пакеты расширения Altera DSP Builder и System Generator IDS системы визуально-имитационного моделирования M*atlab*/Simulink (версия 8.3, сборка R2014a) обеспечивают высокоуровневое оптимизированное HDL-представление проектов с автоматическим компилированием в базис ПЛИС Xilinx и Altera с последующим созданием испытательных стендов.

Рассмотрим разработку последовательного КИХ-фильтра на 4 отвода с использованием линии задержки на основе двухпортовой памяти и одного блока умножения и накопления (1 MAC-блок) в Altera DSP Builder и САПР Quartus II: $y = C_0x_0 + C_1x_1 + C_2x_2 + C_3x_3$. Предположим, что коэффициенты фильтра известны: $C_0 = -2$, $C_1 = -1$, $C_2 = 7$ и $C_3 = 6$. Такие фильтры еще называют MAC-фильтрами.

За основу проектируемого КИХ-фильтра возьмем имитационную модель с именем FIR_MAC32.mdl КИХ-фильтра на 32 отвода (входной сигнал и коэффициенты фильтра представлены с 17-разрядной точностью, частота дискретизации входного сигнала 1 МГц) из справочной системы САПР Quartus II по адресу:

quartus\dsp_builder\DesignExamples\Demos\Filters\Mac32.

Рассмотрим случай, когда на вход фильтра поступает сигнал –5, 3, 1, 0, 0 и 0 т.д. Правильные значения на выходе фильтра: 10, –1, –40, –10, 26, 6 и 0 т.д.

Проверить правильность расчетов профильтрованных значений позволяет использование функции дискретной фильтрации filter, где [-2,-1,7,6] – вектор коэффициентов нерекурсивной части, 1 – вектор коэффициентов рекурсивной части (в нашем случае они раны нулю), а x=[-5,3,1,0,0,0] сигнал, подлежащий фильтрации.

```
x=[-5,3,1,0,0,0];
y = filter([-2,-1,7,6],1,x);
```

Это позволит быстро рассчитать правильные значения профильтрованного сигнала, т.е. получить вектор [10 -1 -40 -10 25 6], который будет использоваться в дальнейшем для проверки работы КИХ-фильтра.

На рис. 3.33 представлена имитационная модель последовательного КИХ-фильтра на 4 отвода для реализации в базисе ПЛИС Altera серии Cyclone III на основе одного MAC-блока. Модель работает с целыми десятичными числами как со знаком, так и без, которые с помощью настроек функциональных блоков преобразуются в двоичный формат для последующего функционального моделирования.

Имитационная модель состоит из следующих функциональных блоков: счетчика (управляющий автомат); линии задержки на базе двухпортовой памяти (ОЗУ) и вспомогательного мультиплексора; LUT (блок памяти в режиме ПЗУ) для хранения коэффициентов фильтра; MAC-блока и регистра для хранения результата; компилятора сигналов; генератора испытательных стендов; формирования синхросигнала.

Функциональный блок BaseClock позволяет в автоматическом режиме формировать синхросигнал и асинхронный сигнал сброса (активный низкий) при последующем переходе от имитационной модели к функциональной в САПР ПЛИС Altera Quartus II ver 12.1. Для функционального моделирования задается синхросигнал с периодом 20 нс и временной шаг симуляции в M*atlab*/Simulink 0.25 с. Шаг симуляции выбирается исходя из следующих соображений, что профильтрованные данные на выходе

фильтра должны обновляться через 4 такта синхроимпульса (рис. 3.34).

Функциональный блок Increment Decrement представляет 2-разрядный суммирующий счетчик (увеличивает свое содержимое на единицу) (рис. 3.35). Играет роль простейшего управляющего автомата. Счетчик управляет работой линией задержки на четыре отвода через функциональный блок «дешифратор» (рис. 3.36).

«Дешифратор» фактически представляет собой компаратор, который сравнивает входной сигнал с заданной декодируемой величиной. Дешифратор вырабатывает активный сигнал (десятичная единица) через каждые три последующих временных шага моделирования (на четвертый шаг), который подключен к входу ShiftEn линии задержки.

Сигнал ShiftEn является адресным входом мультиплексора 2 в 1. При ShiftEn=1 происходит запись десятичных чисел, поступающих на вход фильтра в ОЗУ линии задержки, а при ShiftEn=0 происходит перезапись содержимого ОЗУ.

Тем самым моделируется работа линии задержки на базе сдвигового регистра. И можно говорить, что ОЗУ работает в режиме циклического буфера. Счетчик также адресуется к данным хранящимся в функциональном блоке LUT. LUT можно рассматривать как таблицу для хранения коэффициентов фильтра. Играет роль блока памяти (ПЗУ).

Линия задержки на основе двухпортовой памяти и мультиплексора показана на рис. 3.37. Обратная связь в линии задержки реализуется путем подачи данных с выходной шины ОЗУ на вход через мультиплексор, на другой вход которого подается сигнал, подлежащий фильтрации.

Двухпортовая память позволяет одновременно считывать информацию из ОЗУ и записывать ее по разным адресам одновременно. Изменения на шине rd_add(1:0) связаны с чтением содержимого ОЗУ и доступны на выходе, а на шине wr_add(1:0) – с записью информации в ОЗУ. В рассматриваемом примере операции чтения и записи

информации в ОЗУ разделены во времени элементом задержки. По умолчанию первоначально память инициализируется вектором [0 0 0 0]. На рис. 3.38 показаны настройки функционального блока Dual-Port RAM.

Настройки функционального блока LUT показаны на рис. 3.39. Выходные шины данных функциональных блоков Dual-Port RAM и LUT должны быть регистерные (рис. 3.38б и рис. 3.39б).

Это обеспечивает правильность функционирования блока MultiplyAccumulate. Настройки функционального блока MultiplyAccumulate, выполняющего роль операции умножения с накоплением, показаны на рис. 3.40. Информационные потоки вычислений в структуре КИХ-фильтра показаны на рис. 3.41.

На рис. 3.42 показано имитационное моделирование в системе M*atlab*/Simulink КИХ-фильтра на четыре отвода. На вход фильтра поступает сигнал –5, 3, 1, 0, 0, 0 и т.д. Правильные значения на выходе фильтра: 10, –1, –40, –10, 26, 6, 0 и т.д.

Работа с закладкой Simple компилятора сигналов (функциональный блок SignalCompiler). При компиляции используются мегафункции с применением языка AHDL, поэтому проект будет состоять из разнородных файлов с расширениями .tdf, .vhd и др.

Поскольку в САПР версии Altera Quartus 12.1 сборка 177 отсутствует встроенный векторный редактор, воспользуемся версией Altera Quarus II 13.1.

На рис. 3.43 показано функциональное моделирование КИХ-фильтра на 4 отвода в САПР Quartus II 13.1. На вход фильтра поступает сигнал –5, 3, 1, 0, 0, 0 и т.д. Правильные значения на выходе фильтра: 10, –1, –40, –10, 26, 6, 0 и т.д. Сравнивая рис. 3.42 и 3.43, можно сделать вывод, что имитационная и функциональная модели фильтров работают корректно.

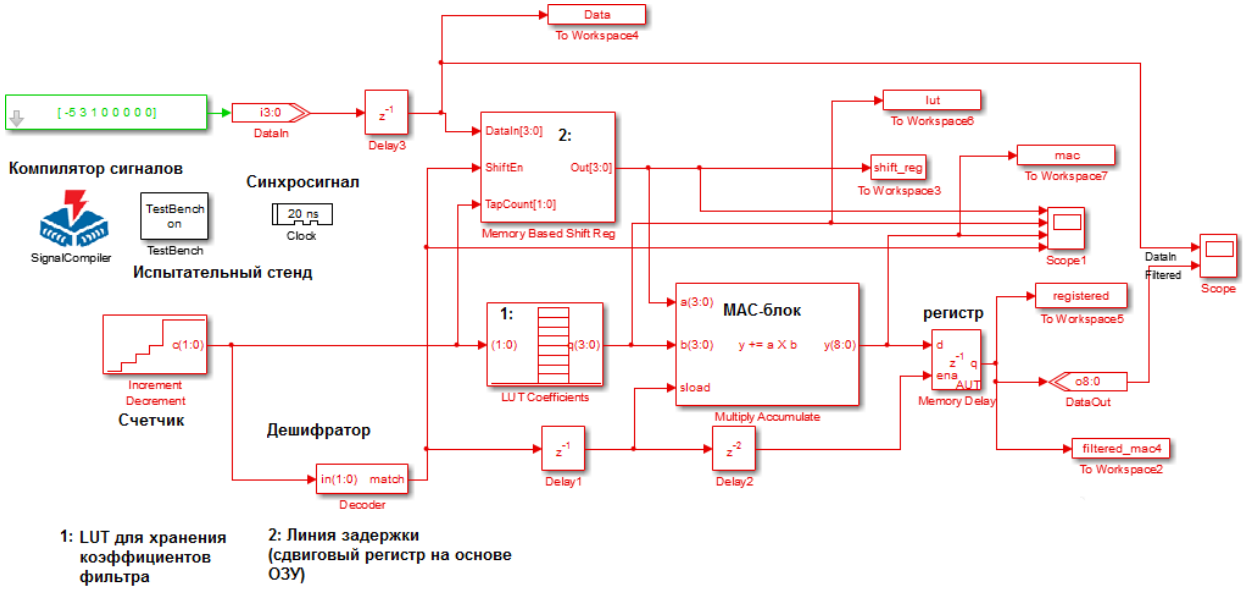


Рис. 3.33. Имитационная модель последовательного КИХ-фильтра на 4 отвода для реализации в базисе ПЛИС Altera серии Cyclone на основе одного МАС-блока

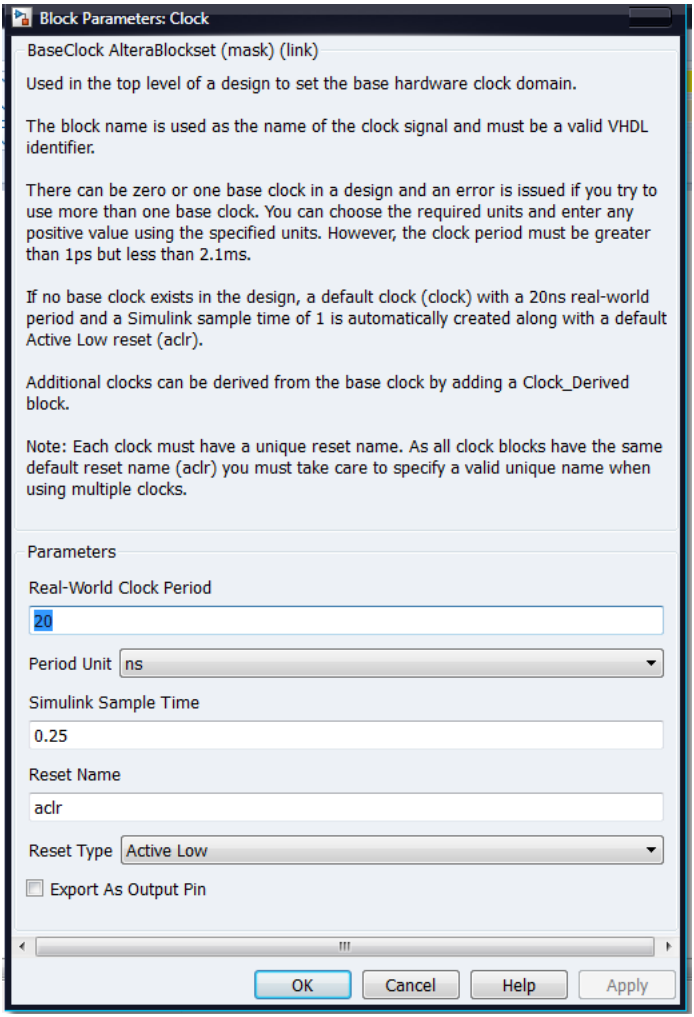


Рис. 3.34. Настройки функционального блока BaseClock. Задается период синхросигнала 20 нс для функционального моделирования в САПР Quartus II ver 12.1 и временной шаг симуляции в M*atlab*/Simulink 0.25 с

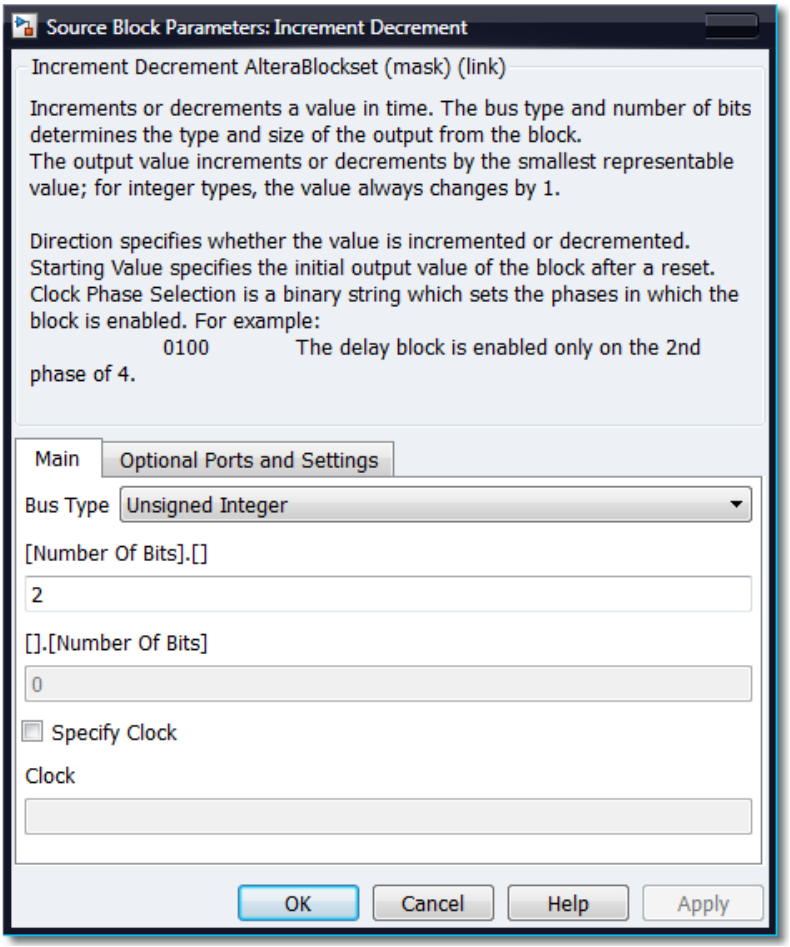


Рис. 3.35. Настройки функционального блока Increment Decrement

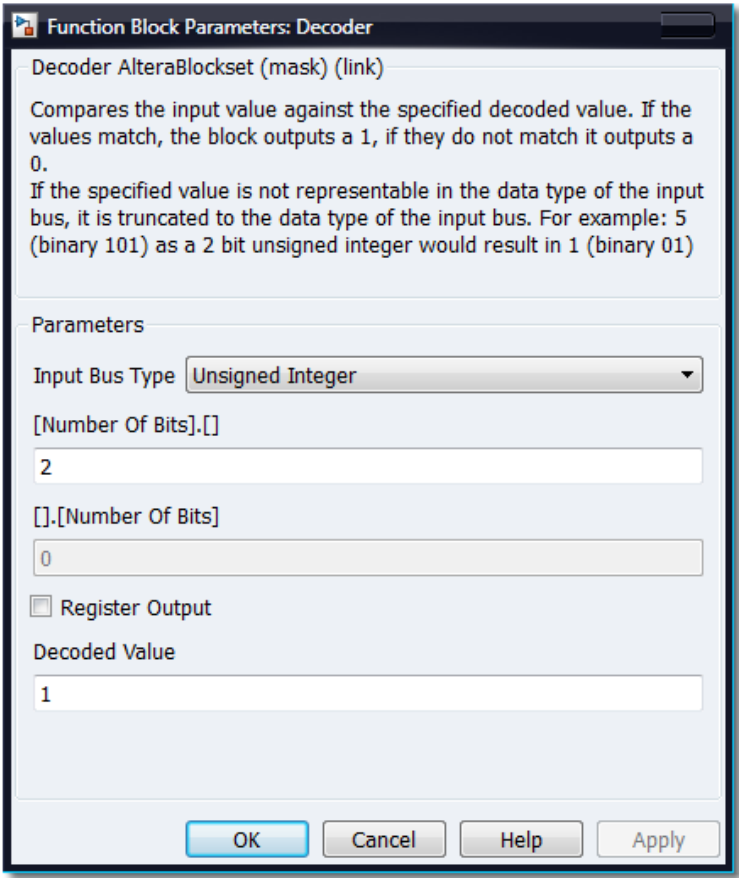


Рис. 3.36. Функциональный блок Decoder. Задается 2-разрядная входная шина типа Unsigned Integer и декодируемое значение – десятичная единица

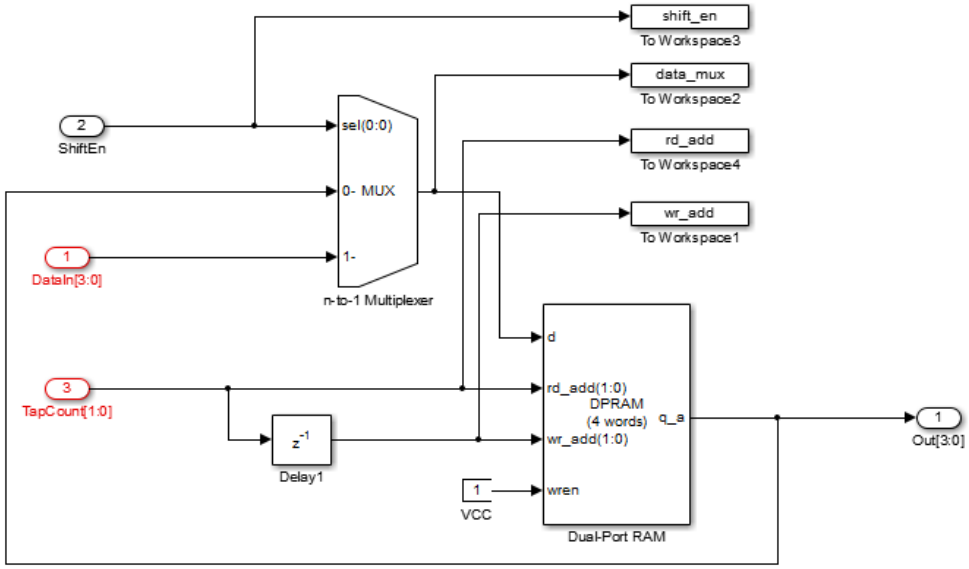
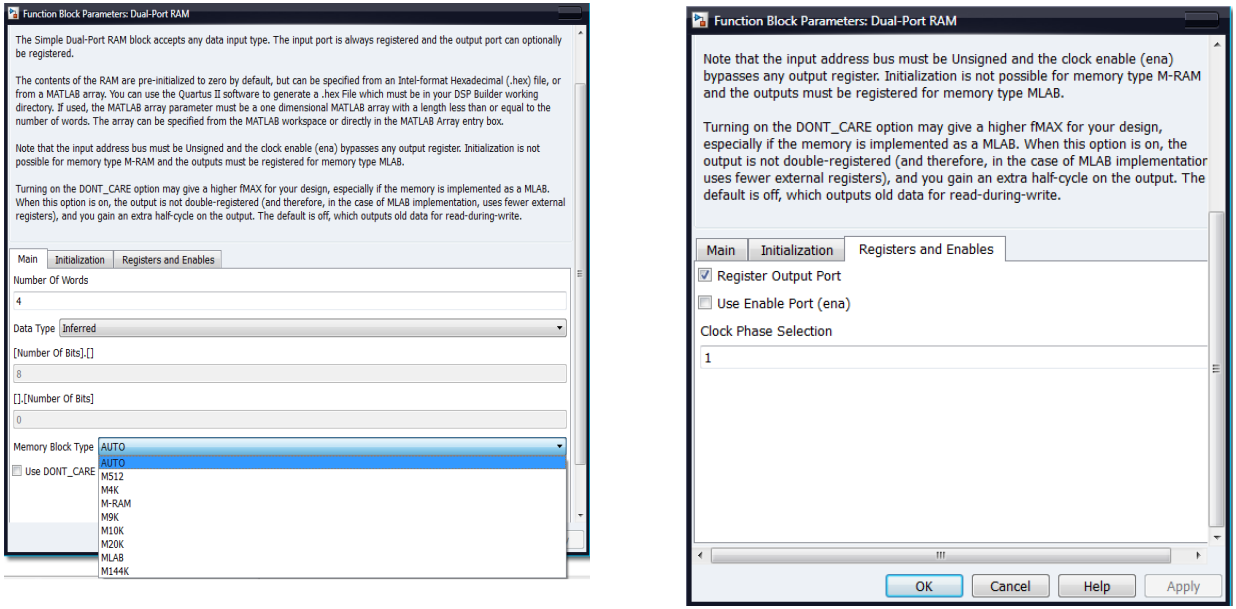
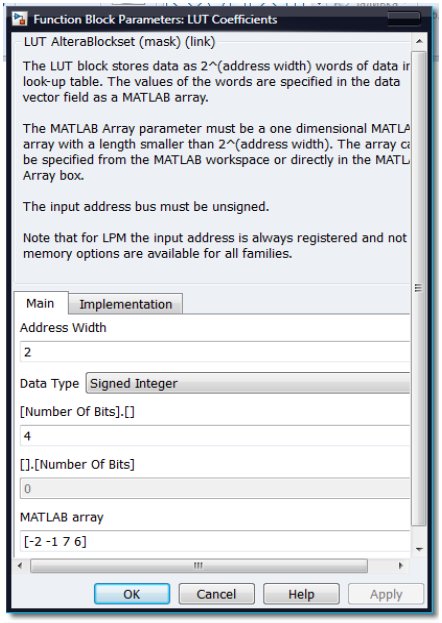


Рис. 3.37. Линия задержки на основе двухпортовой памяти (функциональный блок Dual-Port RAM) и мультиплексора (функциональный блок Multiplexer)

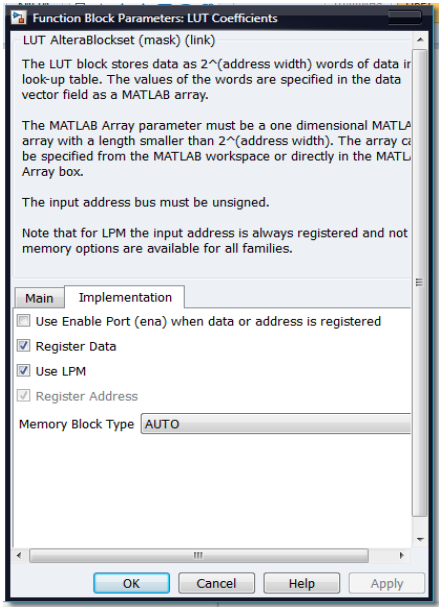


а) б)

Рис. 3.38. Настройки функционального блока Dual-Port RAM: а) закладка Main (задаются четыре 8-разрядных слова, тип памяти Auto); б) закладка Registers and Enables (выходной порт шины данных ОЗУ регистерный)



а)



б)

Рис. 3.39. Настройки функционального блока LUT: а) закладка Main (задается 2-разрядная адресная шина блока памяти и вектор коэффициентов фильтра [– 2 – 1 7 6], целые десятичные числа со знаком представляются с 4-битной точностью); б) закладка Implementation (выходная шина данных блока памяти должна быть регистерная, опция Register Data)

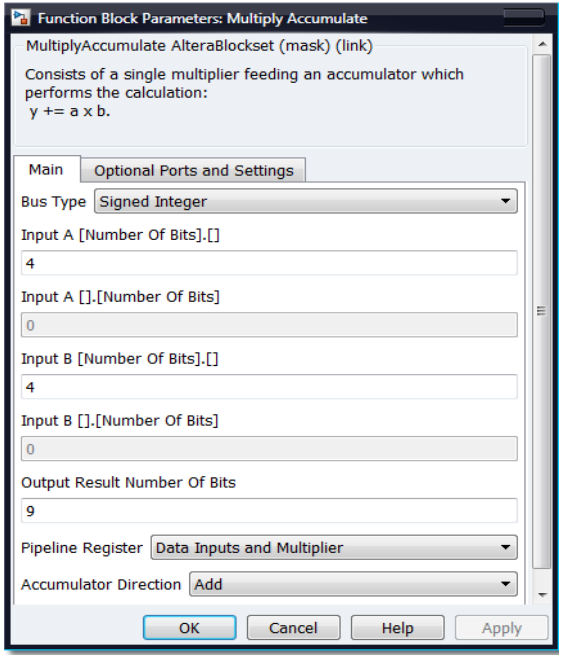


Рис. 3.40. Настройки функционального блока выполняющего роль операции умножения с накоплением (MultiplyAccumulate). Задается разрядность шин A и B (4 бита, целые числа со знаком) и точность представления результата вычисления (9 бит)

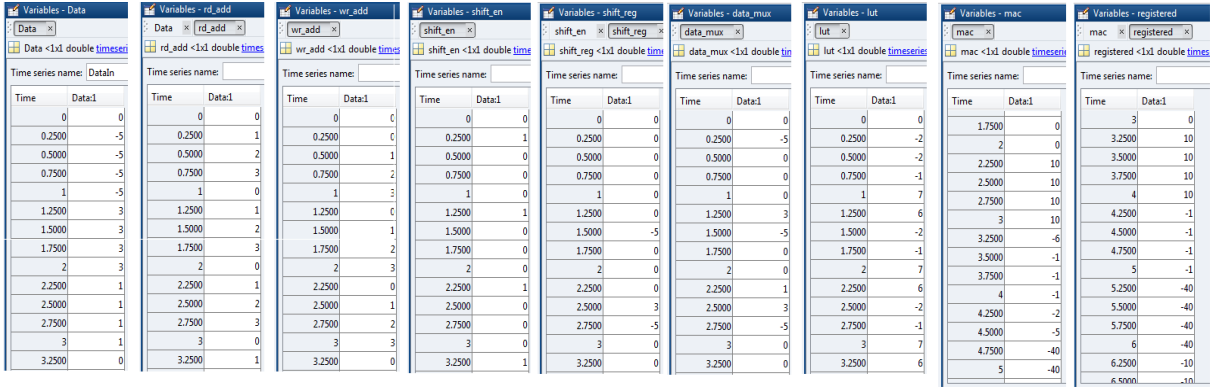


Рис. 3.41. Информационные потоки: а) сигнал, подлежащий фильтрации после операции задержки; б) и в) rd_add и wr_add – адреса на шинах, связанных с операциями чтения и записи информации в ОЗУ; г) shift_en – сигнал разрешения записи новых значений сигнала, подлежащего фильтрации в линию задержки; д) shift_reg – значения на выходной шине ОЗУ (шина q_a, рис. 6.9); е) data_mux – сигнал на выходе мультиплексора или значения на входной шине данных ОЗУ (шина d, рис. 6.9); к) lut – коэффициенты КИХ-фильтра; л) – mac – значения сигнала после операции умножения и накопления; м) registered – профильтрованные значения. Обозначение сигналов согласно рис. 3.33 и рис. 3.37

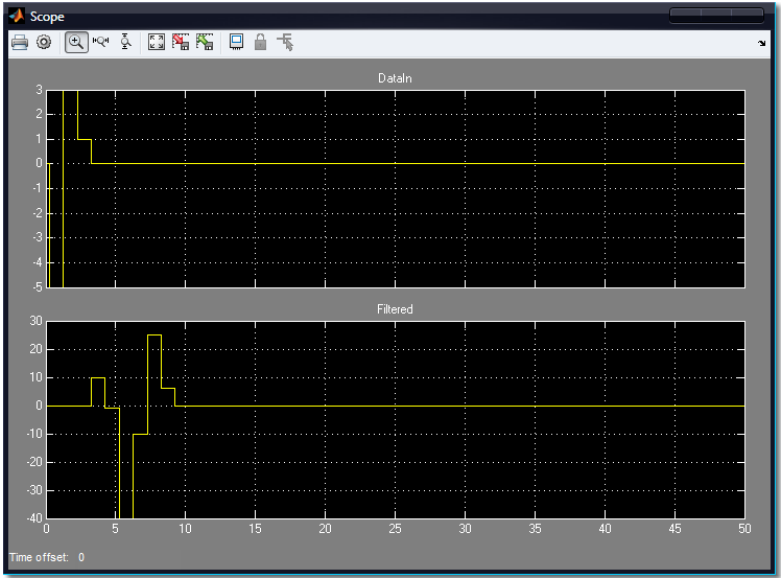


Рис. 3.42. Имитационное моделирование в системе M*atlab*/Simulink КИХ-фильтра на 4 отвода. На вход фильтра поступает сигнал –5, 3, 1, 0, 0, 0 и т.д. Правильные значения на выходе фильтра: 10, –1, –40, –10, 26, 6, 0 и т.д.

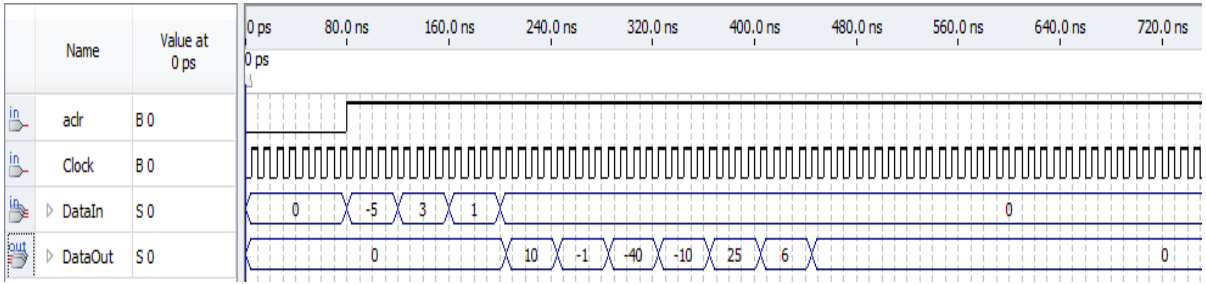


Рис. 3.43. Функциональное моделирование КИХ-фильтра на 4 отвода в САПР Quartus II 13.1. На вход фильтра поступает сигнал –5, 3, 1, 0, 0, 0 и т.д. Правильные значения на выходе фильтра: 10, –1, –40, –10, 26, 6, 0 и т.д.

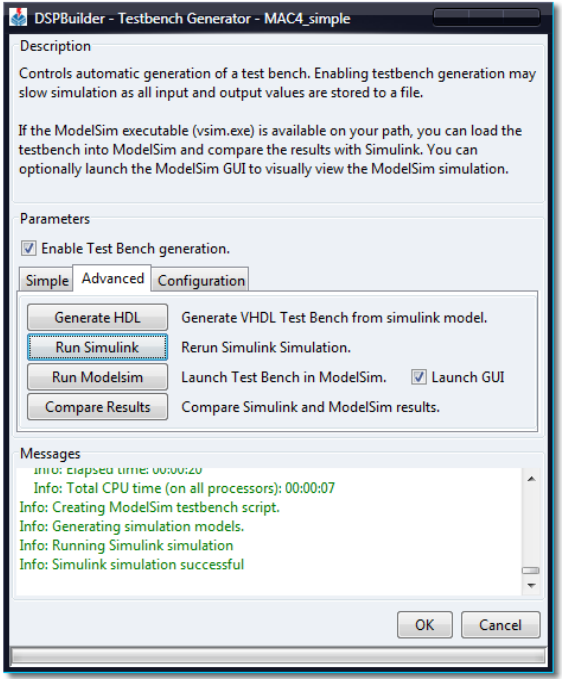


Рис. 3.44. Генератор испытательных стендов (функциональный блок Testbench Cenerator)

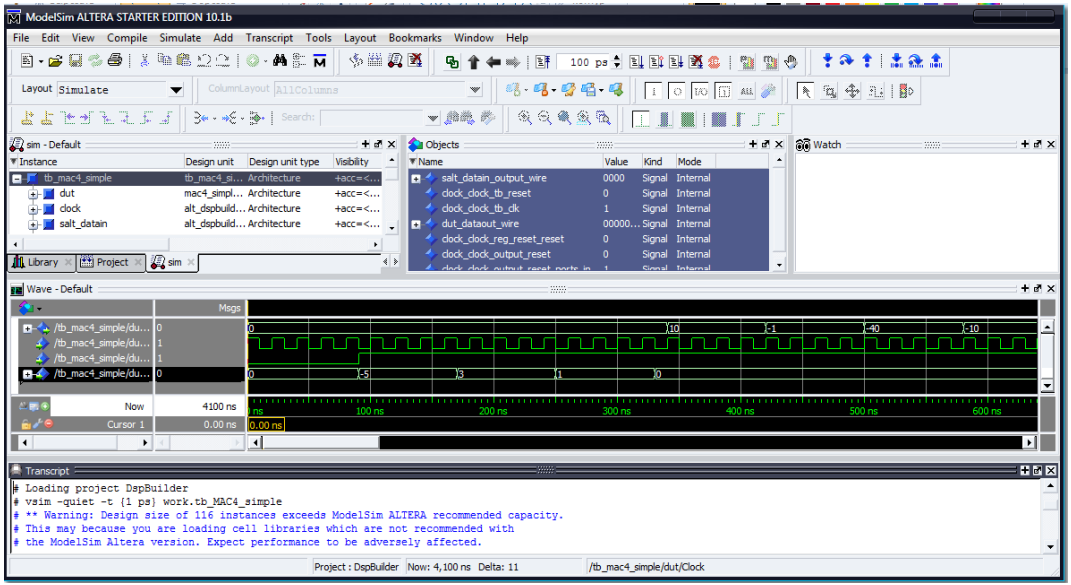


Рис. 3.45. Функциональное моделирование КИХ-фильтра на 4 отвода в Altera-ModelSim. На вход фильтра поступает сигнал –5, 3, 1, 0, 0, 0 и т.д. Правильные значения на выходе фильтра: 10, –1, –40, –10, 26, 6, 0 и т.д.

Если установлен симулятор Altera-ModelSim, то необходимо добавить в модель функциональный блок генератор испытательных стендов (Testbench Cenerator). На рис. 3.44 показано создание тестбенча проекта в автоматическом режиме для последующей симуляции в Altera-ModelSim. На рис. 3.45 показано функциональное моделирование КИХ-фильтра на 4 отвода в Altera-ModelSim.

В табл. 3.2 приведены сравнительные оценки ресурсов ПЛИС при реализации КИХ-фильтра на 4 отвода с использованием пакета расширения Altera DSP Builder (модель FIR_MAC32), а в табл. 3.3 – Xilinx System Generator IDS системы визуально-имитационного моделирования M*atlab*/Simulink.

Таблица 3.2

Общие сведения по числу задействованных ресурсов ПЛИС Altera Cyclone III Device EP3C5F256C6, временная модель Slow (напряжение питания ядра 1200 мВ, температура 85 ^{0}C)

Общее число логических элементов (Total logic elements)	Триггеров логических элементов (Dedicated logic registers)	Аппаратных умножителей с размерностью операндов 9×9 (Embedded Multiplier 9-bit elements)	Рабочая частота в наихудшем случае Fmax, МГц
30	29	1	258

Таблица 3.3

Оценка ресурсов ПЛИС Xilinx серии Spartan-6 xa6slx4-3tqg144

Ресурсы ПЛИС	Функциональные блоки	
	n-tap MAC FIR filter	Dual Port Memory MAC FIR Filter
Рабочая частота, МГц	233	279
Число ЦОС-блоков DSP-48A	1	1
Триггеров	52	57
Секций с LUT	39	38

Продолжим изучение модели FIR_MAC32.mdl. Коэффициенты фильтра рассчитываются с помощью функции (метод рядов Фурье со взвешиванием, обратное преобразование Фурье с использованием оконных функций) (fir1(31,3/8)) .* 2^16-1 с учетом масштабного множителя с частотой среза нормированной к частоте Найквиста Fc=3/8 или 187 кГц/500 кГц (Fn=Fd/2= 500 кГц).

На вход фильтра подается сумма двух синусоидальных сигналов с частотами f1=250 кГц (амплитуда 2^14-1) и f2=100 кГц (амплитуда 2^13-1). Функция fir1 позволяет рассчитать нерекурсивные КИХ-фильтры с линейной ФЧХ.

На рис. 3.46 показана модель КИХ-фильтр на 4 отвода с использованием линии задержки на двухпортовой памяти, ПЗУ для хранения коэффициентов фильтра, MAC-блок настроенный для фильтрации реального сигнала.

Для функционального моделирования в Altera-ModelSim задается синхросигнал с периодом 20 нс и временной шаг симуляции в Matlab/Simulink 0.25 с. Шаг симуляции выбирается исходя из следующих соображений, первое, профильтрованные значения на выходе фильтра появляются через четыре такта синхроимпульса. Второе, на вход КИХ-фильтра подается ЛЧМ-сигнал с частотой дискретизации сигнала $F_S = 1$ кГц ([0:1/fs:1] – вектор дискретных значений времени; 1+[0:1/fs:1].*300 – частота импульса). Шаг модельного времени (Sample time, интервал дискретизации) – 1 с (рис. 3.46).

Коэффициенты фильтра и входной сигнал умножаются на масштабный множитель 2^8-1, профильтрованный сигнал также делится на это значение. Результаты умножения и накопления представляются с 34-разрядной точностью. Блок Bus Conversion осуществляет сдвиг вправо старших 17-разрядов шины (33:0) на позицию (24:8), что равносильно делению на 255:

$$y/65025 = (C_0 * 255)(x_0 * 255) + (C_1 * 255)(x_1 * 255) + \\ + (C_2 * 255)(x_2 * 255) + (C_3 * 255)(x_3 * 255)$$

Входной сигнал, коэффициенты фильтра представляются с 17-разрядной точностью (целые значения со знаком).

В роли управляющего автомата последовательного КИХ-фильтра выступает функциональный блок Increment Decrement настроенный на функцию 2-разрядного суммирующего счетчика. Счетчик через функциональный блок Decoder (дешифратор) управляет работой линией задержки на основе двухпортовой памяти ОЗУ, MAC-блоком и регистром результата, а также используется для адресации к ПЗУ и ОЗУ. В настройках функционального блока Decoder задается 2-разрядная входная шина типа Unsigned Integer и декодируемая величина – десятичная единица.

Ниже показан пример расчета временной функции ЛЧМ-сигнала в системе Matlab.

```
fs=1e3; % частота дискретизации 1кГц
t=0:1/fs:1; % вектор дискретных значений
f0=1+[t*300]; % частота импульса
s1=cos(2.*pi.*t.*f0); получаемый сигнал
plot(s1);
```

Для проверки правильности работы фильтра используется функциональный блок Digital Filter, задается структура Direct form (рис. 3.47). На рис. 3.48, рис. 3.49 и рис. 3.50 показаны настройки функциональных блоков ПЗУ, двухпортовой памяти ОЗУ и блока умножения с накоплением. На рис. 3.51 показано имитационное моделирование фильтрации ЛЧМ-сигнала последовательным КИХ-фильтром на четыре отвода.

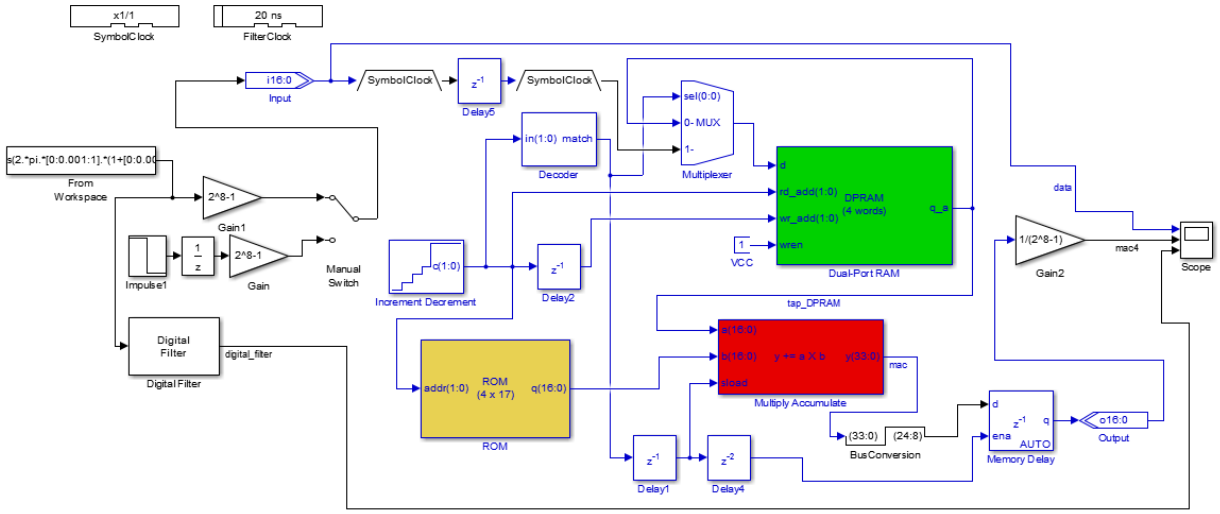
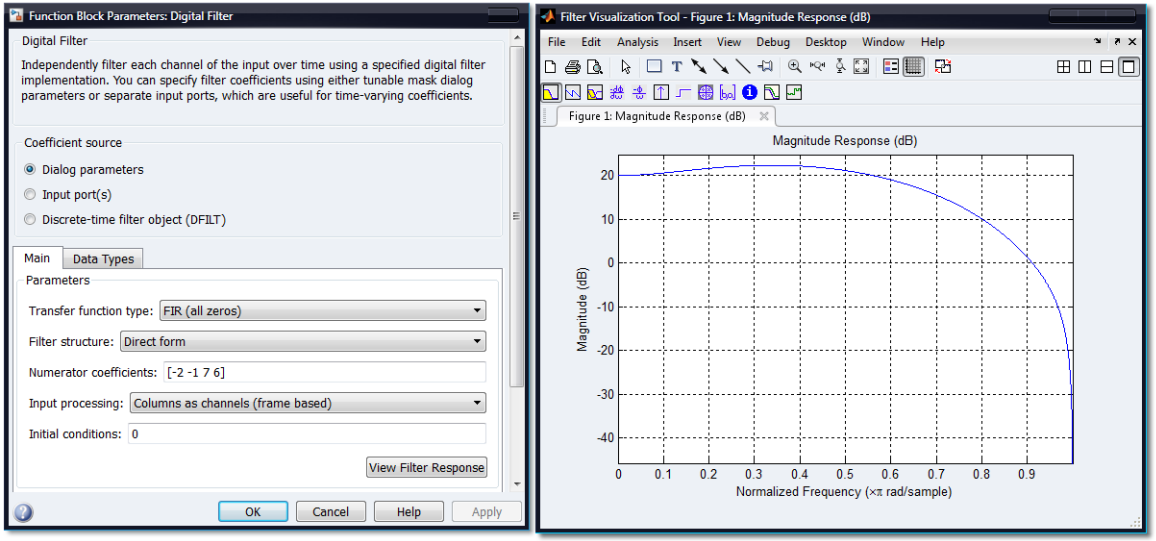


Рис. 3.46. Имитационная модель последовательного КИХ-фильтра на четыре отвода для реализации в базисе ПЛИС Altera на основе одного MAC-блока, подготовленная для фильтрации ЛЧМ-сигнала



а) б)

Рис. 3.47. Расчет цифрового фильтра: а) выбор структуры КИХ-фильтра и задание коэффициентов; б) импульсная характеристика

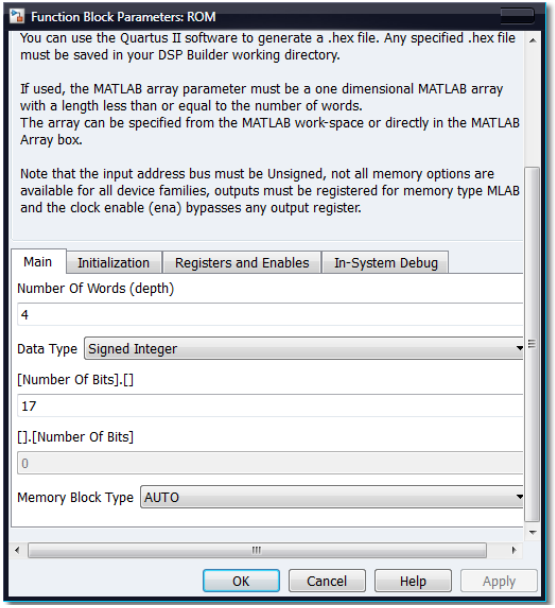


Рис.3.48. Настройки функционального блока ПЗУ (ROM) используемого для хранения коэффициентов фильтра

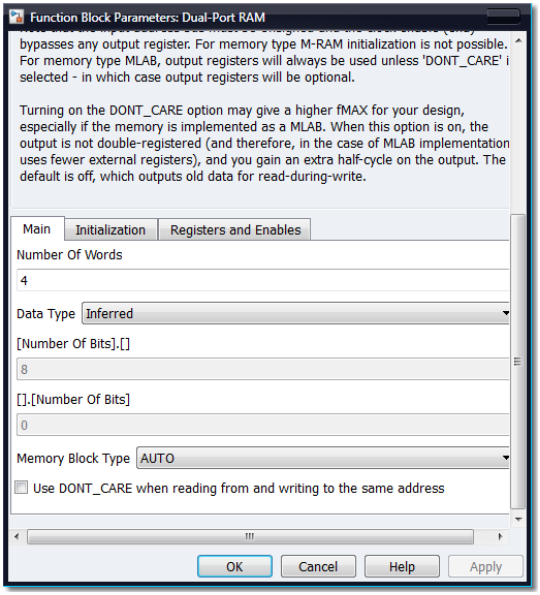


Рис.3.49. Настройки функционального блока двухпортовой памяти ОЗУ (Dual-Port RAM) используемого для организации линии задержки

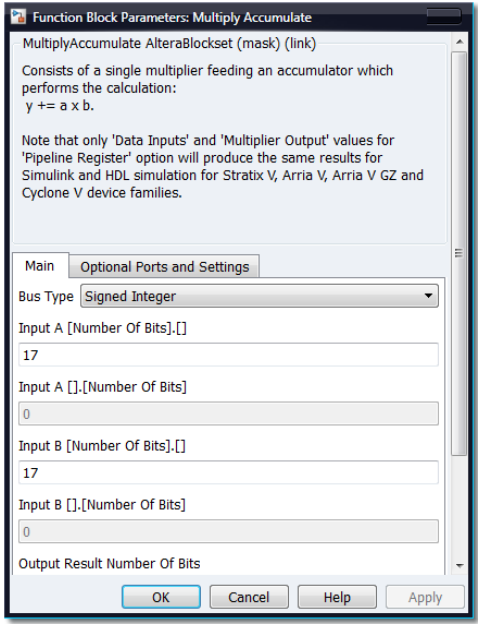


Рис. 3.50. Настройки функционального блока MultiplyAccumulate выполняющего функцию умножения с накоплением (MAC-блок)

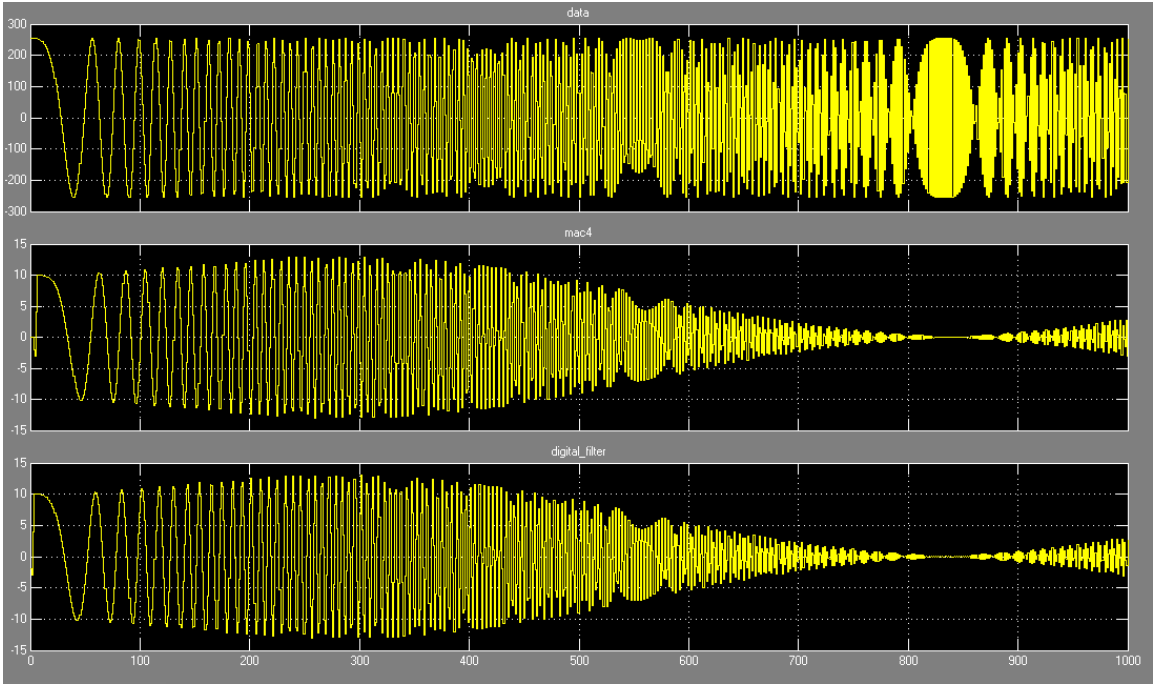


Рис. 3.51. Имитационное моделирование фильтрации ЛЧМ-сигнала последовательным КИХ-фильтром на четыре отвода

Рассмотрим работу КИХ-фильтра на модельном сигнале. На рис. 3.52 показано подключение сигнала [–5 3 1 0 0 0] к входу фильтра. Подадим на вход КИХ-фильтра сигнал [-5 3 1 0 0 0] с частотой 1 МГц, интервал дискретизации 1e-6 с (рис. 3.53), шаг моделирования в Matlab/Simulink 25e-8 с (рис. 3.54), время окончания моделирования 5000*2.5e-8, зададим синхросигнал с периодом 20 нс.

На рис. 3.55 показано имитационное моделирование работы КИХ-фильтра. Правильные значения на выходе фильтра: [10 –1 –40 –10 25 6].

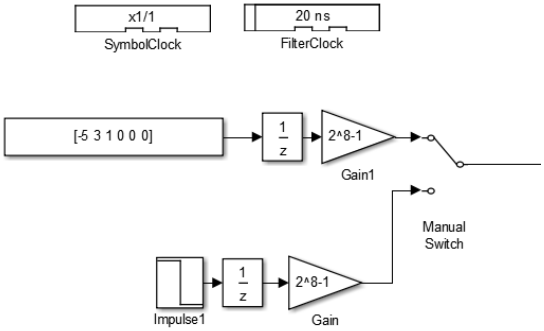


Рис. 3.52. Фрагмент модели КИХ-фильтра

На рис. 3.56 и рис. 3.57 показаны возможные варианты реализации последовательного фильтра. На рис. 3.56 КИХ-фильтр реализован на линии задержки на функциональном блоке Shift Taps (на основе ОЗУ) и мультиплексоре 4 в 1 (задается интервал дискретизации входного сигнала 1e-6 с, шаг моделирования в Matlab/Simulink 25e-8 с, синхросигнал с периодом 20 нс), а на рис. 3.57 используется линия задержки на регистрах из функциональных блоков Delay AlteraBlockset и мультиплексоре 4 в 1 (задается временной шаг симуляции в

Matlab/Simulink 0.25 с, интервал дискретизации входного сигнала 1 с, синхросигнал с периодом 100 нс).

В модели последовательного КИХ-фильтра с линией задержки на регистрах, показанной на рис. 3.57 совместно с управляющим автоматом, используется компаратор (a = = b), на вход b которого подается константа «ноль». Компаратор вырабатывает управляющий сигнал в самом начале счета, а не на следующий временной шаг моделирования как при работе дешифратора, когда задается декодируемая величина 1. Поэтому на выходе компаратора присутствует дополнительная задержка на один такт, позволяющая согласовать работу линии задержки с остальными блоками фильтра.

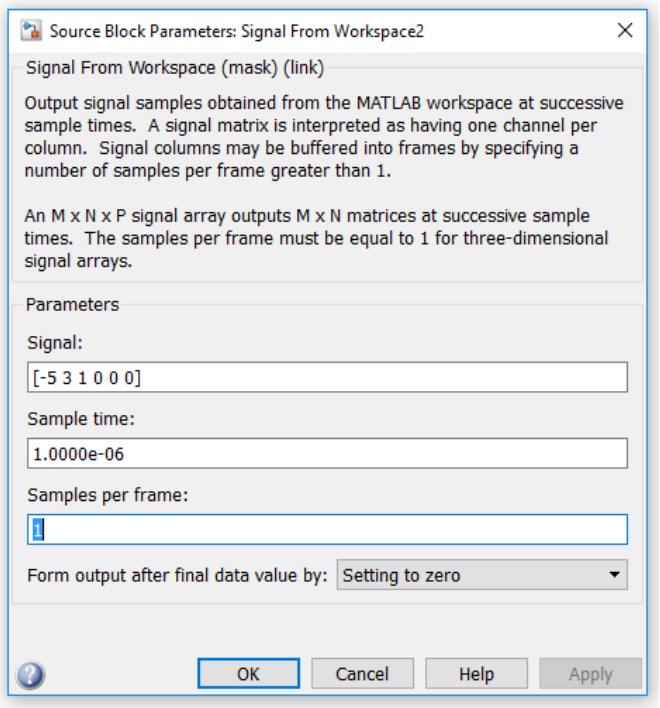


Рис. 3.53. Задание параметров сигнала. Интервал дискретизации 1e-6 с

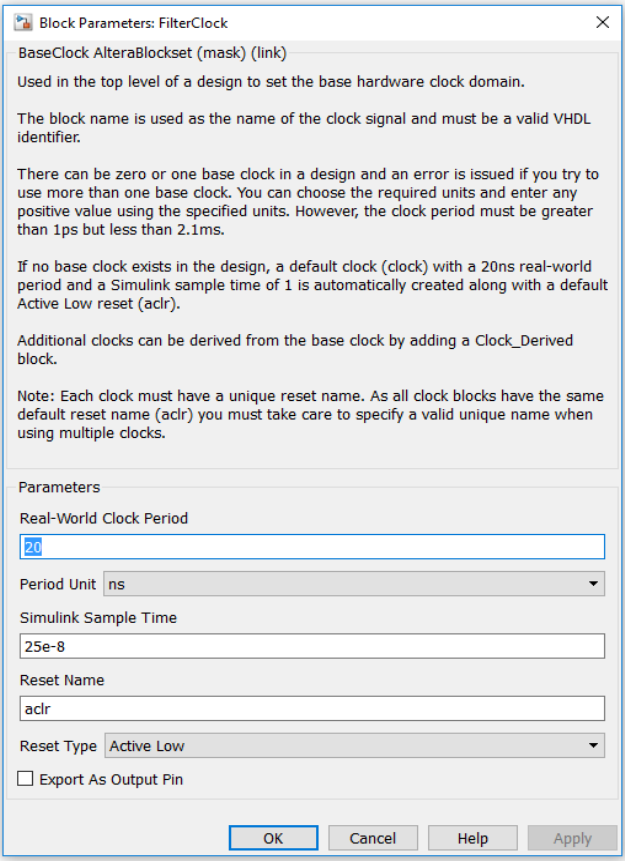


Рис. 3.54. Шаг моделирования в Matlab/Simulink 25e-8 с

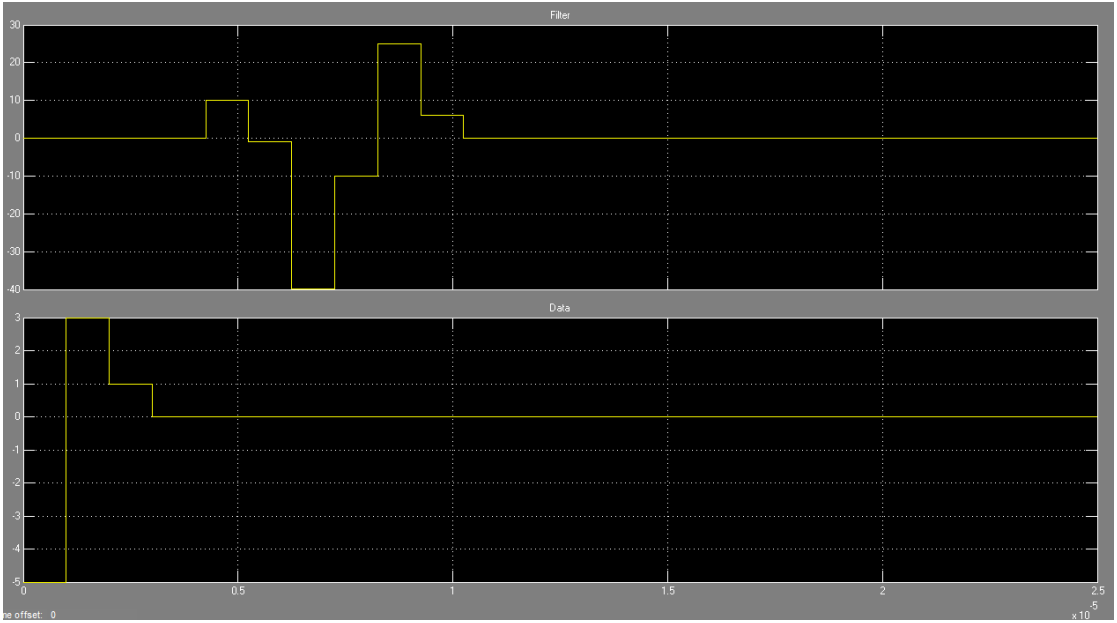


Рис. 3.55. Имитационное моделирование работы последовательного КИХ-фильтра. На вход фильтра поступает сигнал [–5, 3, 1, 0, 0, 0]. Правильные значения на выходе фильтра: [10 –1 –40 –10 25 6]

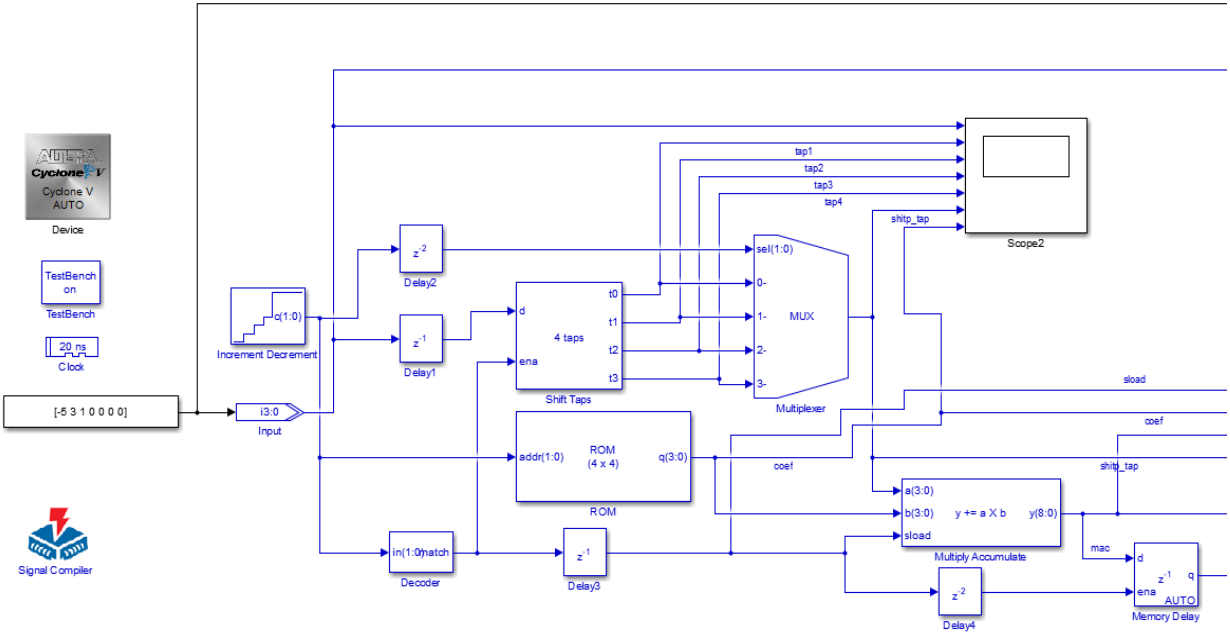


Рис. 3.56. Фрагмент модели последовательного КИХ-фильтра. Линия задержки на функциональном блоке Shift Taps

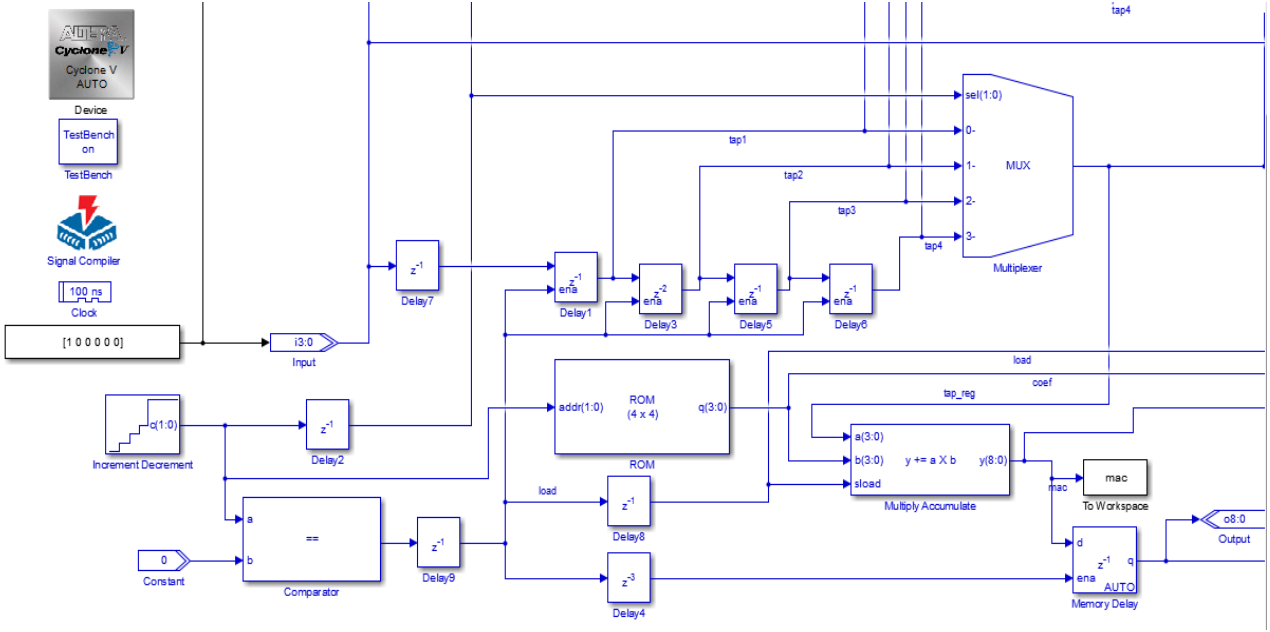


Рис. 3.57. Фрагмент модели последовательного КИХ-фильтра. Линия задержки на регистрах из функциональных блоков Delay с разной латентностью

Для корректной работы линии задержки на регистрах с мультиплексором необходимо латентность второго регистра увеличить до двух. Коммутация отводов линии задержки осуществляется с помощью мультиплексора четыре в один, на адресный вход которого подключается выход управляющего автомата. Для согласования работы линии задержки на основе ОЗУ с мультиплексором (рис. 3.56) необходимо сигнал с выхода управляющего автомата, подключаемый на адресный вход мультиплексора (sel(1:0)), задержать на два такта, а для случая на рис. 3.57 на один такт. Число уровней конвейризации мультиплексоров (Number Of Pipeline Stages) для модели на рис. 3.56 – ноль, а для модели на рис. 3.57 – один.

Рассмотрены различные варианты реализации последовательного КИХ-фильтра на 4 отвода. Профильтрованные значения на выходе последовательного КИХ-фильтра появляются через четыре такта синхроимпульса, для параллельного фильтра результат фильтрации доступен уже после первого такта синхроимпульса. Поэтому для модели последовательного КИХ-фильтра на 4 отвода необходимо обеспечить временной шаг симуляции в Matlab/Simulink 0.25 с, а интервал дискретизации входного сигнала 1 с или например, интервал дискретизации взять 1e-6 с, а шаг моделирования в Matlab/Simulink 25e-8 с, т.е. 25e-8*4=1e-6 с. Можно использовать и такой вариант, в поле Base Clock Multiplicand Numerator блока Clock AlteraBlockset установить значение 4, а шаг моделирования задать 0.0625 с при интервале дискретизации сигнала 1 с. Для параллельного фильтра интервал дискретизации входного сигнала 1 с и временной шаг симуляции в Matlab/Simulink тоже составляет 1 с.

При разработке моделей КИХ-фильтров необходим строгий учет латентности функциональных блоков, входящих в их состав, в том числе обращать внимание на число уровней конвейризации мультиплексоров, а также принимать во внимание то, что выходы блоков ОЗУ и ПЗУ являются регистерными.

4. ПРОЕКТИРОВАНИЕ КИХ-ФИЛЬТРОВ НА РАСПРЕДЕЛЕННОЙ АРИФМЕТИКЕ

4.1. КИХ-фильтры на последовательной распределенной арифметике

Распределенная арифметика широко используется при проектировании высокопроизводительных КИХ- и БИХ-фильтров, адаптивных фильтров, специальных вычислителей, например с применением быстрого преобразования Фурье, дискретного вейвлет-преобразования и др., для реализации мультимедиа-систем в базисе ПЛИС. Поэтому представляет определенный интерес рассмотреть основы такой арифметики на примере проектирования КИХ-фильтра на четыре отвода.

Использование КИХ-фильтров на параллельной распределенной арифметике позволяет для ПЛИС получать рекордное быстродействие за счет использования «безумножительных» схем умножения, не снижаемое при увеличении числа отводов и коэффициентов фильтра, а также точности представления входных данных. Это особенно актуально для проектов использующих бюджетные серии зарубежных ПЛИС и особенно отечественных, в которых отсутствуют аппаратные умножители или их количество ограничено, но обладающих большим количеством логических ресурсов или блочной памятью. А также в том случае, если пользователь откажется от применения в своем проекте при разработке КИХ-фильтров, например, в САПР Quartus II различных мегафункций использующих аппаратные умножители, встроенные в ЦОС-блоки ПЛИС (ALTMULT_ADD, ALTMULT_ACCUM) или мегаядер FIR Compiler или FIR II (Quartus Prime).

Например, ПЛИС серии 5576 не имеют встроенных умножителей, а ПЛИС серии 5578 содержат от 14 до 66 умножителей с размерностью операндов 18×18. На практике

же требуются КИХ-фильтры с крутым спадом АЧХ, что приводит к числу отводов более 110 и выше.

ПЛИС Intel серии Cyclone 10GX типа 10CX220 выпуска 2019 года содержат 192 ЦОС-блока с переменной точностью и количеством встроенных умножителей с размерностью операндов 18×19 – 384 (или 27x27 – 192) с аппаратной поддержкой формата с плавающей запятой. А ПЛИС СнК Intel серии Stratix типа 10DX выпуска 2019 года уже содержат 11520 умножителей с размерностью операндов 18×19.

Использование фильтров на распределенной арифметике не утратило значение и в наши дни. Практически во всех руководствах пользователя по применению зарубежных ПЛИС содержится много примеров по применению распределенной арифметики на практике. Например, инструмент для разработки и отладки высокопроизводительных систем цифровой обработки сигналов в базисе ПЛИС фирмы Xilinx - System Generator IDS содержит функциональный блок FIR Compiler v5.0 являющийся аналогом функции FIR Compiler v5.0 генератора параметризированных ядер XLogiCORE IP. В справочной системе пакета расширения Altera DSP Builder также присутствуют примеры КИХ-фильтров на последовательной распределенной арифметике, например: quartus\dsp_builder\DesignExamples\Demos\Filters\DA32.

В ЦОС-приложениях коэффициенты фильтра могут быть представлены как положительными, так и отрицательными числами (целочисленными значениями со знаком), в свою очередь, информационные сигналы, поступающие на вход фильтра, также могут быть представлены как положительными, так и отрицательными числами. Поэтому важно рассмотреть такие понятия, как дополнение до единицы и дополнение до двух, т.е. обратный и дополнительный коды, а также понятие «расширение знака».

Дополнение до двух наиболее эффективно в операциях умножения и накопления чисел со знаком.

Уравнение КИХ-фильтра представляется как арифметическая сумма произведений:

$$y = \sum_{k=0}^{K-1} C_k \cdot x_k \ , \qquad (4.1)$$

где $y-$ отклик цепи; x_k- k-я входная переменная; C_k-весовой коэффициент k-й входной переменной, который является постоянным для всех n; $K-$ число отводов фильтра.

На рис. 4.1 показана прямая реализация КИХ-фильтра по формуле $y = C_0x_0 + C_1x_1 + C_2x_2 + C_3x_3$ с использованием сдвиговых регистров для организации линии задержки на четыре отвода (такой функциональный узел называют многоразрядный сдвиговый регистр), перемножителей для умножения сигналов на константы и одного сумматора с внутренней организацией «дерево сумматоров». На рис. 4.2 показана общепринятая методика умножения с накоплением (MAC), характерная для реализации в базисе сигнальных процессоров, используемая для построения КИХ-фильтра на четыре отвода из-за отсутствия встроенных умножителей.

На рис. 4.3 показан один из вариантов построения блока умножения с накоплением и алгоритм реализации умножения методом сдвига и сложения с накоплением. Демонстрируется аппаратная реализация умножения числа x (0101, D5) на константу C (1011, D11) с использованием многоразрядного мультиплексора 2 в 1, на один из входов которого подается константа D11, а на другой – ноль, и масштабирующего аккумулятора (сумматора) для суммирования частичных произведений с соответствующими весами. На адресный вход мультиплексора с помощью сдвигового регистра подается число x (D5) младшими разрядами вперед. Результатом умножения является десятичное число 55. Рисунок 4.4 показывает умножение десятичного числа 11 на 5 методом правого сдвига с

накоплением по рекуррентной формуле, показанной на рис. 4.3.

Рассмотрим проектирование КИХ-фильтров в базисе ПЛИС с использованием распределенной арифметики. Преимущество последовательной распределенной арифметики, реализованной в базисе ПЛИС, заключается в снижении объема задействованных ресурсов за счет отказа от использования встроенных умножителей. Структура КИХ-фильтра на четыре отвода будет состоять из одной LUT-таблицы, содержащей комбинацию сумм коэффициентов, являющихся константами всех возможных вариантов на ее адресных входах накапливающего (масштабирующего) сумматора и многоразрядного сдвигового регистра (рис. 4.5).

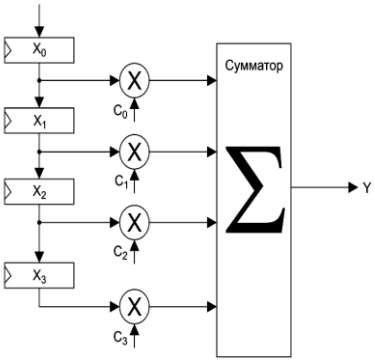


Рис. 4.1. Прямая реализация КИХ-фильтра на четыре отвода

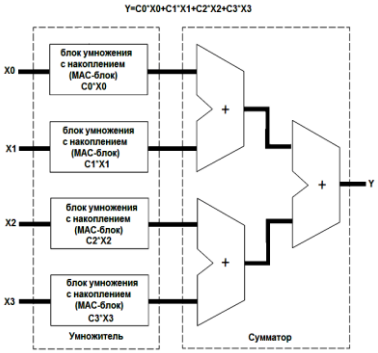


Рис. 4.2. Параллельный алгоритм реализации уравнения КИХ-фильтра на четыре отвода с использованием четырех блоков умножения с накоплением

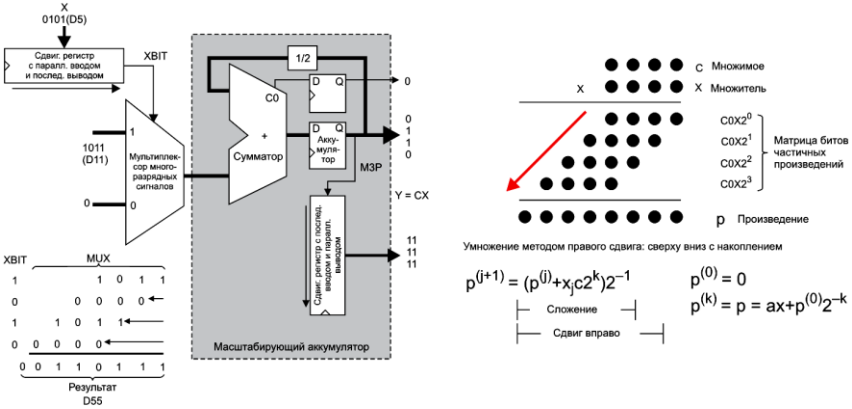


Рис. 4.3. Блок умножения с накоплением и алгоритм реализации умножения методом сдвига и сложения с накоплением

C	1 0 1 1	
X	0 1 0 1	
$p^{(0)}$	0 0 0 0	
$+X_0C$	1 0 1 1	
$2p^{(1)}$	0 1 0 1 1	
$p^{(1)}$	0 1 0 1	1
$+X_1C$	0 0 0 0	
$2p^{(2)}$	0 0 1 0 1	1
$p^{(2)}$	0 0 1 0	1 1
$+X_2C$	1 0 1 1	
$2p^{(3)}$	0 1 1 0 1	1 1
$p^{(3)}$	0 1 1 0	1 1 1
$+X_3C$	0 0 0 0	
$2p^{(4)}$	0 0 1 1 0	1 1 1
$p^{(4)}$	0 0 1 1	0 1 1 1

Рис. 4.4. Умножение десятичного числа 11 на 5 методом правого сдвига с накоплением

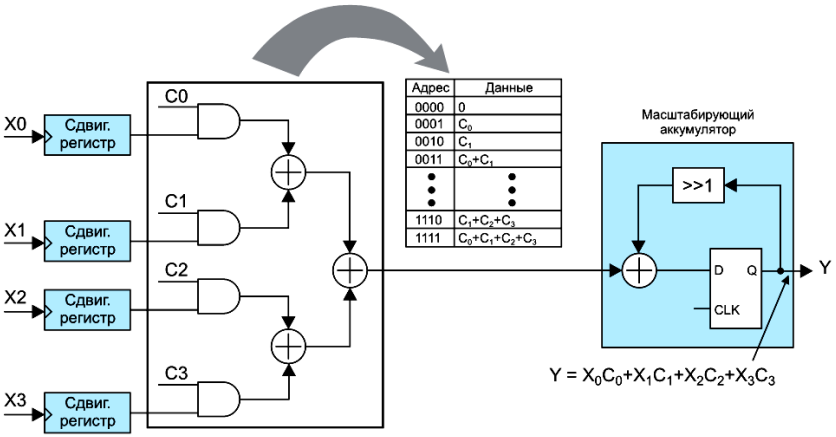


Рис. 4.5. Идея использования распределенной арифметики на примере КИХ-фильтра на четыре отвода

Если рассматривать входные переменные x_k как целые десятичные числа со знаком в дополнительном двоичном коде, то

$$x_k = -x_{k(\mathrm{B}-1)} 2^{B-1} + \sum_{b=0}^{B-2} x_{kb} \cdot 2^b , \qquad (4.2)$$

где B – разрядность кода. Подставим выражение (4.2) в (4.1), получим

$$y = \sum_{k=0}^{K-1} C_k \cdot \left[- x_{k(\mathrm{B}-1)} 2^{B-1} + \sum_{b=0}^{B-2} x_{kb} \cdot 2^b \right] = \\ = -\sum_{k=0}^{K-1} x_{k(B-1)} 2^{B-1} C_k + \sum_{k=0}^{K-1} \sum_{b=0}^{B-2} x_{kb} 2^b C_k . \qquad (4.3)$$

Раскроем все суммы в выражении (4.3) и сгруппируем числа по степеням B:

$$
\begin{aligned}
y = &\left[x_{00}\cdot C_0 + x_{10}\cdot C_1 + x_{20}\cdot C_2 + \ldots + x_{(K\text{-}1)0}\cdot C_{K-1}\right] + \\
&+\left[x_{01}\cdot C_0 + x_{11}\cdot C_1 + x_{21}\cdot C_2 + \ldots + x_{(K-1)1}\cdot C_{K-1}\right]\cdot 2^1 + \\
&+\left[x_{02}\cdot C_0 + x_{12}\cdot C_1 + x_{22}\cdot C_2 + \ldots + x_{(K-1)2}\cdot C_{K-1}\right]\cdot 2^2 + \\
&\ldots. \\
&\ldots. \\
&+\left[x_{0(B-2)}\cdot C_0 + x_{1(B-2)}\cdot C_1 + x_{2(B-2)}\cdot C_2 + \ldots + x_{(K-1)(B-2)}\cdot C_{K-1}\right]\cdot 2^{B-2} - \\
&-\left[x_{0(B-1)}\cdot C_0 + x_{1(B-1)}\cdot C_1 + x_{2(B-1)}\cdot C_2 + \ldots + x_{(K-1)(B-1)}\cdot C_{K-1}\right]\cdot 2^{B-1}.
\end{aligned}
\qquad (4.4)
$$

Выражение (4.4) для КИХ-фильтра на четыре отвода ($K = 4$), в котором входная переменная $x_k(n)$ 4-разрядная ($B = 4$), запишем в виде

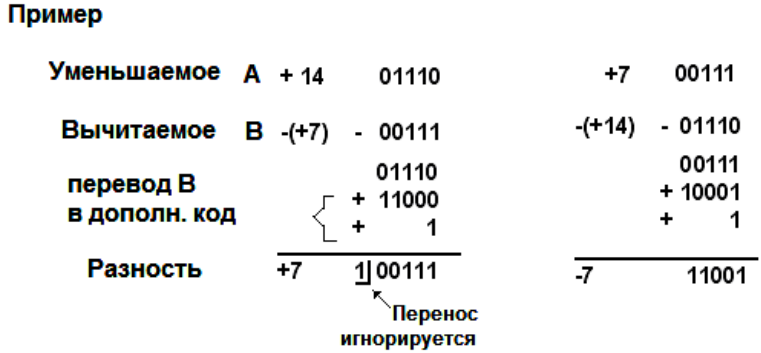
$$
\begin{aligned}
y &= P_0 + P_1\cdot 2^1 + P_2\cdot 2^2 - P_3\cdot 2^3. \\
P_0 &= X_{00}C_0 + X_{10}C_1 + X_{20}C_2 + X_{30}C_3, \\
P_1 &= X_{01}C_0 + X_{11}C_1 + X_{21}C_2 + X_{31}C_3, \\
P_2 &= X_{02}C_0 + X_{12}C_1 + X_{22}C_2 + X_{32}C_3, \\
P_3 &= X_{03}C_0 + X_{13}C_1 + X_{23}C_2 + X_{33}C_3,
\end{aligned}
\qquad (4.5)
$$

где $P_0, P_1, P_2, P_3 -$ частичные произведения.

Вычисление результата $y(n)$ начинается путем адресации всеми битами младшего значащего разряда всех k входных переменных LUT-таблицы, содержащей комбинацию сумм коэффициентов фильтра. Выходное значение просмотровой таблицы сохраняется в масштабирующем аккумуляторе. После этого LUT-таблица адресуется следующими битами от младшего значащего всех входных переменных, результат умножается на 2^1 (путём сдвига слова влево) и добавляется в аккумулятор. Данная операция выполняется над всеми значащими битами, кроме знакового, выходное значение LUT-таблицы, адресуемой старшими битами входных переменных, вычитается из аккумулятора

(рис. 4.6). Одна четырехвходовая LUT-таблица обеспечивает 16 частичных произведений, которые являются комбинациями сумм коэффициентов КИХ-фильтров.

Дополнение до двух можно получить, если прибавить 1 к результату обращения. Обращение логически эквивалентно инверсии каждого бита в числе. Вентили Исключающее ИЛИ можно применить для избирательной инверсии в зависимости от значения управляющего сигнала. Прибавление 1 к результату обращения можно реализовать, задавая 1 на входе переноса c_0 (рис. 4.6).



Пример 1. Вычитание с использованием дополнительного кода (дополнение до двух). Осуществляются инвертирование вычитаемого и суммирование и перенос 1 в младший значащий разряд с последующим сложением

Рассмотрим процесс вычисления более подробно. Предположим, что коэффициенты фильтра целочисленные со знаком известны и равны $C_0 = -2$, $C_1 = -1$, $C_2 = 7$ и $C_3 = 6$. В противном случае необходимо воспользоваться инструментом FDATool (Filter Design & Analysis Tool) системы M*atlab*/Simulink для вычисления значений коэффициентов.

В момент времени $n=0$ на вход КИХ-фильтра подается входная переменная $x_0(n)$ (отсчет, например, десятичное число, равное –5, представленное в дополнительном четырехзначном двоичном коде как 1011), которая сохраняется в регистре PSR_1 (перед вычислением регистры PSR1-PSR4 обнуляются).

Первый цикл обработки состоит в адресации всеми битами младшего значащего разряда всех $K=4$ входных переменных 0001 LUT-таблицы (рис. 4.7). Из LUT-таблицы извлекается коэффициент C_0, представленный в дополнительном коде 111110 с расширением знака на два разряда и поступающий в масштабирующий аккумулятор, где происходит его сложение с нулем. Операция расширения знака для чисел, представленных в дополнительном коде, показана на рис. 4.6.

Полученный результат без учета МЗР сохраняется в регистре Reg 1, а в сдвиговый регистр Shif Reg 2 сохраняется МЗР. Расширение знака для чисел, поступающих на вход масштабируемого аккумулятора перед сложением, и последующее отбрасывание МЗР у полученного результата обеспечивают эквивалент операции масштабирования.

Второй цикл обработки (рис. 4.8) начинается с адресации всеми битами более старшего, младшего, значащего разряда всех k входных переменных LUT-таблицы. Из LUT-таблицы опять извлекается коэффициент C_0, представленный в дополнительном коде 111110 с расширением знака на два разряда, который поступает в масштабирующий аккумулятор, где происходит его сложение с ранее полученным результатом, сохраненным в регистре Reg 1 с расширением знака до 6 разрядов. Полученный МЗР сохраняется в регистре Shif Reg 2, а СЗР игнорируется.

Третий цикл обработки позволяет накопить в регистре Shif Reg 2 число 010 (рис. 4.9). Четвертый цикл обработки

заканчивается вычитанием всех битов знакового разряда всех k входных переменных LUT-таблицы (рис. 4.10).

Извлеченное из LUT-таблицы число переводится в дополнительный код с помощью операции взятия обратного кода (инверсия всех битов) и прибавления 1 к МЗР входа B масштабирующего аккумулятора. В результате таких манипуляций в регистре Shif Reg 2 накапливается число 1010 (десятичное число 10), что соответствует формуле 1: $y(n) = C_0 x_0$. А в регистре Reg 3 будет накоплено двоичное десятиразрядное число 0000001010.

Предположим, что на вход КИХ-фильтра подается, например, десятичное число, равное 3, представленное в дополнительном четырехзначном двоичном коде как 0011, то $y = C_0 x_0 + C_1 x_1 = -2 * 3 + (-1) * (-5) = -6 + 5 = -1$.

Старое значение регистра PSR_1 (–5) сохраняется в регистр PSR_2 и замещается новым числом 3. Получим новые значения адресации LUT-таблицы 0011, 0011, 0000 и 0010. Осуществив четыре цикла обработки, получим в регистре Reg 3 двоичное десятиразрядное число 1111111111 (–1 в дополнительном коде в десятиразрядном представлении).

Электрическая схема КИХ-фильтра на четыре отвода с использованием последовательной распределенной арифметики в САПР ПЛИС Quartus II компании Altera показана на рис. 4.11.

Для хранения комбинации сумм коэффициентов КИХ-фильтра (LUT-таблица) используется мультиплексор 16 в 1. На информационных входах мультиплексора в шестиразрядном представлении с использованием дополнительного кода хранится булева функция: $f = x_0 C_0 + x_1 C_1 + x_2 C_2 + x_3 C_3$.

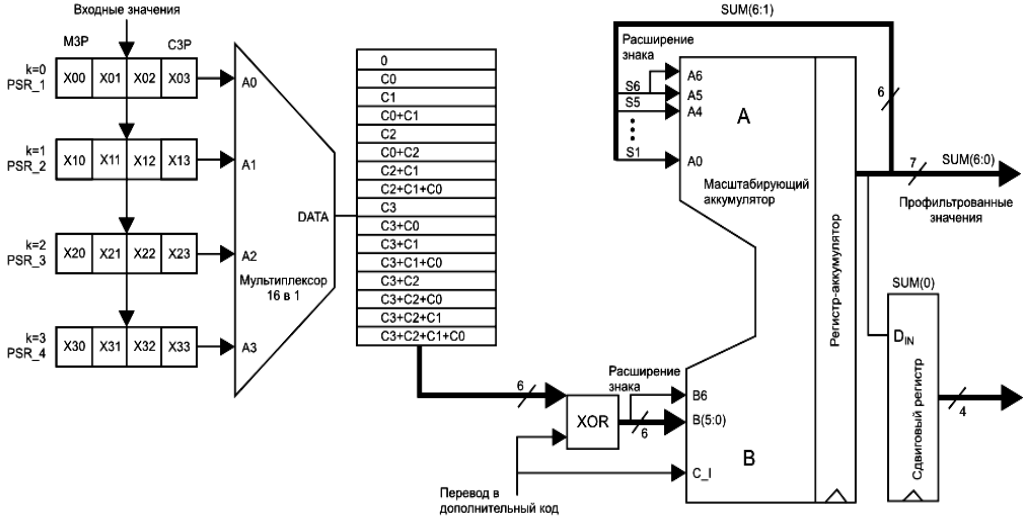


Рис. 4.6. Упрощенная схема КИХ-фильтра на распределенной арифметике

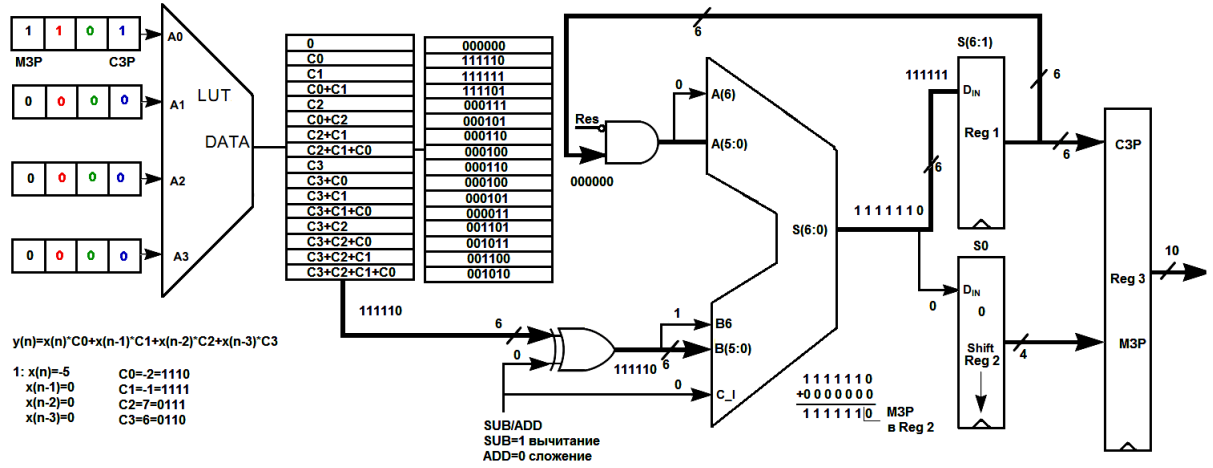


Рис. 4.7. На вход КИХ-фильтра подается десятичное число, равное -5, представленное в дополнительном четырехзначном двоичном коде как 1011, первый цикл обработки (адресация 0001). Суммирование частичного произведения P_0 с весом 2^0 с 0

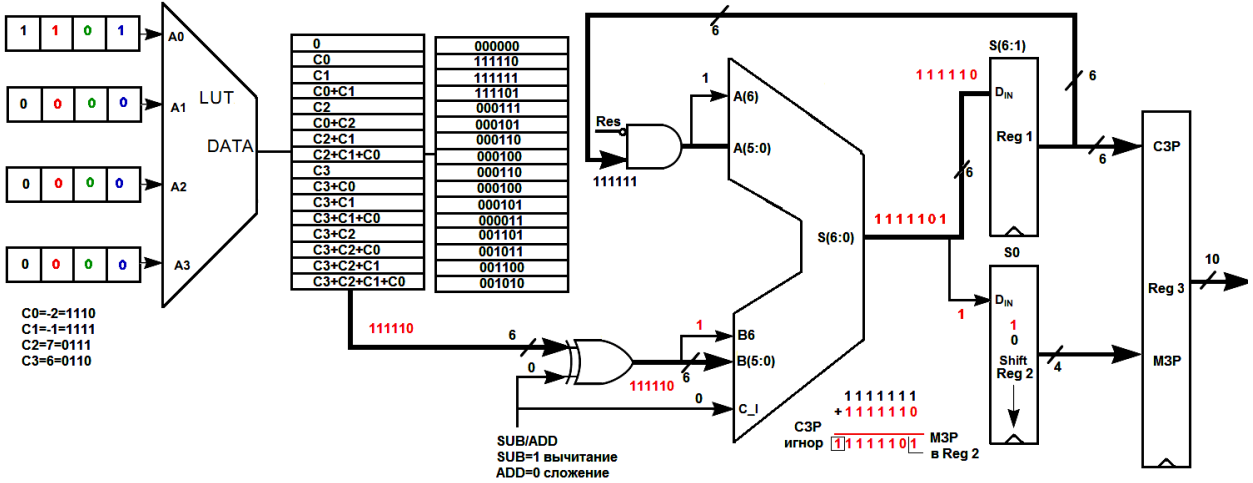


Рис. 4.8. Второй цикл обработки (адресация 0001). Суммирование частичного произведения P_1 с весом 2^1 с частичным произведением P_0 с весом 2^0

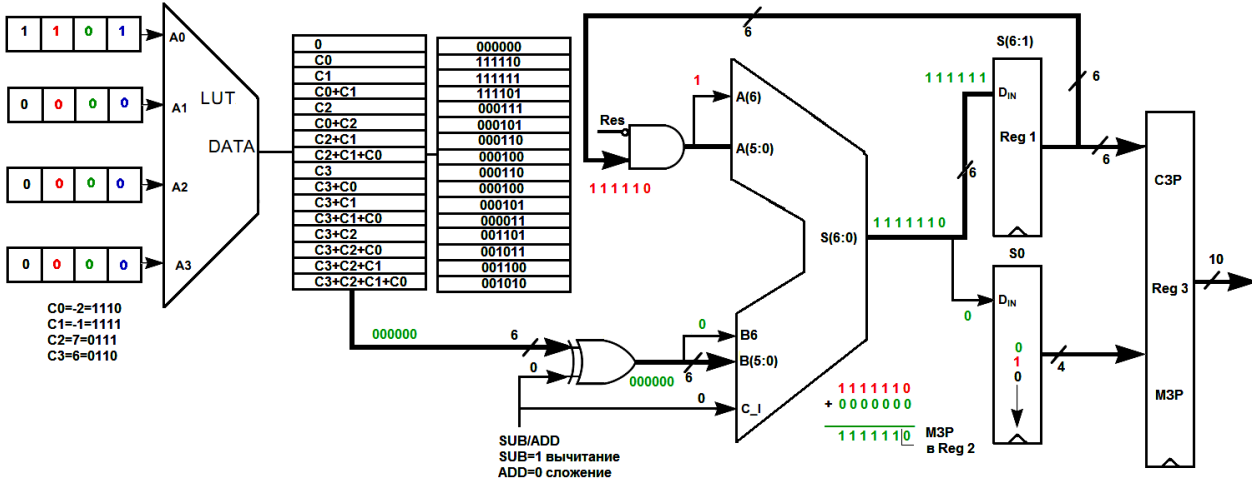


Рис. 4.9. Третий цикл обработки (адресация 0000). Суммирование частичного произведения P_2 с весом 2^2 с суммой частичных произведений P_0 с весом 2^0 и P_1 с весом 2^1

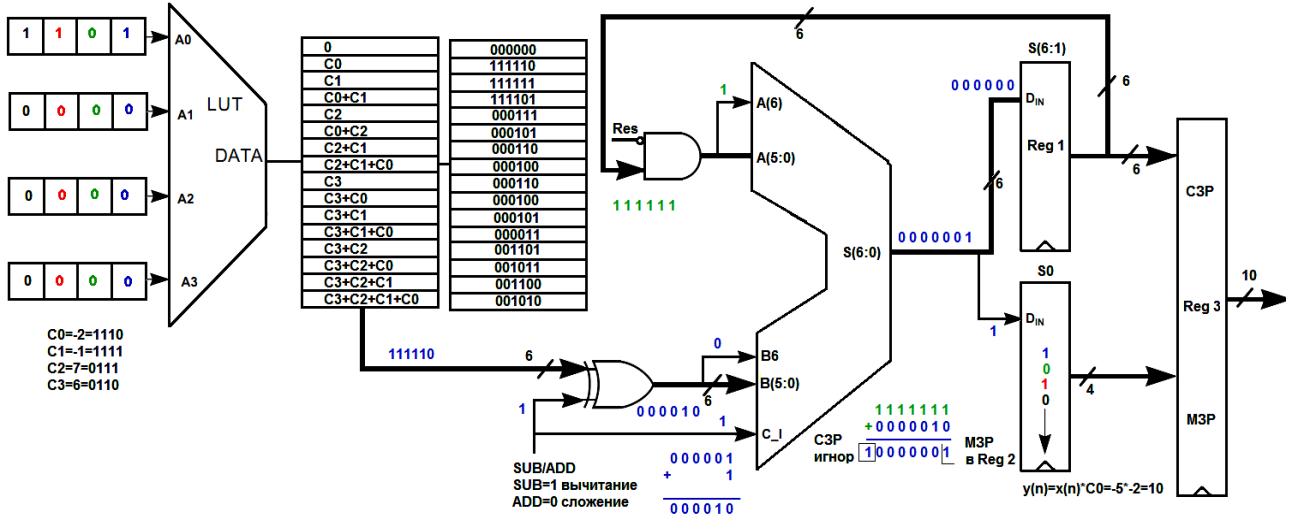


Рис. 4.10. Четвертый цикл обработки (адресация 0001). Суммирование (вычитание) частичного произведения P_3 с весом -2^3, представленного в дополнительном коде с суммой частичных произведений P_2 с весом 2^2, P_1 с весом 2^1 и P_0 с весом 2^0

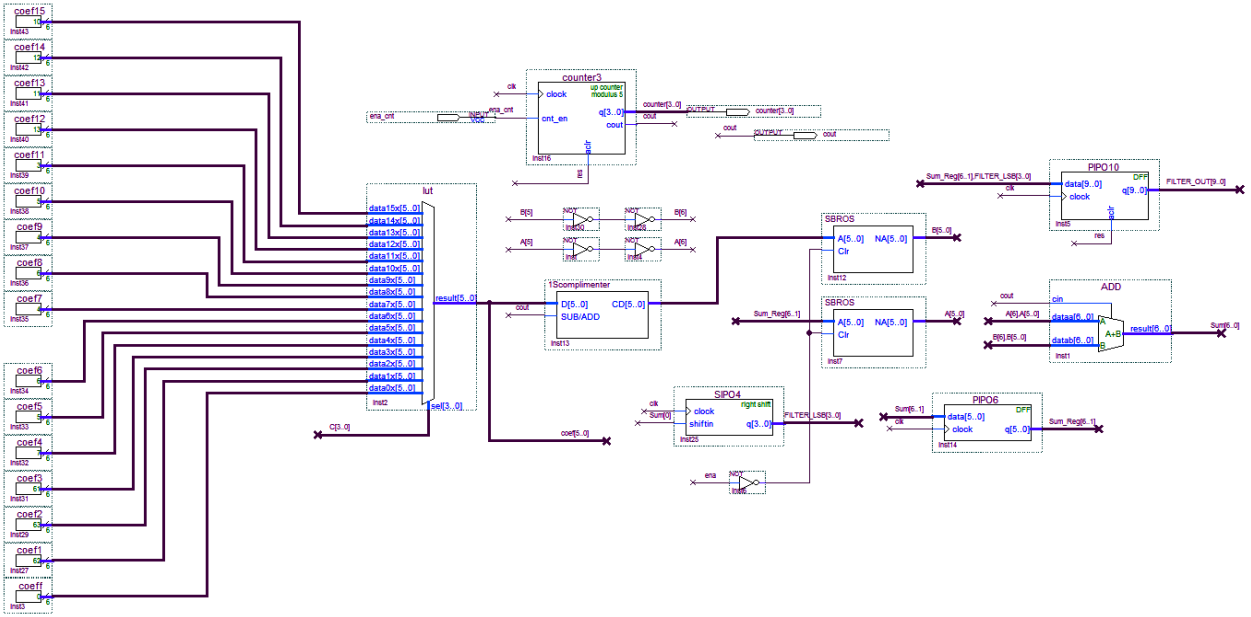


Рис. 4.11. Фрагмент схемы КИХ-фильтра на четыре отвода. Показаны мультиплексор 16 в 1 для хранения комбинации сумм коэффициентов, блок вычисления обратного кода, два блока очистки данных на входах сумматора, счетчик с модулем счета 5 и вспомогательные регистры

На адресные входы мультиплексора поступают биты младшего значащего разряда всех k входных переменных LUT-таблицы. Перед началом работы регистры линии задержки (рис. 4.12) и счетчик обнуляются. Входной отсчет (X0) первоначально сохраняется в 4-разрядном регистре PSR4 со сдвигом вправо с параллельным входом и последовательным выходом (для отладки системы добавляется параллельный выход). При сдвиге вправо в старший значащий разряд регистра PSR4 добавляется 1. За четыре такта синхронизации входной отсчет X0 окажется в сдвиговом регистре SISO4 с последовательным входом и последовательным выходом, за следующие четыре такта – в другом регистре SISO4 и т.д. Каждый регистр SISO4 осуществляет сдвиг вправо. Регистр PSR4 и три регистра SISO4 играют роль линии задержки КИХ-фильтра (многоразрядный сдвиговый регистр).

В качестве простейшего управляющего автомата используется суммирующий счетчик с модулем счета 5. Его задача – отсчитать три значения (частичные произведения), поступающие с выхода мультиплексора, и вычесть четвертое из накопленной суммы. Так как операция вычитания заменяется взятием обратного кода и прибавлением 1 к МЗР, можно использовать обычный 7-разрядный сумматор со входом переноса Cin. В регистре SIPO4 сохраняется МЗР полученной суммы, а в регистре PIPO6 – результат суммирования без учета этого МЗР. Расширение знака на входах сумматора осуществляется с помощью простого копирования СЗР полученной суммы. Регистр PIPO6 и сумматор ADD со схемами расширения знака представляют масштабируемый аккумулятор. Для корректной работы необходимо после обработки каждого входного отсчета сбрасывать входы сумматора в ноль. Это обеспечивают блоки SBROS, представляющие набор элементов 2И. Полученный результат (профильтрованные

входные отсчеты), представляемый в дополнительном коде, сохраняется в регистре PIPO10 с десятибитной точностью.

На рис. 4.13 показаны временные диаграммы работы КИХ-фильтра на распределенной арифметике. На вход КИХ-фильтра подаются входные отсчеты -5 (не показан), 3, 1, 0 в дополнительном коде (представляются как целые числа со знаком). Профильтрованные значения на выходе фильтра (подсвечены оранжевым цветом): 10, -1, –40, 25.

Интересно сравнить временные диаграммы работы КИХ-фильтра на четыре отвода, построенного с помощью мегаядра FIR Compiler САПР ПЛИС Quartus II.

Использование Mega Core Fir Compiler позволяет быстро спроектировать цифровой КИХ-фильтр исходя из заданных параметров. Быстрота и малая трудоемкость расчетов делают данное программное обеспечение незаменимым при проектировании КИХ-фильтров в базисе ПЛИС фирмы Altera.

На рис. 4.14 показаны настройки мегаядра FIR Compiler САПР ПЛИС Quartus II и импульсная характеристика проектируемого фильтра. Целочисленные коэффициенты фильтра загружаются из файла, не масштабируются, имеют представление в формате с фиксированной запятой. Выбирается структура КИХ-фильтра на последовательной распределенной арифметике и ПЛИС серии Stratix III. Галочкой отмечается, что фильтр имеет сильно несимметричную структуру коэффициентов. Для хранения коэффициентов фильтра и отсчетов используются логические ячейки адаптивных логических модулей. Задается также входная и выходная спецификации фильтра (разрядность представления входных и выходных данных). На рис. 4.15 показана тестовая схема КИХ-фильтра с использованием мегаядра FIR Compiler, а на рис. 4.16 – временные диаграммы работы. Входные отсчеты -5, 3, 1, 0. Профильтрованные значения на выходе: 10, -1, -40, 25.

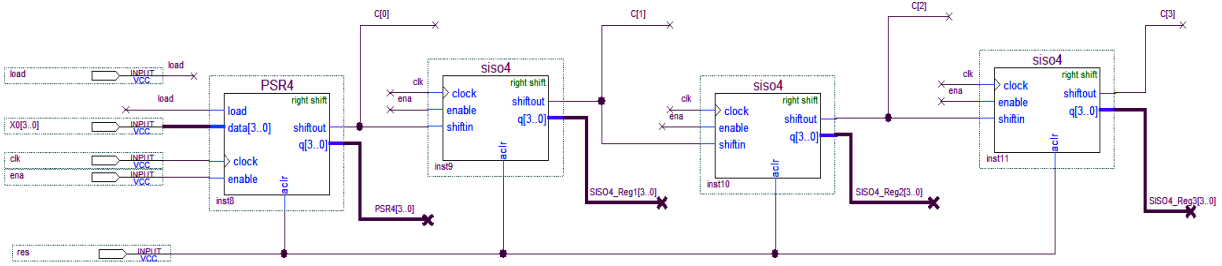


Рис. 4.12. Фрагмент схемы КИХ-фильтра. Линия задержки

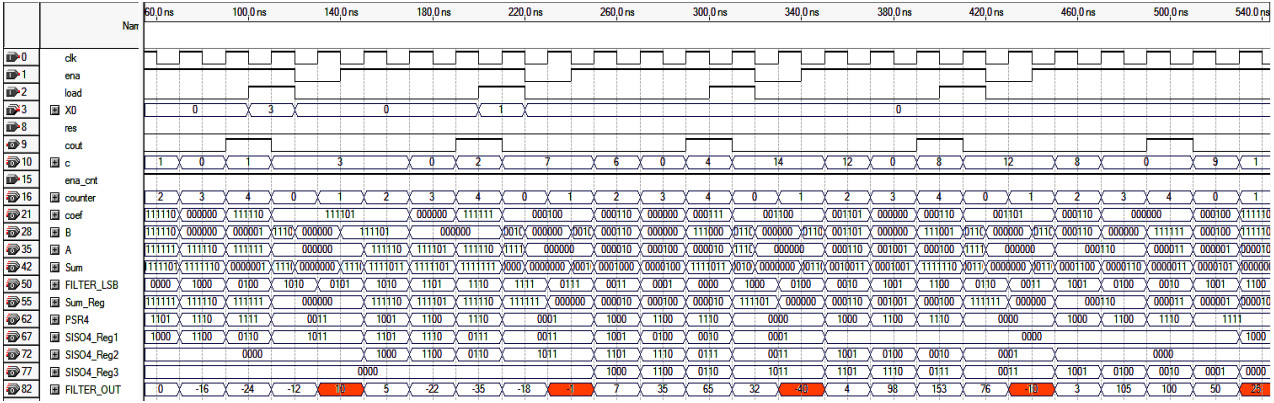


Рис. 4.13. Временные диаграммы работы КИХ-фильтра на распределенной арифметике

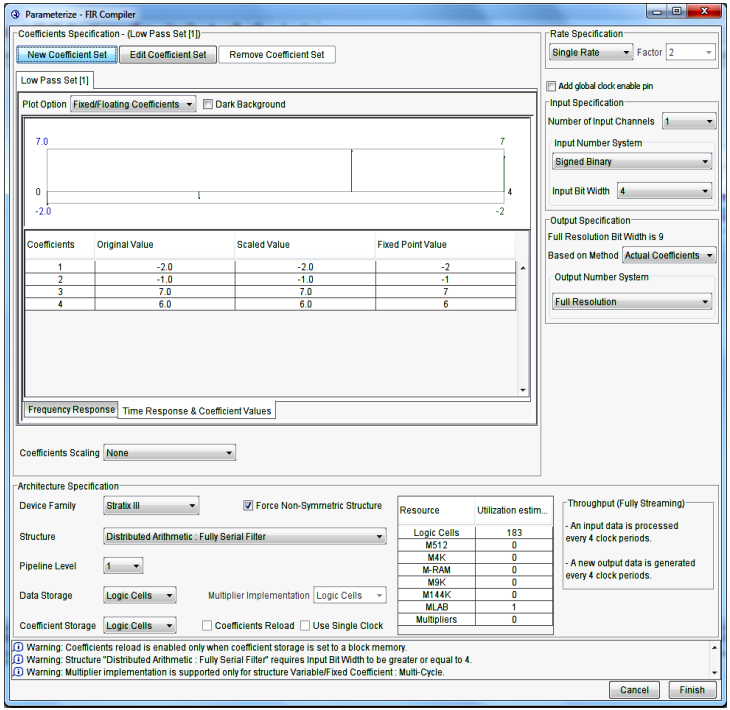


Рис. 4.14. Настройки мегаядра FIR Compiler САПР ПЛИС Quartus II (импульсная характеристика и коэффициенты фильтра)

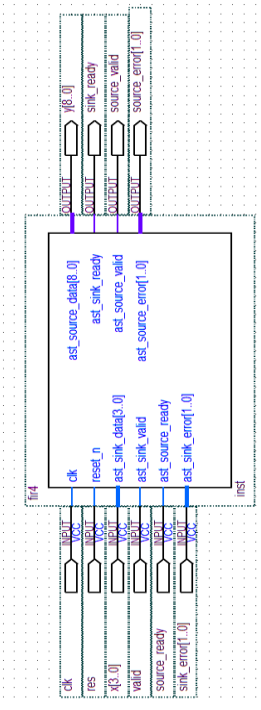


Рис. 4.15. Тестовая схема КИХ-фильтра с использованием мегаядра FIR Compiler

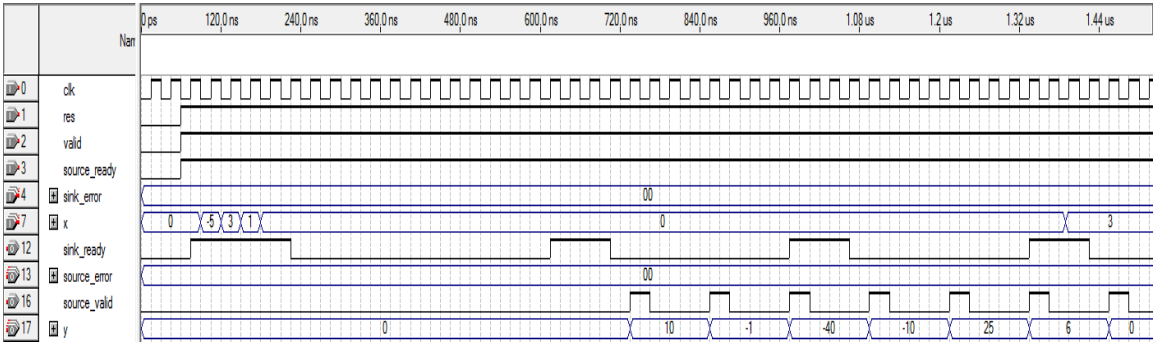


Рис. 4.16. Временные диаграммы работы КИХ-фильтра на мегаядре FIR Compiler

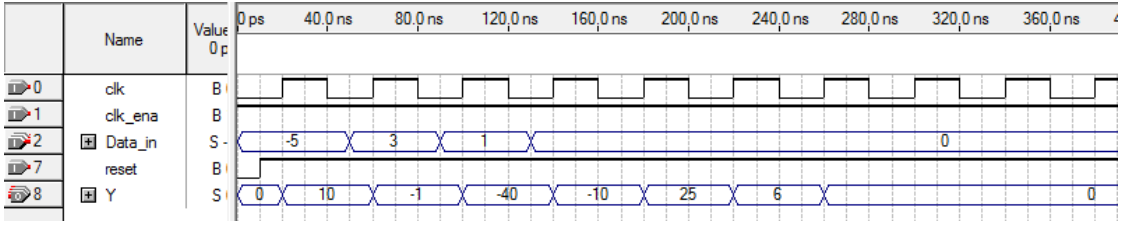


Рис. 4.17. Временные диаграммы работы КИХ-фильтра на четыре отвода по коду языка VHDL (пример 2)

Разработаем код высокоуровневого языка описания аппаратных средств VHDL КИХ-фильтра (пример 1 и 2) с использованием прямой реализации по формуле $y = C_0x_0 + C_1x_1 + C_2x_2 + C_3x_3$ и посмотрим временные диаграммы (рис. 4.17). Сравнивая временные диаграммы (рис. 4.13 и рис. 4.16 с рис. 4.17), видим, что профильтрованные значения на выходе у трех фильтров совпадают, однако у фильтров на распределенной арифметике «нужные» выходные значения обновляются через каждые 4 такта. Существенным отличием является наличие у разных фильтров различных вспомогательных сигналов. Дополнительно мегафункция FIR Compiler имеет встроенный интерфейс, облегчающий взаимодействие с периферийными устройствами.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
package coeffs is
type coef_arr is array (0 to 3) of signed(3 downto 0);
constant coefs: coef_arr:= coef_arr'("1110", "1111", "0111", "0110");
  end coeffs;
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.coeffs.all;
entity fir_var is
 port (clk, reset, clk_ena: in std_logic;
date: in signed (3 downto 0);
q_reg: out signed (9 downto 0));
end fir_var;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
entity filter_4 is
port (din : in std_logic_vector(3 downto 0);
reset, clk : in std_logic;
Sout : out std_logic_vector(7 downto 0));
end filter_4;
ARCHITECTURE a OF filter_4 IS
constant C0: std_logic_vector(3 downto 0) :="1110";
constant C1: std_logic_vector(3 downto 0) :="1111";
constant C2: std_logic_vector(3 downto 0) :="0111";
constant C3: std_logic_vector(3 downto 0) :="0110";
signal x0,x1,x2,x3:std_logic_vector(3 downto 0);
signal m0,m1,m2,m3:std_logic_vector(7 downto 0);
```

```
architecture beh of fir_var is
begin
process(clk,reset)
type shift_arr is array (3 downto 0)
     of signed (3 downto 0);
variable shift: shift_arr;
variable tmp: signed (3 downto 0);
variable pro: signed (7 downto 0);
variable acc: signed (9 downto 0);
begin
if reset='0' then
for i in 0 to 3 loop
shift(i):= (others => '0');
end loop;
q_reg<= (others => '0');
elsif(clk'event and clk = '1') then
if clk_ena='1' then
shift(0):=date;
pro := shift(0) * coefs(0);
acc := conv_signed(pro, 10);
for i in 2 downto 0 loop
pro := shift(i+1) * coefs(i+1);
acc := acc + conv_signed(pro, 10);
shift(i+1):= shift(i);
end loop;
end if; end if;
q_reg<=acc;
end process; end beh;
```

Пример 1. Код языка VHDL КИХ-фильтра на четыре отвода

```
BEGIN
m0<=(signed(x0)*signed(C0));
m1<=(signed(x1)*signed(C1));
m2<=(signed(x2)*signed(C2));
m3<=(signed(x3)*signed(C3));

Sout<=(signed(m0)+signed(m1)+
signed(m2)+signed(m3));
process(clk,reset)
begin
if reset='1' then
x0<=(others=>'0');
x1<=(others=>'0');
x2<=(others=>'0');
x3<=(others=>'0');
elsif (clk'event and clk='1') then
x0(3 downto 0) <=din(3 downto 0);
x1(3 downto 0) <=x0(3 downto 0);
x2(3 downto 0) <=x1(3 downto 0);
x3(3 downto 0) <=x2(3 downto 0);
end if;
end process;
END a;
```

Пример 2. Код языка VHDL КИХ-фильтра на четыре отвода (вариант)

В данном разделе на примере проектирования простейшего КИХ-фильтра на четыре отвода показаны основы распределенной арифметики, широко используемой для разработки высокопроизводительных цифровых устройств цифровой обработки сигналов.

4.2. Проектирование КИХ-фильтра на распределенной арифметике в системе визуально-имитационного моделирования Matlab/Simulink с использованием Altera DSP Builder

Имитационная модель КИХ-фильтра, рассматриваемая в этом разделе, была доработана на основе примера справочной системы пакета расширения Altera DSP Builder по адресу: quartus\dsp_builder\DesignExamples\Demos\Filters\DA32.

На рис. 4.18 показана имитационная модель КИХ-фильтра на последовательной распределенной арифметике для реализации в базисе ПЛИС Altera. Часть назначений функциональных блоков (входы и выходы из ПЛИС, элемент задержки, линия задержки на основе двухпортовой памяти с дистанцией 4) фильтра рассмотрены в разделе 3.2. Сосредоточимся на основных блоках.

Блок BaseClock. Для функционального моделирования в Altera-ModelSim задается синхросигнал с периодом 20 нс (поле Real-World Clock Period) и временной шаг симуляции в Matlab/Simulink 1 с (поле Simulink Sample Time) в блоке BaseClock. Интервал дискретизации входного сигнала (Sample time) в функциональном блоке Signal From Workspace – 4 с (соответствует числу битов в блоке Pattern).

Блок Pattern (формирует периодическую повторяющуюся последовательность битов 1000) выполняет функцию простейшего управляющего цифрового автомата КИХ-фильтра. С одной стороны он управляет загрузкой чисел в регистр (функциональный блок, преобразующий параллельный код в последовательный с обозначением Parallel To Serial, выполняет функцию как у регистров PSR1-PSR4 на рис. 4.7), а с другой – формирует вес -2^3 частичного произведения P_3 с учетом латентности перед последним суммированием и результат фильтрации.

На рис. 4.19 показано формирование содержимого 4-входовой LUT по формуле:

$$q = C_0 a_0 + C_1 a_1 + C_2 a_2 + C_3 a_3 = -2a_0 - 1a_1 + 7a_2 + 6a_3 .$$

Комбинации сумм коэффициентов КИХ-фильтра представляются с 6-разрядной точностью, а сами коэффициенты с 4-разрядной.

На рис. 4.20 показан масштабирующий аккумулятор КИХ-фильтра. Функциональный блок XOR (Исключающее ИЛИ) совместно с блоком Adder формируют дополнение до двух путем прибавления 1 к результату обращения. Обращение логически эквивалентно инверсии каждого бита в числе. Вентиль Исключающее ИЛИ применяется для избирательной инверсии в зависимости от значения управляющего сигнала формируемого блоком Pattern (сигнал clr масштабирующего аккумулятора задержанный на пять тактов синхрочастоты). Прибавление 1 к результату обращения реализуется путем формирования единицы на входе переноса c_{in} 7-разрядного сумматора. МЗР результата суммирования, полученный с помощью функционального блока Extract Bit, поступает в блок, выполняющий преобразование последовательной шины (одноразрядный сигнал) в параллельную с обозначением Serial to Parallel, что соответствует сдвиговому регистру Shif Reg 2 на рис. 4.7.

В настройках функционального блока с обозначением Serial to Parallel задается формат представления чисел после преобразования из последовательного кода в параллельный – целые десятичные числа со знаком и то, что первый входящий бит будет представлять МЗР в формируемом 4-разрядном слове (рис. 4.21).

Сам же 7-разрядный результат суммирования задерживается на такт и в функциональном блоке с обозначением Shift by Two осуществляется отбрасывание МЗР, что равносильно сдвигу числа влево по разрядной сетке (умножение на два), которое и поступает по обратной связи на

один из входов сумматора (вход a) через мультиплексор управляемым сигналом clr. При этом объединяют старшие разряды a(5:0) и младшие b(3:0) в 10-разрядный сигнал с помощью блока выполняющего функцию конкатенации и сохраняют его в регистре результата.

Операция задержка и сдвиг на рис. 4.20 соответствуют функционалу регистра Reg 1, а сохранение МЗР соответствует функционалу Reg 2, объединение результатов соответствует функционалу регистра результата Reg 3 на рис. 4.7. Для сброса входа a сумматора в ноль после завершения четырех циклов обработки на второй вход мультиплексора в обратной связи подключен ноль. Если не предусмотреть сброс одного из входов сумматора в ноль, то он продолжит накапливать сумму после четвертого цикла.

На рис. 4.22 показано имитационное моделирование работы КИХ-фильтра на последовательной распределенной арифметике. На вход фильтра поступает сигнал [–5 3 0 0 0]. Правильные значения на выходе фильтра: [10 –1 –40 –10 25 6]. На рис. 4.23а-е показаны информационные потоки, а на рис. 4.23ж приведено сравнение информационных потоков, протекающих в имитационной модели КИХ-фильтра и в электрической схеме, показанной на рис. 4.24.

Отметим, что самостоятельная разработка проекта КИХ-фильтра в САПР Quartus II является не из лёгких задач. Для этих целей, чтобы облегчить разработчику жизнь существует специальная мегафункция FIR Compiler, о которой речь пойдет чуть ниже. Данная мегафункция позволяет разрабатывать КИХ-фильтры как на последовательной, так и на параллельной распределенной арифметиках.

Одна из возможных реализаций КИХ-фильтра на 4 отвода с использованием последовательной распределенной арифметике в САПР ПЛИС Quartus II компании Altera пусть и не самой совершенной показана на рис. 4.24.

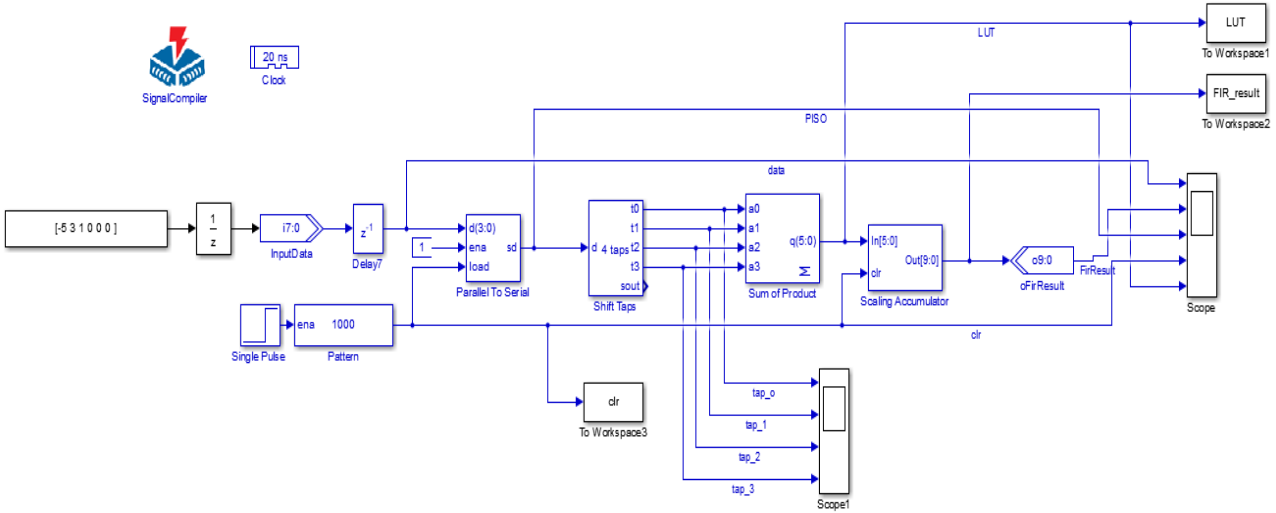


Рис. 4.18. Имитационная модель КИХ-фильтра на последовательной распределенной арифметике для реализации в базисе ПЛИС Altera

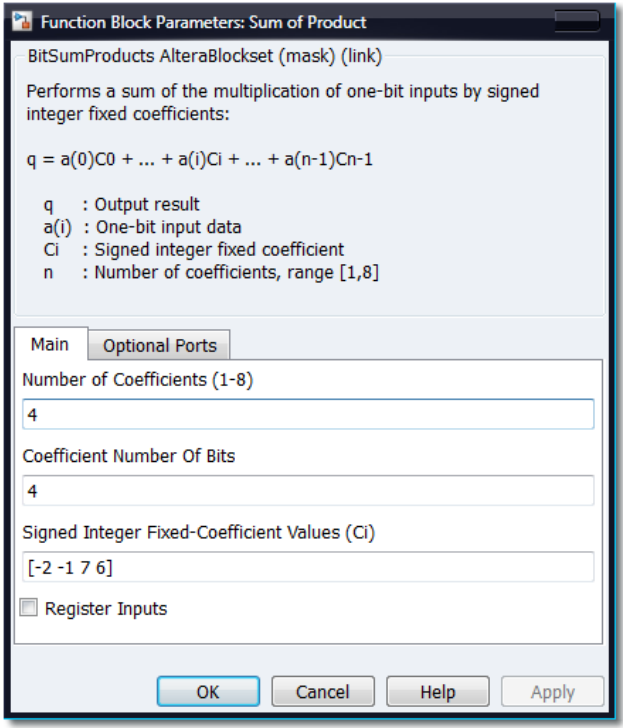


Рис. 4.19. Формирование содержимого LUT

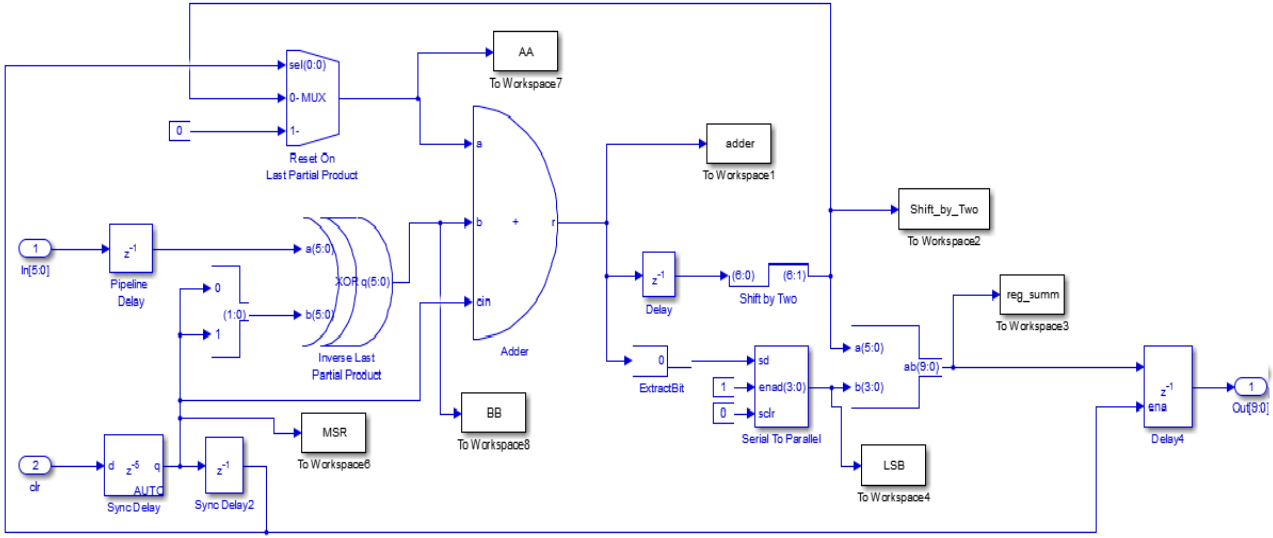


Рис. 4.20. Масштабирующий аккумулятор КИХ-фильтра

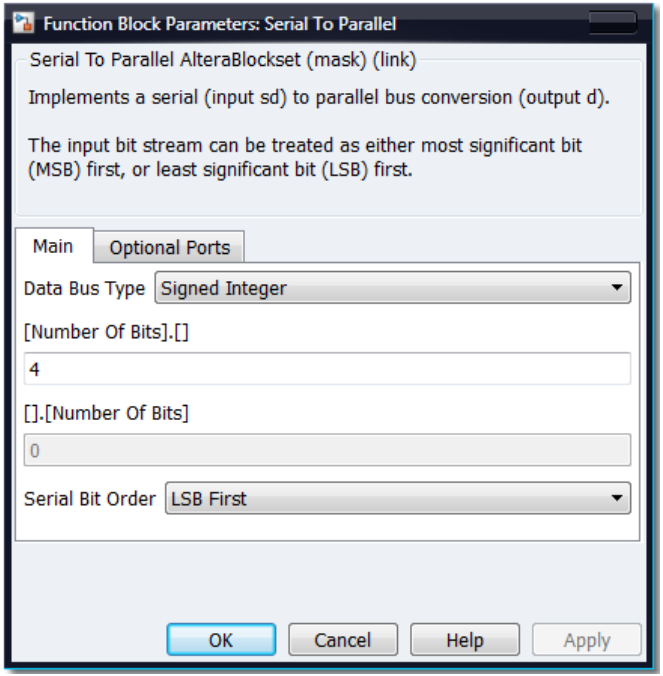


Рис. 4.21. Функциональный блок, выполняющий преобразование последовательного кода (одноразрядный сигнал) в параллельный

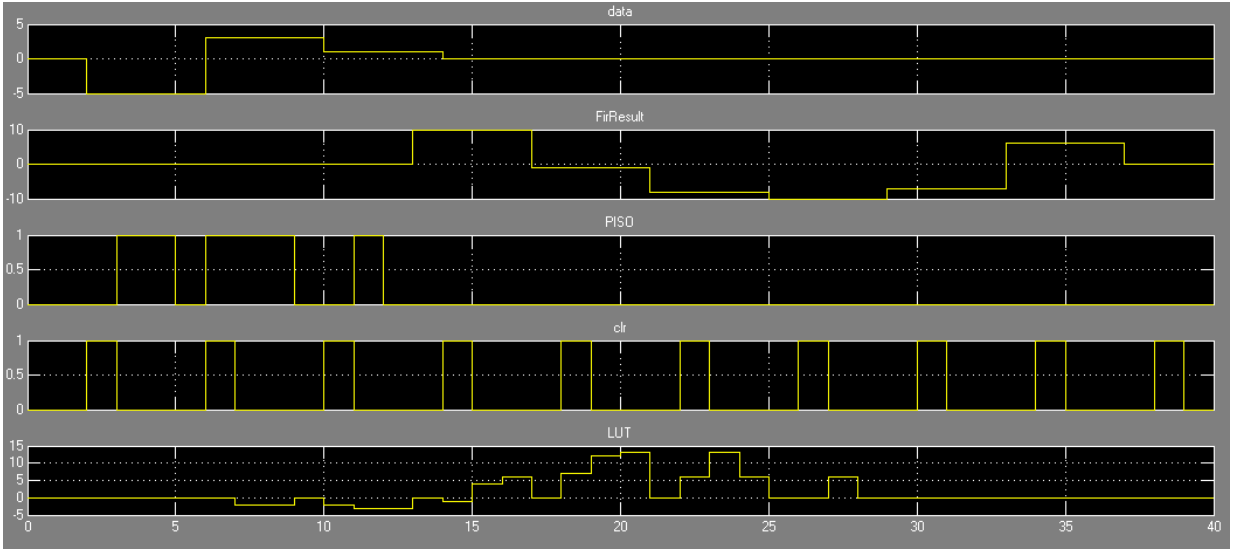


Рис. 4.22. Имитационное моделирование работы КИХ-фильтра на последовательной распределенной арифметике

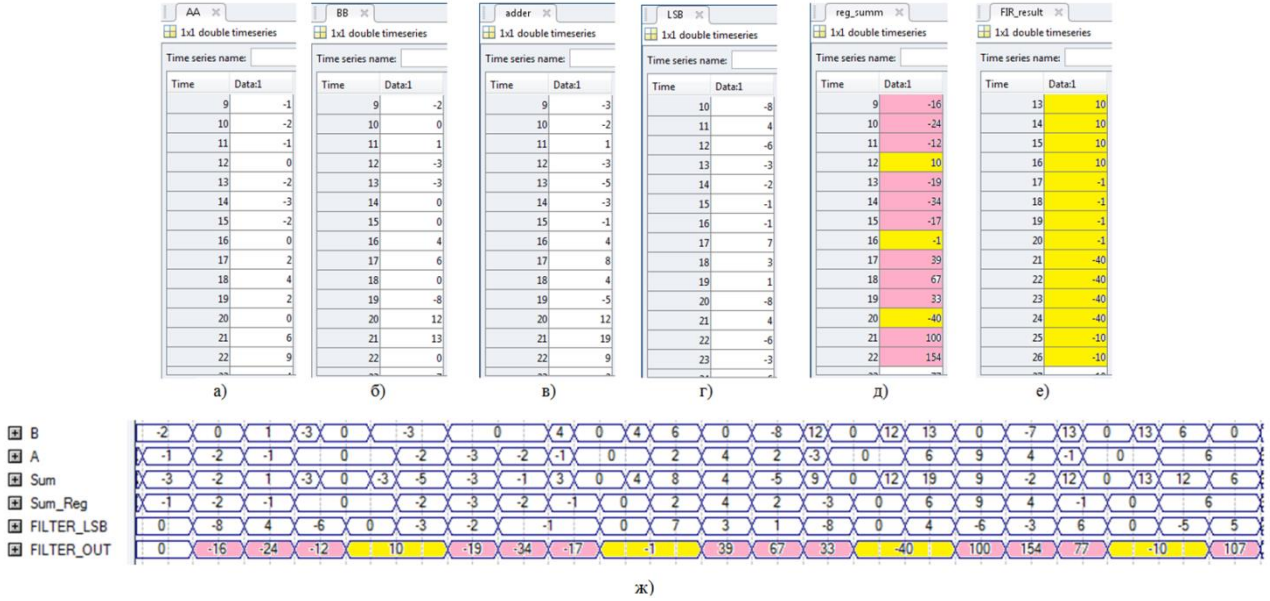


Рис. 4.23. Сравнение информационных потоков, протекающих в имитационной модели КИХ-фильтра и в электрической схеме проекта на рис. 4.24

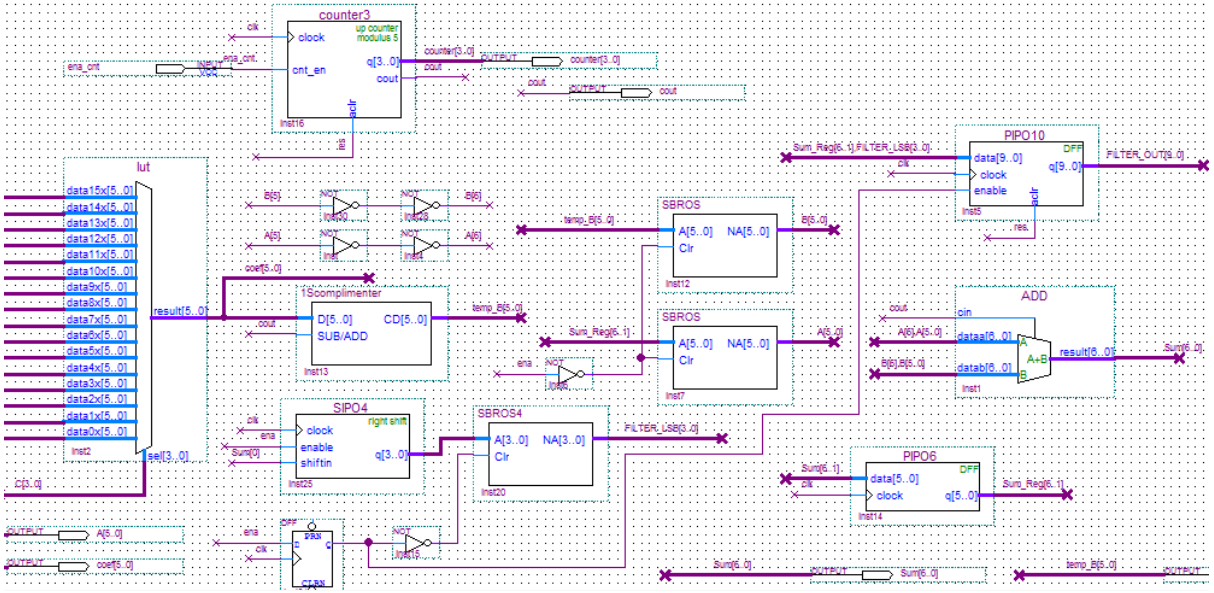


Рис. 4.24. Фрагмент схемы КИХ-фильтра на 4 отвода. Показаны мультиплексор 16 в 1 для хранения комбинации сумм коэффициентов, блок вычисления обратного кода, два блока очистки данных на входах сумматора, счетчик с модулем счета 5 и вспомогательные регистры

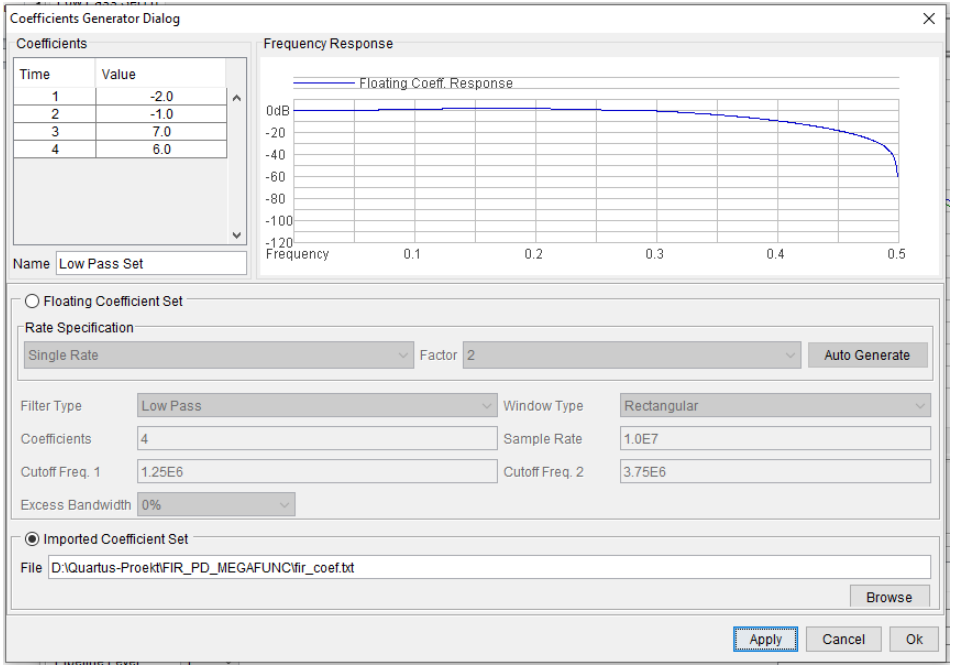


Рис. 4.25. Настройки мегафункции FIR Compiler САПР ПЛИС Quartus II (АЧХ КИХ-фильтра, коэфициенты загружаются из файла)

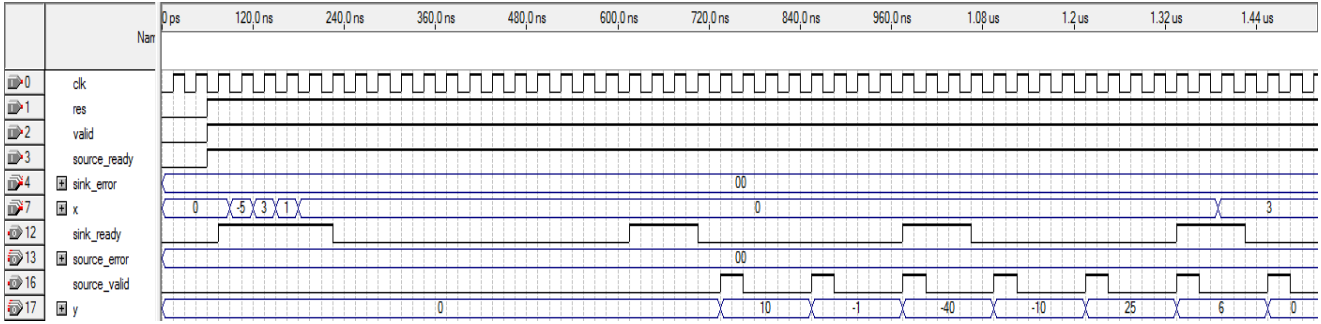


Рис. 4.26. Временные диаграммы работы КИХ-фильтра на мегафункции FIR Compiler

При разработке проекта, показанного на рис. 4.7 и рис. 4.11, не использовались знания об имитационной модели, представленной на рис. 4.18. Промежуточные результаты представления профильтрованных значений формируются в регистре результата (Reg 3, рис. 4.7) по другому механизму. Итоговые результаты фильтрации сигнала показаны оранжевым цветом на рис. 4.13.

Доработаем проект (рис. 4.11) так, чтобы информационные потоки совпадали с потоками имитационной модели на рис. 4.18. Переработанный проект показан на рис. 4.24. Для хранения комбинации сумм коэффициентов КИХ-фильтра используется мультиплексор 16 в 1 (4-входовая LUT). На информационных входах мультиплексора в шестиразрядном представлении с использованием дополнительного кода хранится булева функция $f = x_0C_0 + x_1C_1 + x_2C_2 + x_3C_3$. На адресные входы мультиплексора поступают биты младшего значащего разряда всех k входных переменных LUT.

В качестве простейшего управляющего автомата используется суммирующий счетчик с модулем счета 5. Его задача отсчитать три значения (частичные произведения), поступающих с выхода мультиплексора, и вычесть четвертое из накопленной суммы. Так как операция вычитания заменяется взятием обратного кода и прибавлением 1 к МЗР, то можно использовать обычный асинхронный 7-разрядный сумматор со входом переноса Cin.

В регистре SIPO4 сохраняется МЗР полученной суммы, а в регистре PIPO6 результат суммирования без учета этого МЗР. Расширение знака на входах сумматора осуществляется с помощью простого копирования СЗР полученной суммы. Регистр PIPO6 и сумматор ADD со схемами расширения знака представляют масштабирующий аккумулятор.

Недостатком предложенной схемы проекта КИХ-фильтра является то, что для корректной работы необходимо

после обработки каждого входного отсчета сбрасывать все входы сумматора в ноль. Это обеспечивают блоки SBROS представляющие набор элементов 2И. А для правильности формирования профильтрованных значений потребовалось еще добавить (в отличие от схемы на рис. 4.11) в регистр SIPO4 синхронный сигнал разрешения работы enable, а на его выходах организовать сброс от этого же сигнала с задержкой на такт. Полученный результат (профильтрованные входные отсчеты) представляемый в дополнительном коде сохраняется в регистре PIPO10 с 10-разрядной точностью. Сравнение информационных потоков, протекающих в имитационной модели КИХ-фильтра и в проекте на рис. 4.23, показало их совпадение.

На рис. 4.25 показаны настройки мегафункции FIR Compiler для генерации последовательного КИХ-фильтра на полностью последовательной распределенной арифметике (Fully Serial Filter) для реализации в базисе ПЛИС Stratix III. Результаты вычислений и коэффициенты фильтра (не масштабируются) хранятся в логических ячейках ПЛИС. Точность представления входных отсчетов 4 бита со знаком, а результат фильтрации представляется с 9-разрядной точностью. На рис. 4.26 представлены временные диаграммы работы КИХ-фильтра на мегафункции FIR Compiler. На вход фильтра поступает сигнал –5, 3, 1, 0, 0, 0 и т.д. на выходе имеем профильтрованные значения: 10, –1, –40, –10, 25, 6, 0 и т.д. Профильтрованные значения появляются на выходе фильтра по стробу source valid и обновляются через каждые 5 тактов синхрочастоты. Промежуточные результаты фильтрации мегафункция не отображает.

Посмотрим, как разработанная модель КИХ-фильтра на последовательной распределенной арифметике справляется с фильтрацией ЛЧМ-сигнала с частотой дискретизации сигнала $F_S = 1$ кГц ([0:1/fs:1] – вектор дискретных значений времени, с частотой импульса 1+[0:1/fs:1].*300). Зададим интервал

дискретизации ЛЧМ-сигнала (Sample time) в функциональном блоке Signal From Workspace – 1 с, а временной шаг симуляции в Matlab/Simulink 0.25 с (поле Simulink Sample Time) в блоке BaseClock. Входной сигнал умножим на масштабный множитель 2^3-1, профильтрованный сигнал также делится на это значение. Масштабный множитель выбирается исходя из 4-разрядной точности представления входных отсчетов сигнала. Коэффициенты фильтра умножаются на масштабный множитель 255 и в последующем предусмотрен сдвиг содержимого регистра результата вправо с позиции (22:0) на (22:8), т.е. реализуется деление на 255 (это сделано для будущих расширений, для 4-разрядной точности представления входных отсчетов это можно и не делать).

Для сравнения будем использовать цифровой фильтр (функция дискретной фильтрации filter с вектором коэффициентов нерекурсивной части [–2,–1,7,6]) и модель последовательного КИХ-фильтра на 4 отвода (МАС-фильтр), показанную на рис. 3.33.

На рис. 4.27 показан фрагмент модели КИХ-фильтра на последовательной распределенной арифметике с учетом масштабирования, а на рис. 4.28 ЛЧМ-сигнал (*а*) и имитационное моделирование в системе Matlab/Simulink КИХ-фильтра (*б*), на функции дискретной фильтрации filter (*в*), МАС-фильтр (рис. 3.33) (*г*). Видно, что при невысокой точности представления входных отсчетов и при отсутствии аппаратного умножителя можно получить приемлемые результаты.

Использованием пакета расширения Altera DSP Builder системы визуально-имитационного моделирования Matlab/Simulink позволяет быстро разрабатывать имитационные модели сложных цифровых устройств, таких как КИХ-фильтры, и исследовать информационные потоки, а мегафункция FIR Compiler САПР Quartus II позволяет без значительных усилий реализовывать такие проекты в базис ПЛИС.

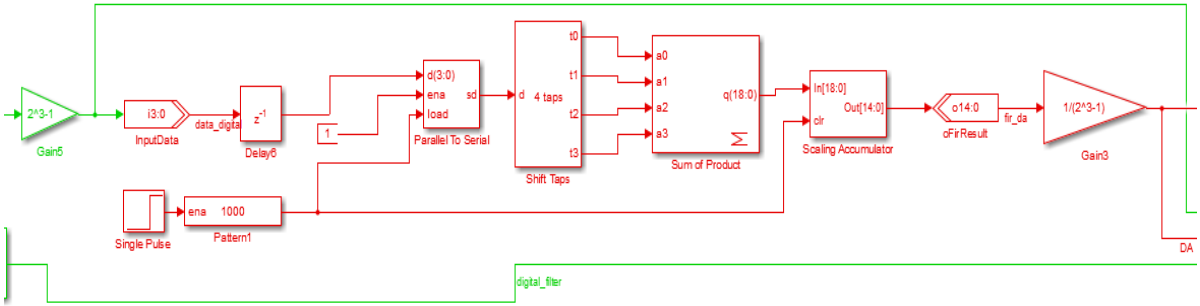


Рис. 4.27. Фрагмент модели КИХ-фильтра на последовательной распределенной арифметике с учетом масштабирования

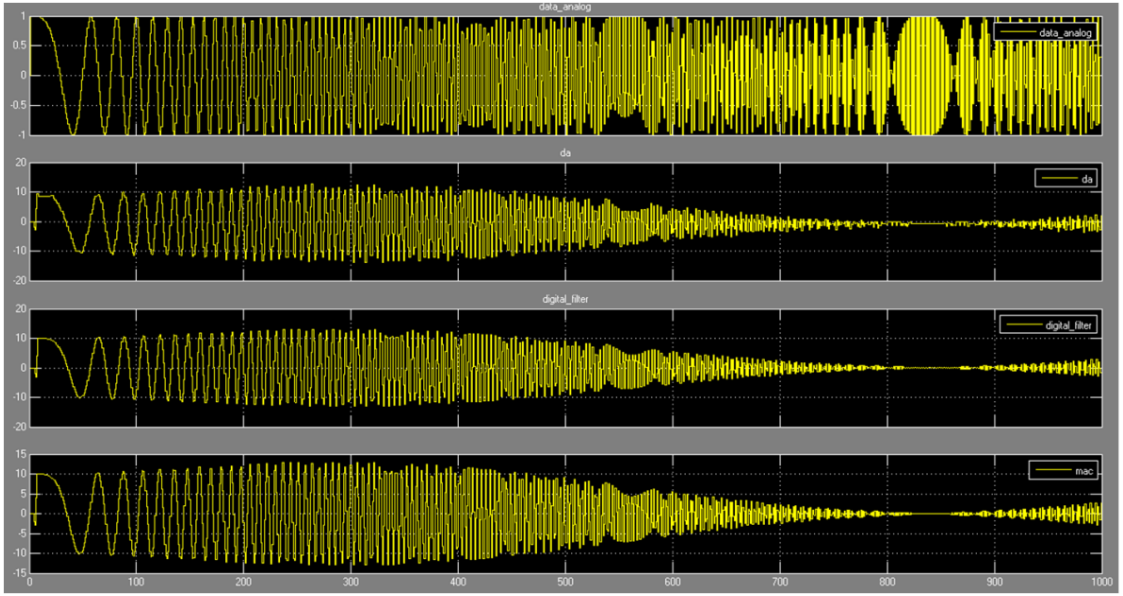


Рис. 4.28. ЛЧМ-сигнал (а) и имитационное моделирование в системе Matlab/Simulink КИХ-фильтра на четыре отвода на основе последовательной распределенной арифметике (б), на функции дискретной фильтрации filter (в), MAC-фильтр (рис.3.33) (г)

4.3. Как устроены промышленные КИХ-фильтры на последовательной распределенной арифметике

Выше рассматривалась простая имитационная модель КИХ-фильтра на 4 отвода с заранее известными коэффициентами, переработанная на основе примера из справочной системы пакета расширения Altera DSP Builder по адресу: quartus\dsp_builder\DesignExamples\Demos\Filters\DA32 с целью изучения базовых принципов распределенной арифметики.

Также была показана модель последовательного КИХ-фильтра на одном блоке умножения и накопления (1 MAC-блок), разработанная на основе примера по адресу: quartus\dsp_builder\DesignExamples\Demos\Filters\Mac32.

Теперь же рассмотрим сложную структуру КИХ-фильтра на 32 отвода. Дадим условное обозначение этому фильтру DA_32, а КИХ-фильтру на 4 отвода – DA_4.

Синтез КИХ-фильтра на 32 отвода осуществляется с помощью функции firls (firls - Least-square linear-phase FIR filter design) в системе Matlab. Функция firls синтезирует нерекурсивный дискретный фильтр с линейной ФЧХ путем минимизации интегральной взвешенной квадратичной ошибки воспроизведения заданной кусочно-линейной АЧХ в заданном диапазоне частотных полос. Ниже приведен код фильтра. При его использовании комментарии, отмеченные значком %, следует удалить. На рис. 4.29 и рис. 4.30 показана импульсная характеристика и АЧХ синтезируемого КИХ-фильтра.

```
% Порядок КИХ-фильтра
FilterOrder = 32
% Точность представления входных отсчетов
InputBitWidth = 8
```

```
% Частотный диапазон полос, вектор пар частотных точек, лежащих в диапазоне от 0 до 1 (1 соответствует частоте Найквиста, то есть половине частоты дискретизации)
LowPassFreqBand = [0 0.1 0.2 1];
% Вектор, содержащий значения амплитуд АЧХ для частот вектора LowPassFreqBand
LowPassMagnBand = [1 0.9 0.0001 0.0001];
% Вычисление значений вектора b, содержащего n+1 коэффициент дискретного КИХ-фильтра с линейной ФЧХ типа II (при четном n) путем минимизации интегральной взвешенной квадратичной ошибки воспроизведения заданной кусочно-линейной АЧХ в заданном диапазоне частотных полос (33 коэффициента)
FlCoef = firls(FilterOrder,LowPassFreqBand,LowPassMagnBand);
% Точность представления коэффициентов фильтра 16 бит
CoefBitWidth = InputBitWidth +
ceil(log2((max(abs(FlCoef))/min(abs(FlCoef)))))
% Вычисление масштабного множителя (32767)
ScalingFactor = (2^(CoefBitWidth-1))-1;
% Вычисление значений коэффициентов с учетом масштабирования и округления
FpCoef = fix(ScalingFactor * FlCoef);
% Построение импульсной характеристики, АЧХ
plot(FpCoef,'o');
title('Coefficient Value');
% Формирование единичного импульса с амплитудой 100,0,0 и т.д. до 1000 значений
ImpulseData = zeros(1,1000);
ImpulseData(1) = 100;
% Вычисление свертки
h = conv(ImpulseData,FpCoef);
% Дискретное преобразование Фурье для вектора h
fftplot(h);
title('Frequency response');
FirSamplingPeriod=1;
```

Пример. Синтез КИХ-фильтра с помощью функции firls системы Matlab

Рассмотрим некоторые команды более подробно. С помощью функции получим значения коэффициентов фильтра до масштабирования:

FlCoef = firls(FilterOrder,LowPassFreqBand,LowPassMagnBand)

Значения коэффициентов фильтра до масштабирования:
0.0047 0.0054 0.0043 0.0009 –0.0045 –0.0108 –0.0161 –0.0181 –0.0146 –0.0040 0.0142 0.0388 0.0672 0.0958 0.1202 0.1367 0.1426 0.1367 0.1202 0.0958 0.0672 0.0388 0.0142 –0.0040 –0.0146 –0.0181 –0.0161 –0.0108 –0.0045 0.0009 0.0043 0.0054 0.0047
Определим точность представления коэффициентов фильтра:
CoefBitWidth = InputBitWidth +
ceil(log2((max(abs(FlCoef))/min(abs(FlCoef)))))
CoefBitWidth = 16
Вычислим масштабный множитель с учетом точности представления коэффициентов фильтра:

ScalingFactor = (2^(CoefBitWidth-1))-1

Значения коэффициентов с учетом масштабирования, округленные в меньшую сторону до ближайшего целого с масштабным множителем ScalingFactor= 32767, симметричны относительно центральной величины 4671:

FpCoef = fix(ScalingFactor * FlCoef)
155, 177, 140, 29, –146, –352, –526, –594, –479, –131, 464, 1270, 2203, 3138, 3938, 4478, 4671, 4478, 3938, 3138, 2203, 1270, 464, –131, –479, –594, –526, –352, –146, 29, 140, 177, 155

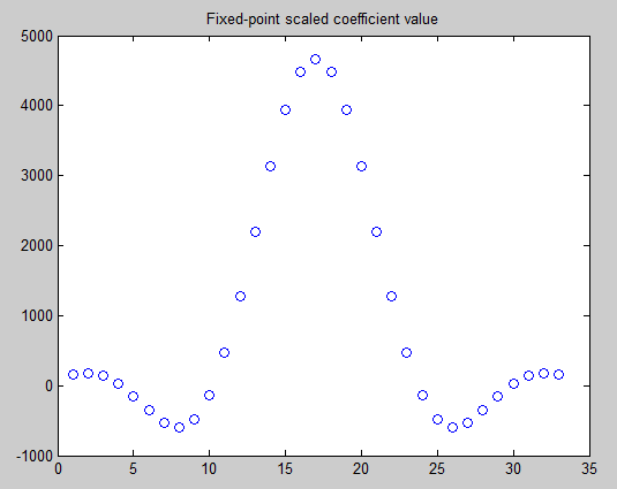


Рис. 4.29. Импульсная характеристика КИХ-фильтра

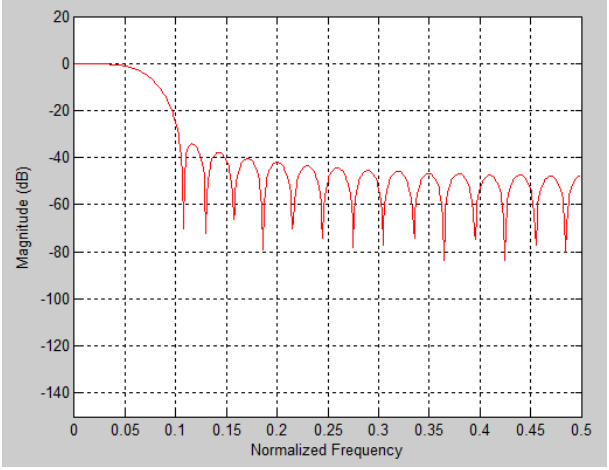


Рис. 4.30. АЧХ синтезируемого КИХ-фильтра

На рис. 4.31 показана имитационная модель КИХ-фильтра на последовательной распределенной арифметике для реализации в базисе ПЛИС Altera. Модель из справочной системы настроена на фильтрацию синусоидального сигнала с шумом.

Рассмотрим блок BaseClock. Для функционального моделирования в Altera-ModelSim задается синхросигнал с периодом 20 нс (поле Real-World Clock Period) и временной шаг симуляции в Matlab/Simulink 1 с (поле Simulink Sample Time) в блоке BaseClock.

Интервал дискретизации входного сигнала (Sample time или время выборки) в функциональном блоке Sine Wave – 8 с (соответствует числу битов в блоке Pattern). На рис. 4.32 показано формирование синусоидального сигнала с амплитудой 100, отчетом на период 100, интервалом дискретизации 8 с, а на рис. 4.33 – шумоподобный сигнал (случайные числа подчиняющиеся нормальному распределению (распределению Гаусса)) с временем выборки 1 с.

Далее эти два сигнала складываются, что в итоге дает зашумленный синосоидальный сигнал, который и подвергается фильтрации.

Точность представления входных отсчетов 8 бит. Отметим, что это невысокая точность по сравнению с точностью представления входных отсчетов для КИХ-фильтра на 32 отвода на основе MAC-блока (пример MAC32). Входной сигнал и коэффициенты фильтра в этом примере представлены с 17-разрядной точностью. Коэффициенты фильтра рассчитываются с помощью функции fir1 (обратное преобразование Фурье с использованием оконных функций), которая также позволяет синтезировать фильтры с линейной ФЧХ.

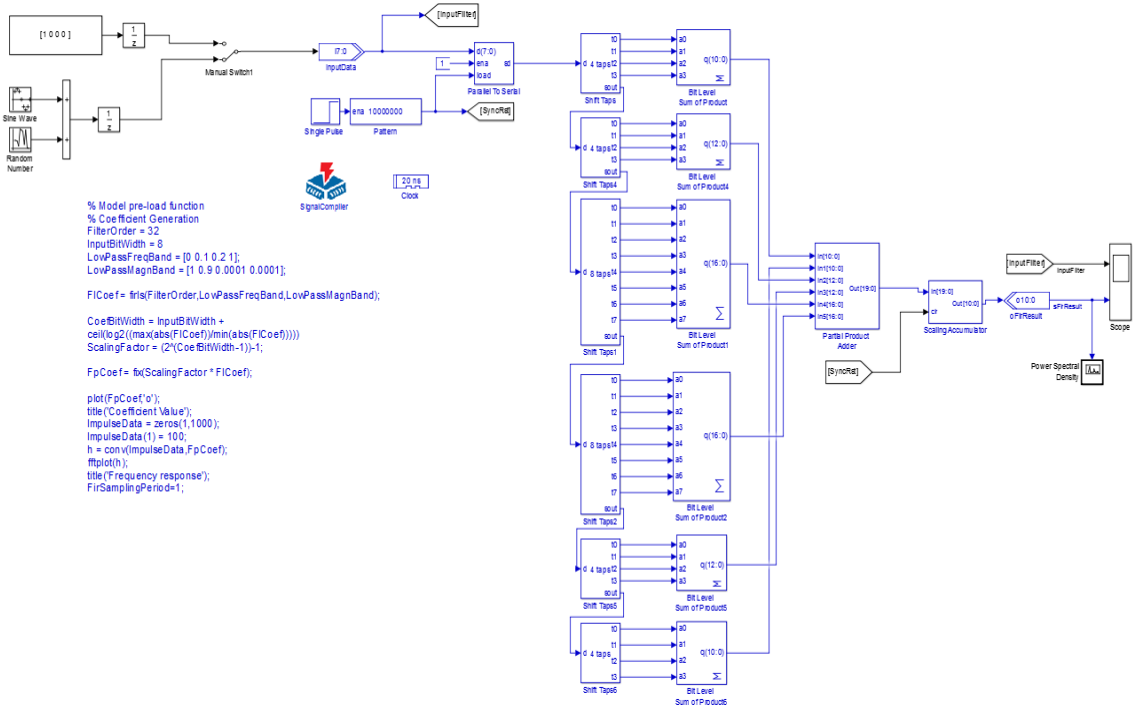


Рис. 4.31. Имитационная модель КИХ-фильтра на последовательной распределенной арифметике для фильтрации зашумленного синусоидального сигнала

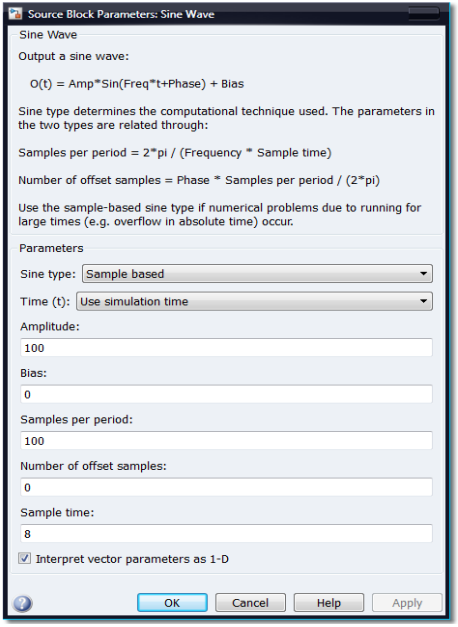


Рис. 4.32. Формирование синусоидального сигнала с амплитудой 100, отчетом на период 100, время выборки 8 с

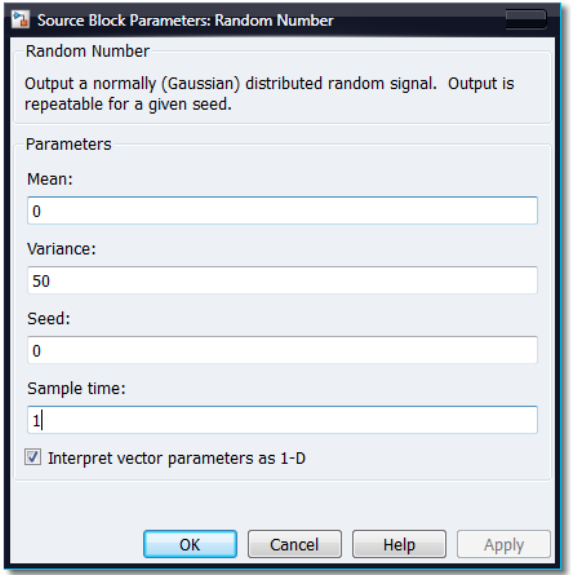


Рис. 4.33. Формирование шумоподобного сигнала, время выборки 1 с

Блок Pattern (формирует периодическую повторяющуюся последовательность битов 10000000) выполняет функцию управляющего цифрового автомата КИХ-фильтра. С одной стороны он управляет загрузкой чисел в 8-разрядный регистр (функциональный блок, преобразующий параллельный код в последовательный с обозначением Parallel To Serial), а с другой – формирует отрицательный вес самого последнего частичного произведения с учетом латентности перед последним суммированием.

С выходов линий задержек на основе функциональных блоков Shift Taps (на основе ОЗУ) с дистанцией 8 снимается 32 отвода. Блоки Shift Taps имеют разное число отводов: 4 блока по 4 и 2 блока по 8. В сумме получается 32 отвода. Поэтому для такого фильтра интервал дискретизации входного сигнала должен быть кратен 8.

Из-за того что синтезированный фильтр имеет 33 коэффициента и коэффициенты симметричны относительно центральной величины, а фильтр в рассматриваемом примере построен на 4- и 8-входовых LUT, то 33 коэффициентом жертвуют. В этом случае коэффициенты перестают быть симметричными, но число коэффициентов становится четным. В этом случае невозможно воспользоваться распространённым приемом с использованием пресумматоров на входах LUT.

Для примера рассмотрим уравнение КИХ-фильтра со структурой 8 отводов 8 бит с несимметричными коэффициентами (входная переменная $x_k(n)$ 8-разрядная):

$$y = P_0 + P_1 \cdot 2^1 + P_2 \cdot 2^2 + P_3 2^3 + P_4 2^4 + P_5 2^5 + P_6 2^6 - P_7 \cdot 2^7 .$$

$$P_0 = X_{00}C_0 + X_{10}C_1 + X_{20}C_2 + X_{30}C_3 + X_{40}C_4 + $$
$$+ X_{50}C_5 + X_{60}C_6 + X_{70}C_7$$
$$P_1 = X_{01}C_0 + X_{11}C_1 + X_{21}C_2 + X_{31}C_3 + X_{41}C_4 + $$
$$+ X_{51}C_5 + X_{61}C_6 + X_{71}C_7 \quad .$$
$$\ldots$$
$$P_7 = X_{07}C_0 + X_{17}C_1 + X_{27}C_2 + X_{37}C_3 + X_{47}C_4 + $$
$$+ X_{57}C_5 + X_{67}C_6 + X_{77}C_7,$$

где $P_0, P_1, P_2, P_3 -$ частичные произведения. Для фильтра с симметричными коэффициентами на примере P_0:

$$P_0 = (X_{00} + X_{70})C_0 + (X_{10} + X_{60})C_1 +$$
$$+ (X_{20} + X_{50})C_2 + (X_{30} + X_{40})C_3.$$

На рис. 4.34а показаны пресумматоры на входах LUT для КИХ-фильтра на 16 отводов с симметричными коэффициентами на последовательной распределенной арифметике. В этом случае требуется 8 последовательных пресумматоров на входах двух 4-входовых LUT. Линия задержки реализована на блоках двухпортовой памяти.

На рис. 4.34б приведена структура такого же фильтра, но коэффициенты уже несимметричные. Имитационная модель КИХ-фильтра на 32 отвода, показанная на рис. 4.31, имеет схожую структуру с КИХ-фильтром на 16 отводов на рис. 4.34б.

В рассматриваемом примере КИХ-фильтра на 32 отвода 4-входовые LUT используются для вычисления комбинаций сумм частичных произведений по формуле:

$$q = C_0a_0 + C_1a_1 + C_2a_2 + C_3a_3,$$

а 8-входовые по формуле:

$$q = C_0a_0 + C_1a_1 + C_2a_2 + C_3a_3 + C_4a_4 + C_5a_5 + C_6a_6 + C_7a_7.$$

Всего используется четыре 4-входовых LUT и два 8-входовых LUT, т.е. 6 LUT с числом входов 32. Таким образом, один 4-входовой LUT с масштабирующим аккумулятором позволяет реализовать КИХ-фильтр со структурой 4 отвода 8 бит.

LUT имеют разную точность вычисления сумм коэффициентов. “Края” импульсной характеристики имеют малые значения коэффициентов, например, рассмотрим левую половину от центра 155, 177, 140, 29 и –146, –352, –526, –594, поэтому достаточно использовать две 4-входовые LUT с точностью представления сумм коэффициентов 9 и 13 бит.

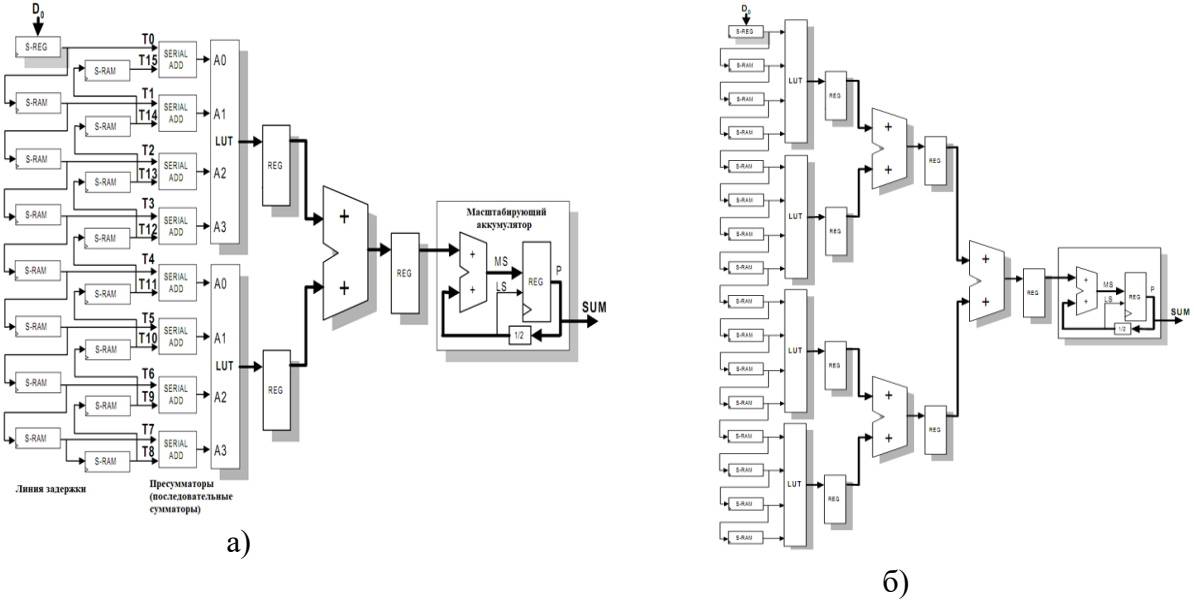


Рис. 4.34. КИХ-фильтра на 16 отводов последовательной распределенной арифметике: а) симметричные коэффициенты и пресумматоры на входах LUT; б) несимметричные коэффициенты

Ближе к центру импульсной характеристики значения коэффициентов увеличиваются и поэтому используются 8-входовые LUT с точностью представления суммы коэффициентов 17 бит: –479, –131, 464, 1270, 2203, 3138, 3938, 4478. Вершина импульсной характеристики 4671. По такому же принципу формируется и правая половина импульсной характеристики. Заполнение LUT для вычисления комбинаций сумм частичных произведений показано в табл. 4.1.

Далее необходимо просуммировать крайние левые и правые суммы частичных произведений с точностью представления сумм 11 разрядов, затем более старшие левые и правые суммы с точностью представления 13 разрядов, затем центральные суммы с точностью представления 17 бит.

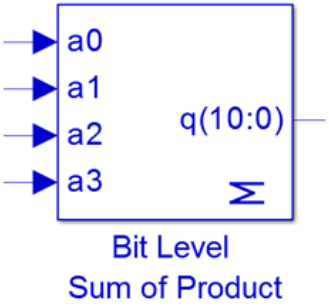
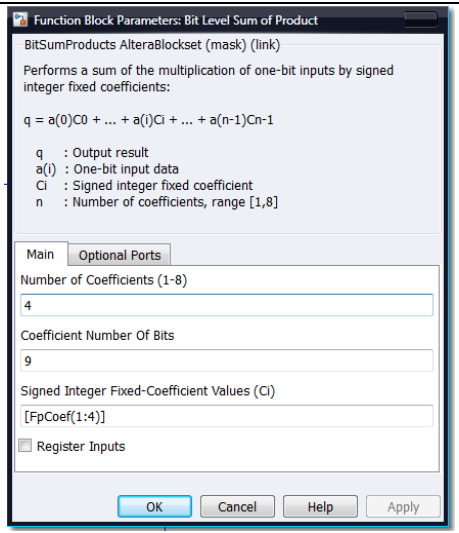
На рис. 4.35 показано суммирование всех сумм частичных произведений с точностью представления результата 20 бит.

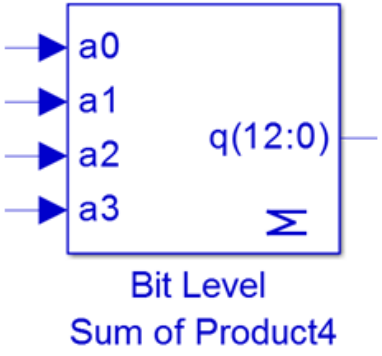
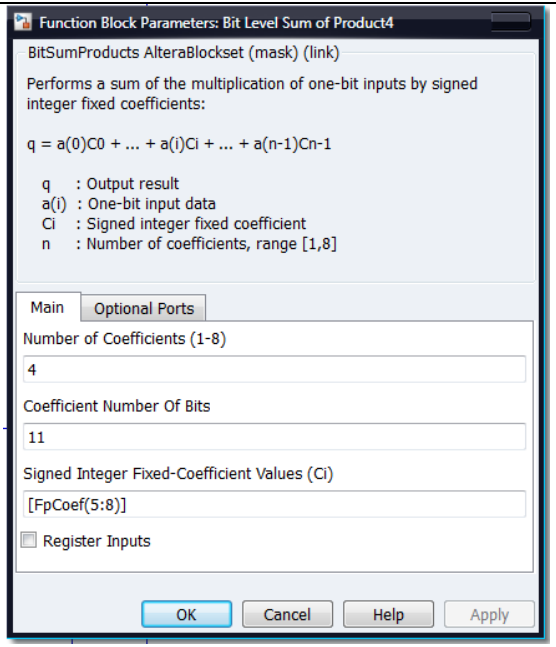
На рис. 4.36 показан масштабирующий аккумулятор КИХ-фильтра. Функциональный блок XOR (Исключающее ИЛИ) совместно с блоком Adder формируют дополнение до двух путем прибавления 1 к результату обращения. Вентиль Исключающее ИЛИ применяется для избирательной инверсии в зависимости от значения управляющего сигнала формируемого блоком Pattern (сигнал clr масштабирующего аккумулятора задерживается на 13 тактов синхрочастоты).

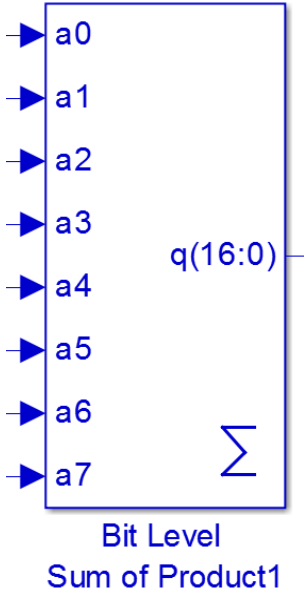
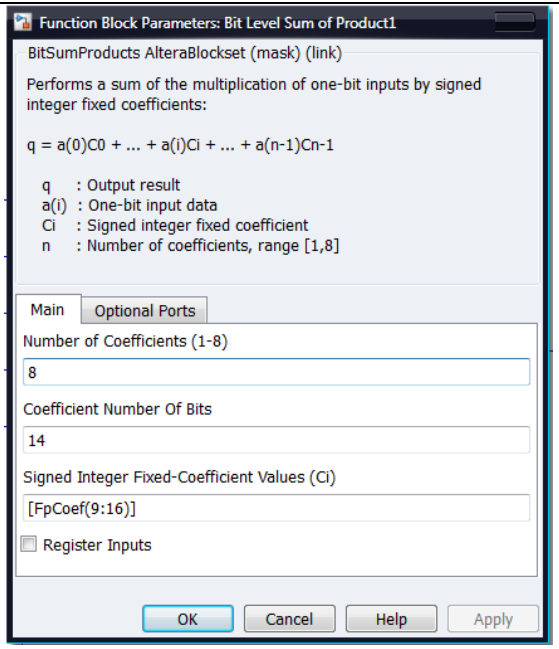
Прибавление 1 к результату обращения реализуется путем формирования единицы на входе переноса c_{in} 21-разрядного сумматора. МЗР результата суммирования, полученный с помощью функционального блока Extract Bit, поступает в блок, выполняющий преобразование последовательной шины (одноразрядный сигнал) в параллельную с обозначением Serial to Parallel (8-разрядный регистр).

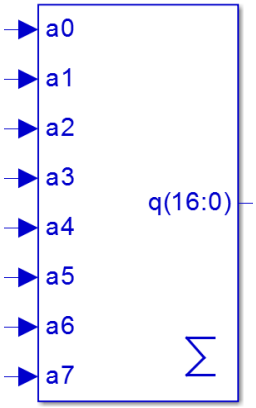
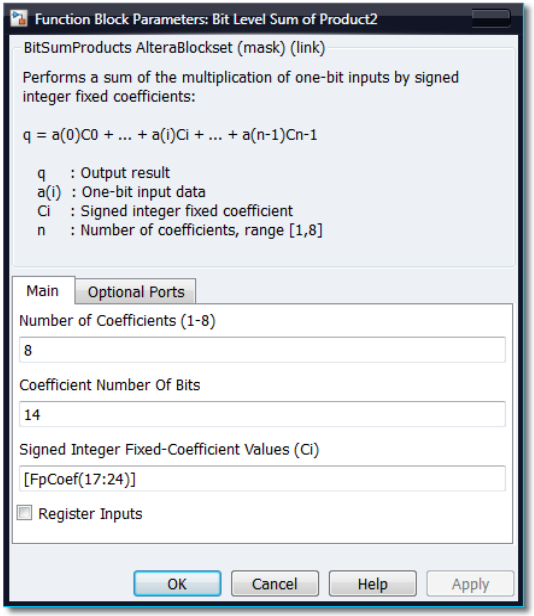
Таблица 4.1

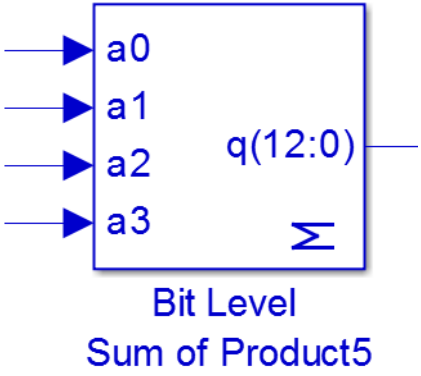
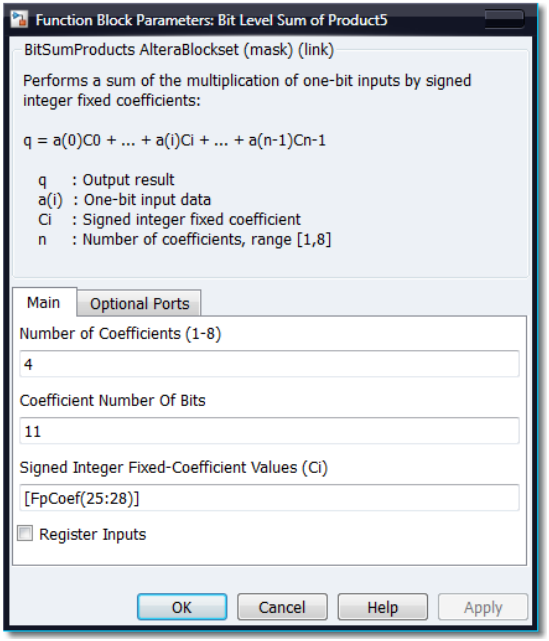
Заполнение LUT для вычисления комбинаций сумм частичных произведений

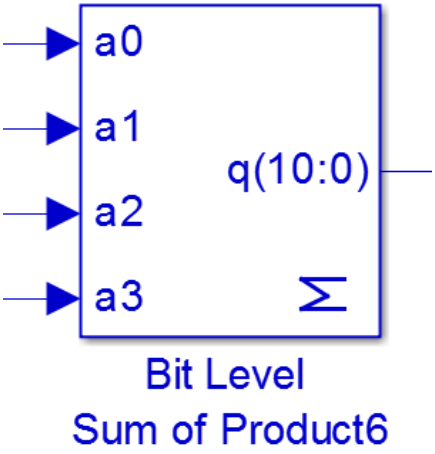
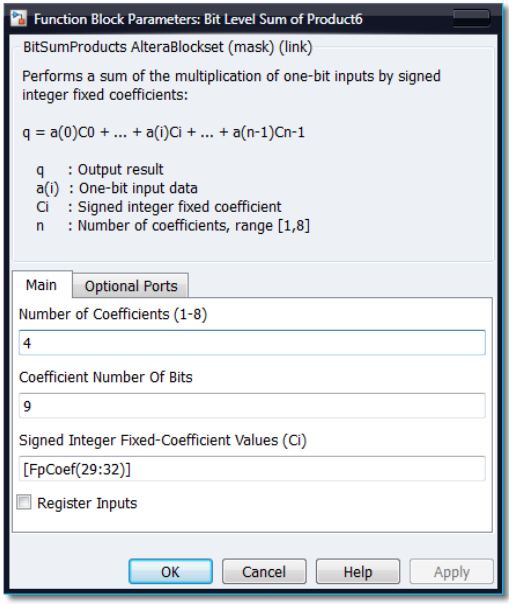
Символ	Настройки функционального блока
1	2
 Условное обозначение 4-LUT с именем Bit Level Sum of Product	 Вариант заполнения 4-LUT: 4 коэффициента, точность представления 9 бит; значения коэффициентов: 155, 177, 140, 29; точность представления сумм частичных произведений 11 бит

1	2
 Условное обозначение 4-LUT с именем Bit Level Sum of Product4	 Вариант заполнения 4-LUT: 4 коэффициента, точность представления 11 бит; значения коэффициентов: –146, –352, –526, –594; точность представления сумм частичных произведений 13 бит

1	2
 Условное обозначение 8-LUT с именем Bit Level Sum of Product1	 Вариант заполнения 8-LUT: 8 коэффициентов, точность представления 14 бит; значения коэффициентов: –479, –131, 464, 1270, 2203, 3138, 3938, 4478; точность представления сумм частичных произведений 17 бит

1	2
 Условное обозначение 8-LUT с именем Bit Level Sum of Product2	 Вариант заполнения 8-LUT: 8 коэффициентов, точность представления 14 бит; значения коэффициентов: 4671, 4478, 3938, 3138, 2203, 1270, 464, –131; точность представления сумм частичных произведений 17 бит

1	2
 Условное обозначение 4-LUT с именем Bit Level Sum of Product5	 Вариант заполнения 4-LUT: 4 коэффициента, точность представления 11 бит; значения коэффициентов: –479, –594, –526, –352; точность представления сумм частичных произведений 13 бит

1	2
 Условное обозначение 4-LUT с именем Bit Level Sum of Product6	 Вариант заполнения 4-LUT: 4 коэффициента, точность представления 9 бит; значения коэффициентов: –146, 29, 140, 177; точность представления сумм частичных произведений 11 бит

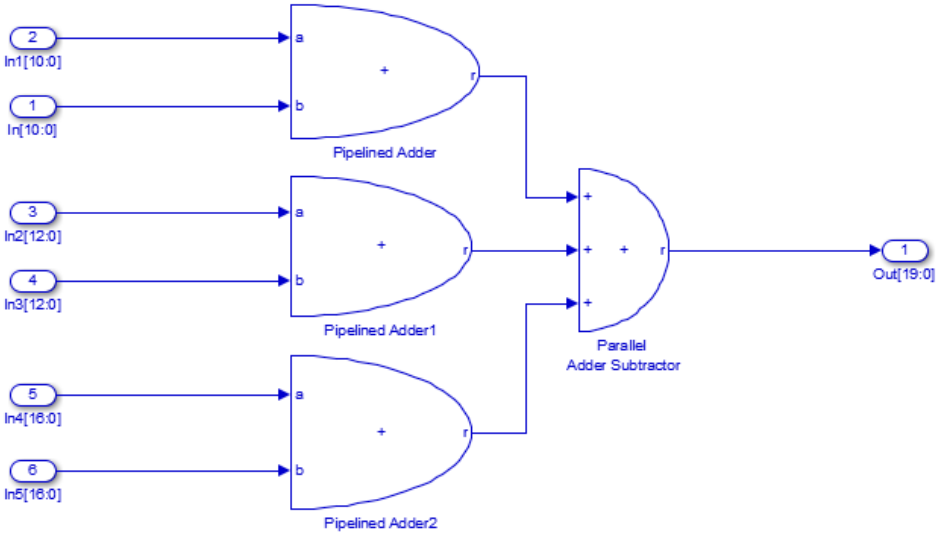


Рис. 4.35. Суммирование всех комбинаций сумм частичных произведений

В настройках функционального блока с обозначением Serial to Parallel задается формат представления чисел после преобразования из последовательного кода в параллельный – целые десятичные числа без знака и то, что первый входящий бит будет представлять МЗР в формируемом 8-разрядном слове.

Сам же 21-разрядный результат суммирования (20:0) задерживается на такт и в функциональном блоке с обозначением Shift by Two осуществляется отбрасывание МЗР, что обеспечивает масштабирование частичных произведений. Далее 20-разрядный сигнал (20:1) поступает по обратной связи на один из входов сумматора (вход a) через мультиплексор управляемый сигналом clr. При этом объединяют старшие разряды a(19:0) и младшие b(7:0) в 28-разрядный сигнал с помощью блока выполняющего функцию конкатенации.

Далее 28-разрядный сигнал поступает в блок осуществляющий представление результатов в требуемой для нас области (Round AlteraBlockset). Назовем такую операцию “округление”. Для того чтобы разобраться как работает этот блок, специально добавим в модель сигнал имитирующий прохождение дельта-функции по структуре фильтра, но заранее нужно обговорить ситуацию, что отмасштабированные коэффициенты КИХ-фильтра мы не увидим. Эта ситуация будет пояснена ниже.

Данный блок осуществляет отбрасывание 17 младших разрядов (16:0) и сохраняет старшие 11 разрядов сигнала (27:17). Данная операция будет равносильна учету того, что коэффициенты КИХ-фильтра подверглись масштабированию. На рис. 4.37 показано число –18674 в дополнительном коде. Красным цветом показаны отбрасываемые 17 младших разрядов, а зеленым выделены оставшиеся 11 старших. Видим, что это десятичное число –1. Подобным образом обрабатывается и зашумленный синусоидальный сигнал. На рис. 4.38 показано удаление шума из сигнала с помощью рассматриваемой модели КИХ-фильтра.

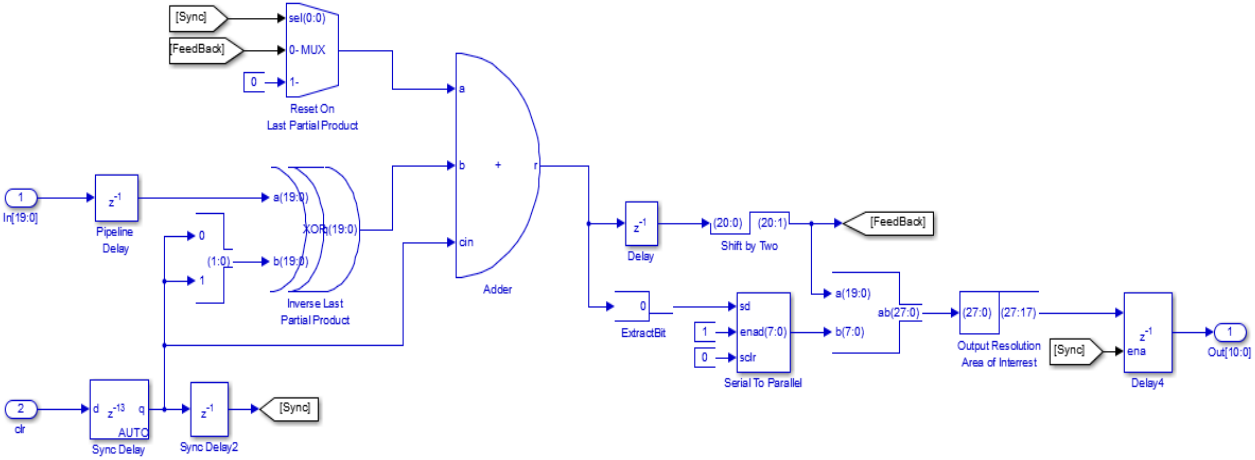


Рис. 4.36. Масштабирующий аккумулятор КИХ-фильтра

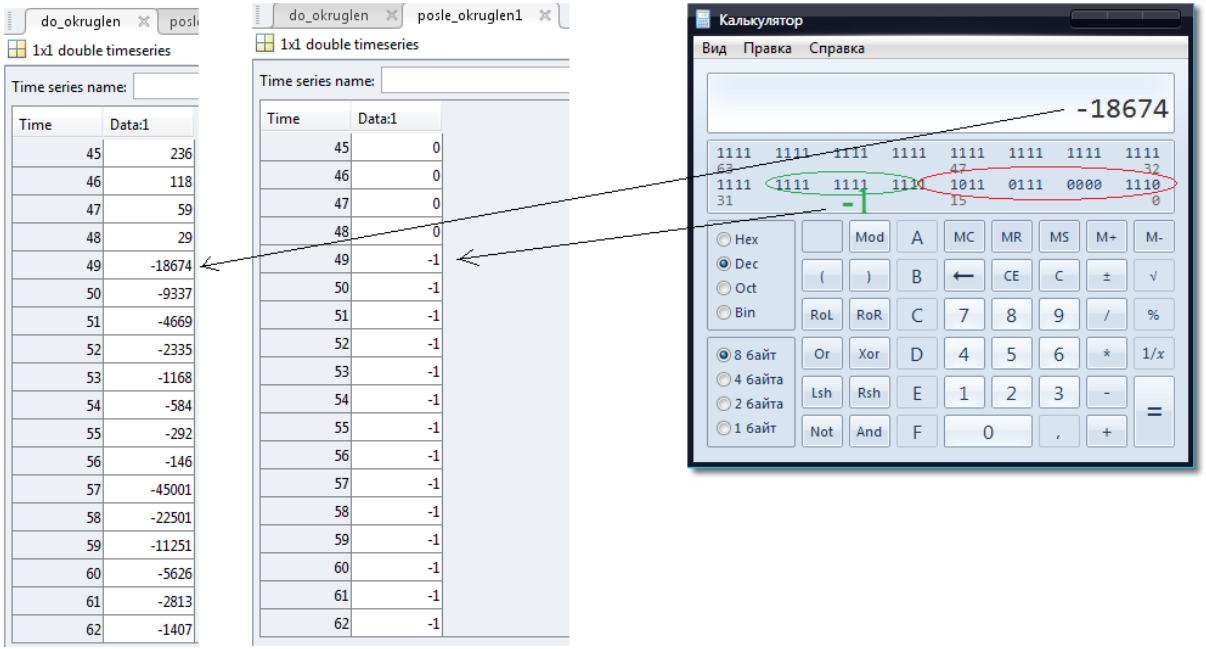


Рис. 4.37. Пример операции округления для отрицательных чисел, представленных в дополнительном коде

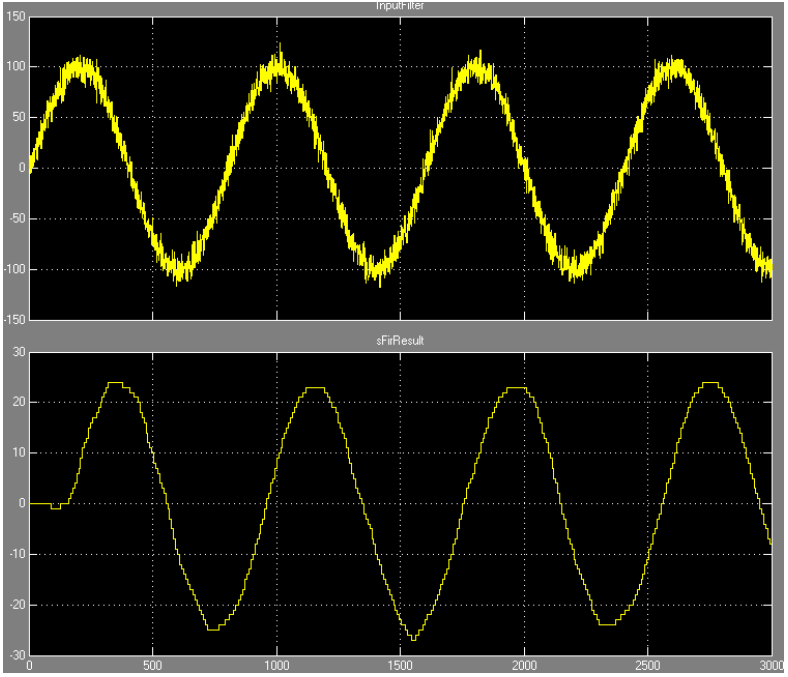


Рис. 4.38. Удаление шума из сигнала с помощью модели последовательного КИХ-фильтра на 32 отвода с использованием последовательной распределенной арифметики

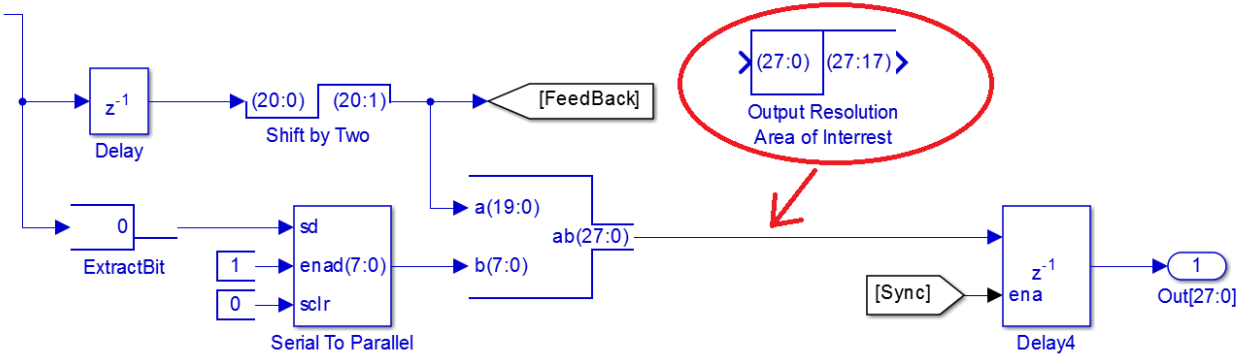


Рис. 4.39. Для моделирования прохождения дельта-функции по структуре фильтра, необходимо удалить функциональный блок, который отбрасывает 17 младших бит (вектор (16:0)) из вектора размерностью (27:0)

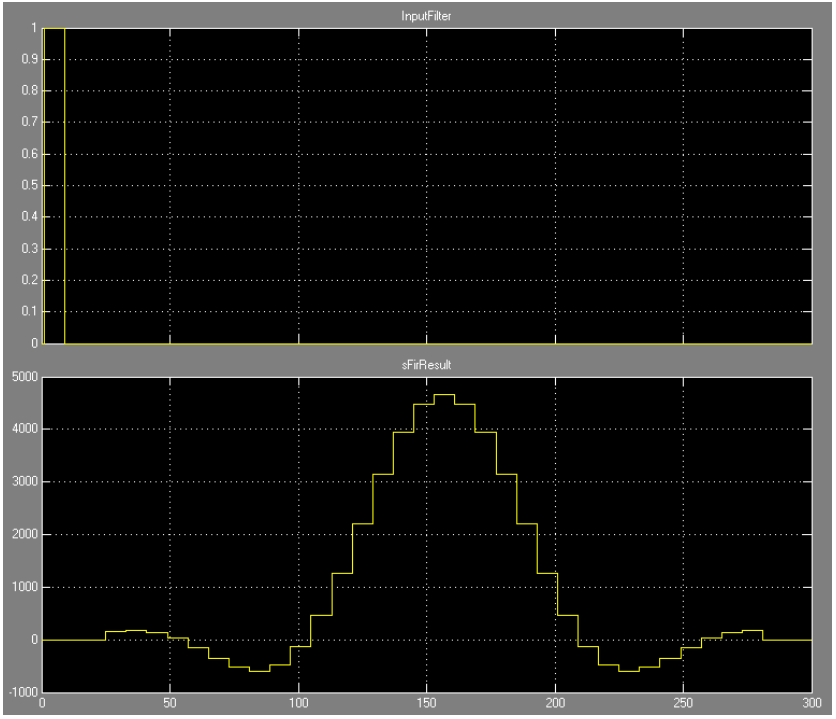


Рис. 4.40. Импульсная характеристика КИХ-фильтра при подачи единичного импульса на вход

Если необходимо построить импульсную характеристику КИХ-фильтра, то необходимо в масштабирующем аккумуляторе удалить функциональный блок, который отбрасывает 17 младших бит (вектор (16:0)) из вектора размерностью (27:0). На рис. 4.39 удаляемый блок выделен красным цветом. На рис. 4.40 показано моделирование импульсной характеристики. При подачи дельта-функции (единичного импульса) на вход на выходе фильтра появляются его отмасштабированные коэффициенты

155, 177, 140, 29, –146, –352, –526, –594, –479, –131, 464, 1270, 2203, 3138, 3938, 4478, 4671, 4478, 3938, 3138, 2203, 1270, 464, –131, –479, –594, –526, –352, –146, 29, 140, 177;

без последнего (155), который отбрасывается.

Покажем, что с помощью небольших изменений можно настроить КИХ-фильтр на 32 отвода на работу КИХ-фильтра на 4 отвода и моделирования ЛЧМ-сигнала (модель DA_32). Сравним результаты фильтрации с доработанным КИХ-фильтром на 4 отвода рассмотренным выше.

На рис. 4.41 показана настройка содержимого самой крайней левой 4-входовой LUT [–2, –1, 7, 6] по формуле:

$$q = C_0 a_0 + C_1 a_1 + C_2 a_2 + C_3 a_3 = -2a_0 - 1a_1 + 7a_2 + 6a_3 .$$

Остальные пять LUT с различными входами заполняются нулями (рис. 4.42). На рис. 4.43а показано, какие изменения нужно внести в модель. Добавляемые функциональные блоки, связанные с частотой, отмечены красным цветом. Входной сигнал КИХ-фильтра на 4 отвода нужно умножить на масштабный множитель 2^3-1. Масштабный множитель выбирался исходя из 4-разрядной точности представления входных отсчетов сигнала. Результат фильтрации также необходимо разделить на этот множитель.

Интервал дискретизации ЛЧМ-сигнала (Sample time) для модели DA_4 в функциональном блоке Signal From Workspace – 1 с. Временной шаг симуляции в Matlab/Simulink 0.25 с (поле Simulink Sample Time) в блоке BaseClock (рис. 4.44а). Интервал дискретизации ЛЧМ-сигнала выбирается из соображения 0.25 с*4, что составит 1 с.

Также нужно использовать ФАПЧ для генерации дочернего синхросигнала. Для этого в модель DA_32 добавляем блок Clock AlteraBlockset. В поле Base Clock Multiplicand Denomerator функционального блока Clock AlteraBlockset установим значение 2 (SymbolClock). С помощью функционального блока Tsamp будет установлен шаг симуляции 0.25/2 (0.125 с) для модели DA_32. Интервал дискретизации ЛЧМ-сигнала для модели DA_32 выбирается из соображения 0.125 с*8, что составит 1 с.

Таким образом, с помощью ФАПЧ при одинаковом интервале дискретизации ЛЧМ-сигнала модели DA_32 и DA_4 будут работать на разных частотах (скоростях), но результат фильтрации КИХ-фильтры будут демонстрировать одинаковый (рис. 4.45).

Приведенная в справочной системе пакета расширения Altera DSP Builder структура КИХ-фильтра на 32 отвода с использованием распределенной последовательной арифметики может быть адаптирована под конкретные задачи пользователя. Рассмотренный пример демонстрирует использование функции firls для синтеза нерекурсивного дискретного фильтра с линейной ФЧХ. Показано как с помощью такого фильтра можно удалить шум из сигнала и фильтровать ЛЧМ-сигнал.

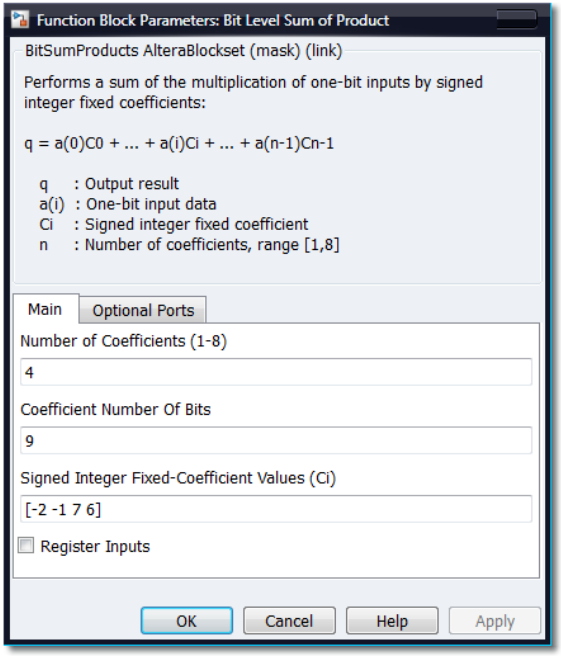


Рис. 4.41. Настройка содержимого 4-входовой LUT

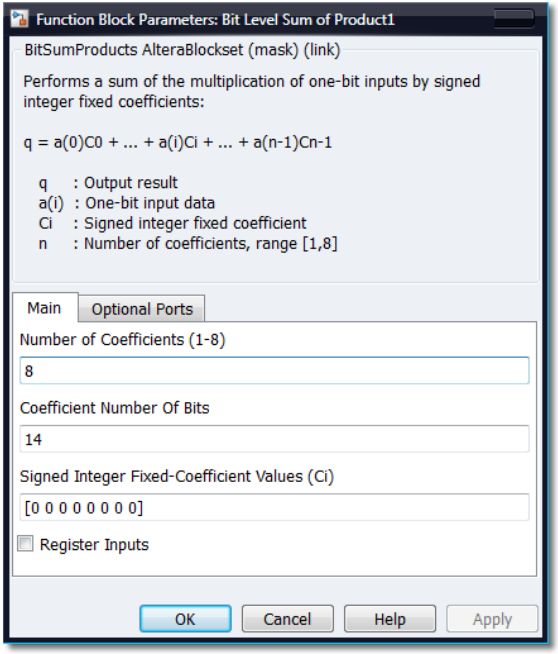
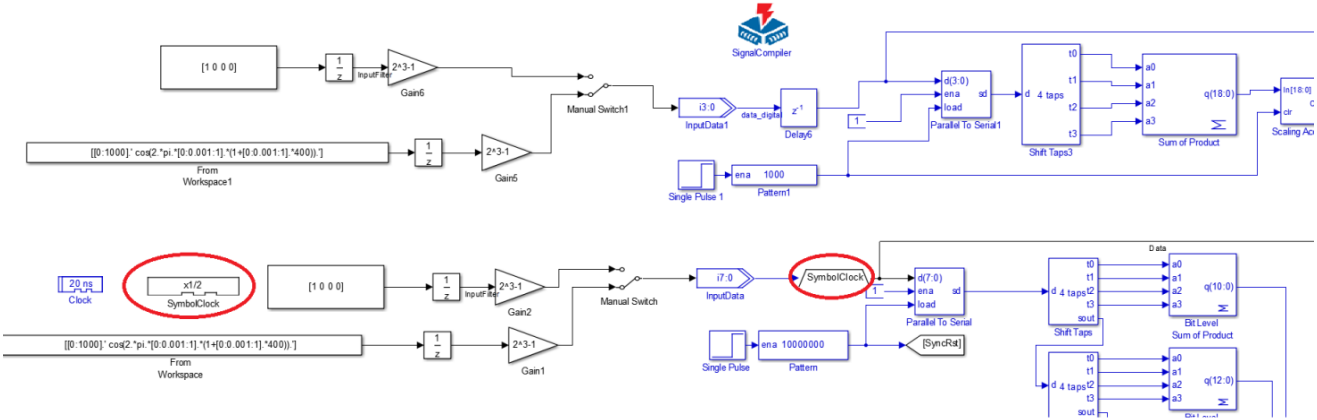
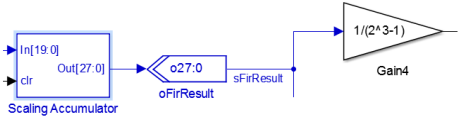


Рис. 4.42. Настройка содержимого 8-входовой LUT



а)



б)

Рис. 4.43. Измения в модели DA_32 для настройки ее на работу КИХ-фильтра на 4 отвода и моделирования фильтрации ЛЧМ-сигнала: а) на входе; б) на выходе

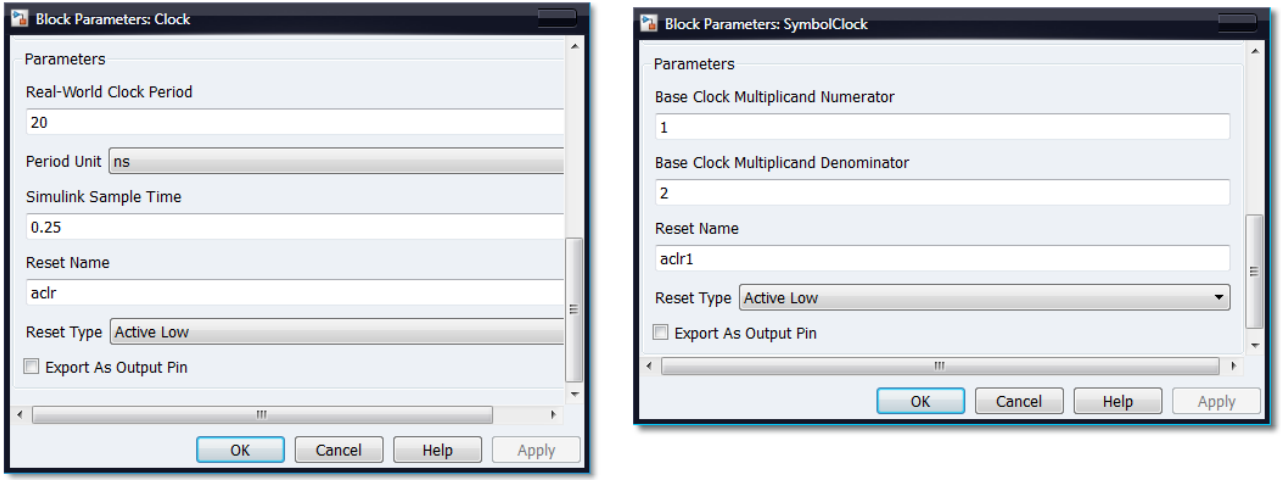


Рис. 4.44. Задание синхросигналов в модели: а) задание шага симуляции в Simulink 0.25 с с помощью блока BaseClock; б) формирование дочернего синхросигнала для модели на основе КИХ-фильтра на 32 отвода из базового синхросигнала с помощью функционального блока Clock AlteraBlockset

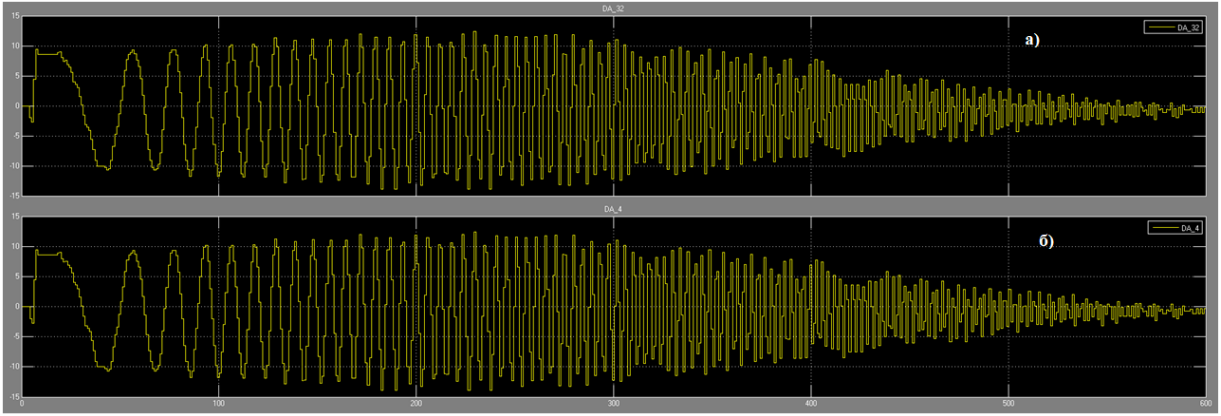


Рис. 4.45. Имитационное моделирование фильтрации ЛЧМ-сигнала КИХ-фильтрами, модель DA_32 (а) и DA_4 (б)

4.4. КИХ-фильтры на параллельной распределенной арифметике

Использование КИХ-фильтров на параллельной распределенной арифметике позволяет для ПЛИС получать рекордное быстродействие за счет использования «безумножительных» схем умножения, не снижаемое при увеличении числа отводов и коэффициентов фильтра, а также точности представления входных данных. Что особенно актуально для проектов, использующих бюджетные серии зарубежных ПЛИС и особенно отечественных, в которых отсутствуют аппаратные умножители или их количество ограничено, но обладающих большим количеством логических ресурсов или блочной памяти.

Цель данного раздела – показать, что основой КИХ-фильтра на параллельной распределенной арифметике является параллельный векторный умножитель, реализация которого в базисе ПЛИС позволяет получить максимальный выигрыш по быстродействию.

Уравнение КИХ-фильтра (нерекурсивного цифрового фильтра с конечно-импульсной характеристикой) представляется как арифметическая сумма произведений:

$$y = \sum_{k=1}^{K} C_k \cdot x_k \,, \qquad (4.6)$$

где y – отклик цепи; x_k – k-я входная переменная; C_k – весовой коэффициент k-й входной переменной, который является постоянным для всех n; K – число отводов фильтра.

Если рассматривать входные переменные x_k как целые десятичные числа со знаком в дополнительном двоичном коде, то

$$x_k = -x_{k(\mathrm{B}-1)} 2^{B-1} + \sum_{b=0}^{B-2} x_{kb} \cdot 2^b \,, \qquad (4.7)$$

где B – разрядность кода; $x_{k(\text{B}-1)}2^{B-1}$ – знаковый разряд. Подставим выражение (4.7) в (4.6), получим

$$y = \sum_{k=1}^{K} C_k \cdot \left[- x_{k(\text{B}-1)} 2^{B-1} + \sum_{b=0}^{B-2} x_{kb} \cdot 2^b \right] = \\ = -\sum_{k=1}^{K} x_{k(B-1)} 2^{B-1} C_k + \sum_{k=1}^{K} \sum_{b=0}^{B-2} x_{kb} 2^b C_k. \tag{4.8}$$

Раскроем все суммы в выражении (4.8) и сгруппируем числа по степеням B:

$$y = \left[x_{10} \cdot C_1 + x_{20} \cdot C_2 + x_{30} \cdot C_3 + \ldots + x_{K0} \cdot C_K\right] \cdot 2^0 + \\ + \left[x_{11} \cdot C_1 + x_{21} \cdot C_2 + x_{31} \cdot C_3 + \ldots + x_{K1} \cdot C_K\right] \cdot 2^1 + \\ + \left[x_{12} \cdot C_1 + x_{22} \cdot C_2 + x_{32} \cdot C_3 + \ldots + x_{K2} \cdot C_K\right] \cdot 2^2 + \\ \ldots. \\ \ldots. \\ + \left[x_{1(B-2)} \cdot C_1 + x_{2(B-2)} \cdot C_2 + x_{3(B-2)} \cdot C_3 + \ldots + x_{K(B-2)} \cdot C_K\right] \cdot 2^{B-2} - \\ - \left[x_{1(B-1)} \cdot C_1 + x_{2(B-1)} \cdot C_2 + x_{3(B-1)} \cdot C_3 + \ldots + x_{K(B-1)} \cdot C_K\right] \cdot 2^{B-1}. \tag{4.9}$$

Каждое выражение в квадратных скобках представляет собой сумму операций И над одним разрядом входной переменной x_k, определяемую весовым фактором B всех k входных переменных и всеми битами весовых коэффициентов C_k.

Для структуры КИХ-фильтра с восемью отводами на распределенной арифметике с несимметричными коэффициентами выражение в квадратных скобках для индекса b может быть записано в виде

$$[sumb] = x_{1b} \cdot C_1 + x_{2b} \cdot C_2 + x_{3b} \cdot C_3 + x_{4b} \cdot C_4 + \\ + x_{5b} C_5 + x_{6b} C_6 + x_{7b} C_7 + x_{8b} C_8,$$

и для фильтра с симметричными коэффициентами:

$$[sumb] = (x_{1b} + x_{8b}) \cdot C_1 + (x_{2b} + x_{7b}) \cdot C_2 + \\ + (x_{3b} + x_{6b}) \cdot C_3 + (x_{4b} + x_{5b}) \cdot C_4.$$

Рассмотрим построение КИХ-фильтра на основе параллельной распределённой арифметики на примере структуры: восемь отводов восемь бит. Перепишем уравнение (4.9) в следующем виде:

$$y = [sum0] + [sum1]2^1 + [sum2] \cdot 2^2 + \ldots.. + \\ + [sum6] \cdot 2^6 - [sum7] \cdot 2^7, \qquad (4.10)$$

при этом под обозначениями sum0, sum1 и т. д. подразумеваются выражения, заключённые соответственно в первых квадратных скобках, во вторых и т. д.

Преобразуем содержимое формулы (4.10) в массив подобных сумм на основе двухвходовых сумматоров:

$$y = \{[sum0] + [sum1]2^1\} + \{[sum2] + [sum3]2^1\}2^2 + \\ + \{[sum4] + [sum5]2^1\}2^4 + \{[sum6] - [sum7]2^1\}2^6 \qquad (4.11)$$

или

$$y = \{\{[sum0] + [sum1]2^1\} + \{[sum2] + [sum3]2^1\}2^2\} + \\ + \{\{[sum4] + [sum5]2^1\} + \{[sum6] - [sum7]2^1\}2^2\}2^4. \qquad (4.12)$$

Разница между принципами параллельной и последовательной распределённой арифметики состоит в числе последовательных масштабирующих суммирований значений квадратных скобок за период изменения входного отсчёта. В случае параллельной распределённой арифметики необходимо иметь B идентичных массивов памяти, параллельно адресуемых всеми битами всех входных переменных, и дерево масштабирующих сумматоров, осуществляющих соответствующее суммирование всех значений квадратных скобок. В данном случае результат формируется за один такт и тем самым достигается наибольшее быстродействие структуры.

Вышеприведенные уравнения (4.11) и (4.12) полностью эквивалентны по своему значению, однако в последнем случае имеется возможность построения свёртывающего иерархического дерева многоразрядных сумматоров, что намного упрощает введение конвейера и достижение максимального быстродействия. В итоге всё дерево будет включать в себя восемь многоразрядных сумматоров. Данная структура на основе параллельной распределённой арифметики обеспечивает практически предельное быстродействие при значительном объёме задействованных ресурсов.

Если рассматривать входные переменные x_k в формате с фиксированной запятой (x_k – дробные значения, $|x_k|<1$, x_{k0} – знаковый разряд), то

$$x_k = -x_{k0} + \sum_{b=1}^{B-1} x_{kb} \cdot 2^{-b},$$

$$\begin{aligned} y(n) &= \sum_{k=1}^{K} C_k \cdot \left[-x_{k0} + \sum_{b=1}^{B-1} x_{kb} \cdot 2^{-b} \right] = \\ &= -\sum_{k=1}^{K} x_{k0} \cdot C_k + \sum_{k=1}^{K} \sum_{b=1}^{B-1} x_{kb} \cdot C_k \cdot 2^{-b}. \end{aligned} \qquad (4.13)$$

Аналогично выражению (4.9) уравнение КИХ-фильтра для формата с фиксированной запятой будет иметь вид

$$\begin{aligned} y(n) = &-\left[x_{10} \cdot C_1 + x_{20} \cdot C_2 + x_{30} \cdot C_3 + \ldots + x_{K0} \cdot C_K\right] \cdot 2^0 + \\ &+ \left[x_{11} \cdot C_1 + x_{21} \cdot C_2 + x_{31} \cdot C_3 + \ldots + x_{K1} \cdot C_K\right] \cdot 2^{-1} + \\ &+ \left[x_{12} \cdot C_1 + x_{22} \cdot C_2 + x_{32} \cdot C_3 + \ldots + x_{K2} \cdot C_K\right] \cdot 2^{-2} + \\ &\ldots. \\ &\ldots. \\ &+ \left[x_{1(B-2)} \cdot C_1 + x_{2(B-2)} \cdot C_2 + x_{3(B-2)} \cdot C_3 + \ldots + x_{K(B-2)} \cdot C_K\right] \cdot 2^{-(B-2)} + \\ &+ \left[x_{1(B-1)} \cdot C_1 + x_{2(B-1)} \cdot C_2 + x_{3(B-1)} \cdot C_3 + \ldots + x_{K(B-1)} \cdot C_K\right] \cdot 2^{-(B-1)}. \end{aligned} \qquad (4.14)$$

Переформулируем выражение (4.14), представим его в следующем виде и сравним с уравнением (4.13):

$$y(n) = -\sum_{k=1}^{K} C_k x_{k0} + \sum_{b=1}^{B-1}\left[\sum_{k=1}^{K} C_k x_{kb}\right] 2^{-b} = \\ = -P_0 + \left[\sum_{b=1}^{B-1} 2^{-b} P_b\right], \tag{4.15}$$

где P – частичные произведения (значения в квадратных скобках выражения (4.14), представляющие комбинации сумм коэффициентов фильтра, которые предварительно вычисляются), а масштабирование частичных произведений может быть параллельным или последовательным.

Видим, что изменение в формулах приводит к перестройке аппаратных ресурсов фильтра (рис. 4.46). По формуле (4.13) получается фильтр с использованием операций умножения с накоплением, а по формуле (4.15) – фильтр на распределенной арифметике без операций явного умножения. А выражения для КИХ-фильтра, представленные целочисленными и дробными значениями, отличаются вычислениями знакового разряда и весовых коэффициентов. Например, для КИХ-фильтра на восемь отводов с целочисленными значениями старший знаковый разряд – это выражение $-[sum7]$ с весом 2^7, а для фильтра с дробными значениями – это $-[sum0]$ с весом 2^0.

Дополнительный код, целочисленные значения:

$$y(n) = \left[\sum_{b=0}^{B-2} 2^b P_b\right] - 2^{B-1} P_{B-1}. \tag{4.16}$$

Дополнительный код, дробные значения:

$$y(n) = -P_0 + \left[\sum_{b=1}^{B-1} 2^{-b} P_b\right]. \tag{4.17}$$

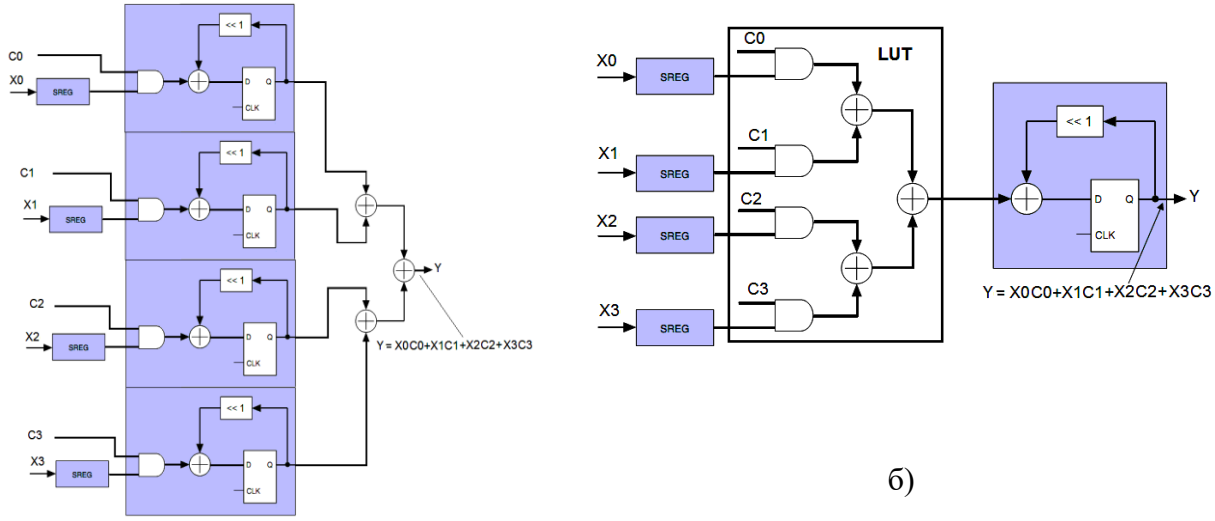


Рис. 4.46. КИХ-фильтр на четыре отвода: а) аппаратная реализация фильтра по формуле (4.13); б) по формуле (4.15) с использованием LUT

Рассмотрим КИХ-фильтр (рис. 4.47) с симметричными коэффициентами (выбираются симметрично относительно центральной величины). В основе стуктуры КИХ-фильтра лежит параллельный векторный умножитель, в качестве которого из-за постоянства коэффициентов используют таблицу перекодировок (LUT). Рассмотрим простейший параллельный векторный умножитель четырех 2-разрядных сигналов на четыре 2-разрядные константы (4-разрядный векторный перемножитель) в предположении, что все величины целочисленные и положительные, представлены в прямом коде (рис. 4.48). Умножение и сложение происходят параллельно с использованием LUT (табл. 4.2).

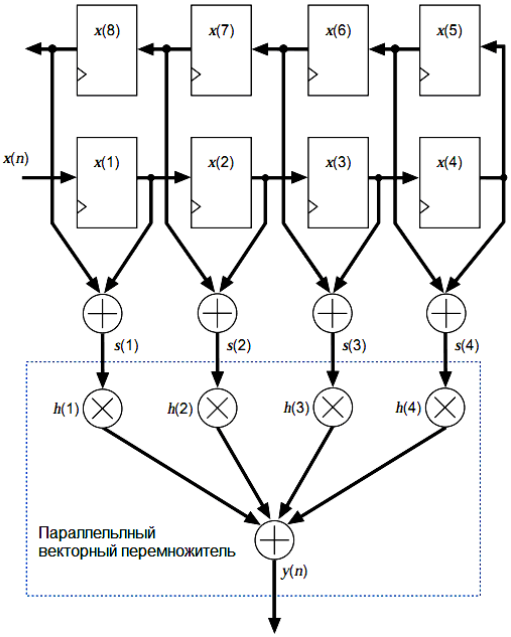


Рис. 4.47. Структура КИХ-фильтра на восемь отводов с симметричными коэффициентами, в основе которой лежит параллельный векторный умножитель

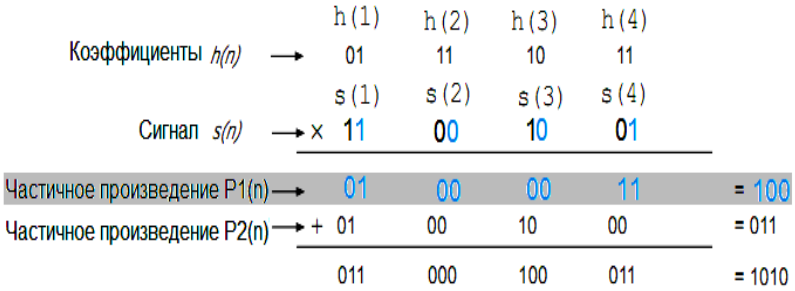


Рис. 4.48. Принцип параллельного векторного умножения

Принцип формирования частичного произведения P1(n) показан на рис. 4.48. Булева функция для формирования P1(n) реализуется таблицей истинности, которая хранится в LUT: $y = [s(1)h(1)] + [s(2)h(2)] + [s(3)h(3)] + [s(4)h(4)]$ Идентичная таблица используется и для формирования P2(n).

Для завершения формирования частичного произведения P2(n) результат необходимо сдвинуть на один разряд влево, что равносильно умножению на 2. Это легко реализовать с помощью сдвиговых регистров. Далее частичные произведения P1(n) и P2(n) необходимо сложить с учетом возможного переполнения. На рис. 4.49 показан параллельный векторный перемножитель четырех 2-разрядных сигналов. Таким образом, требуются две идентичные таблицы, двоичный сдвиг влево и операция суммирования.

На рис. 4.50 показана структура КИХ-фильтра 8 отводов 8 бит на распределенной арифметике с несимметричными и симметричными коэффициентами, обеспечивающими точность вычислений от 8 до 19 бит (полная точность), в основе которой лежит параллельный векторный перемножитель.

Таблица 4.2

Формирование частичного произведения P1(n)

s(n)	P1	y=[s(1)*h(1)]+[s(2)h(2)] +[s(3)h(3)]+[s(4)h(4)]
0000	0	00 + 00 + 00 + 00 = 0000
0001	h(1)	00 + 00 + 00 + 01 = 0001
0010	h(2)	00 + 00 + 11 + 00 = 0011
0011	h(2) + h(1)	00 + 00 + 11 + 01 = 0100
0100	h(3)	00 + 10 + 00 + 00 = 0010
0101	h(3) + h(1)	00 + 10 + 00 + 01 = 0011
0110	h(3) + h(2)	00 + 10 + 11 + 00 = 0101
0111	h(3) + h(2) + h(1)	00 + 10 + 11 + 01 = 0110
1000	h(4)	11 + 00 + 00 + 00 = 0011
1001	h(4) + h(1)	11 + 00 + 00 + 01 = 0100
1010	h(4) + h(2)	11 + 00 + 11 + 00 = 0110
1011	h(4) + h(2) + h(1)	11 + 00 + 11 + 01 = 0111
1100	h(4) + h(3)	11 + 10 + 00 + 00 = 0101
1101	h(4) + h(3) + h(1)	11 + 10 + 00 + 01 = 0110
1110	h(4) + h(3) + h(2)	11 + 10 + 11 + 00 = 1000
1111	h(4) + h(3) + h(2) + h(1)	11 + 10 + 11 + 01 = 1001

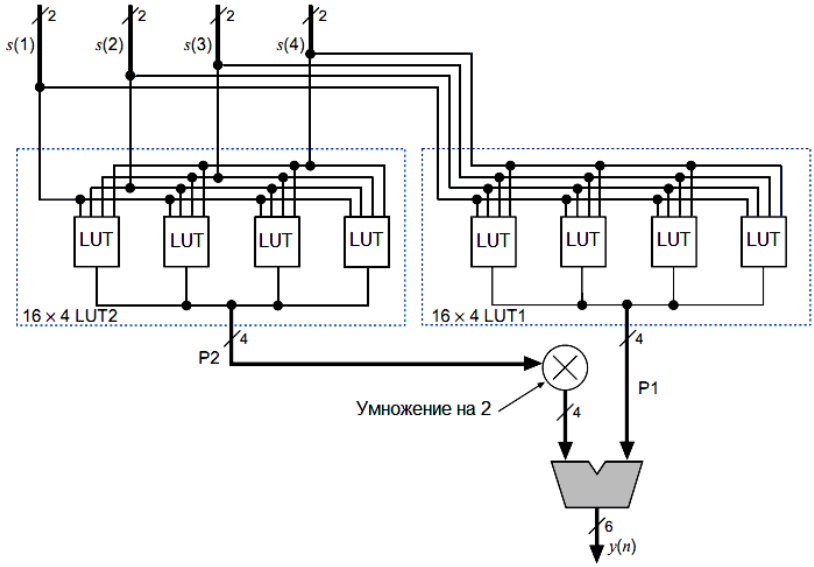
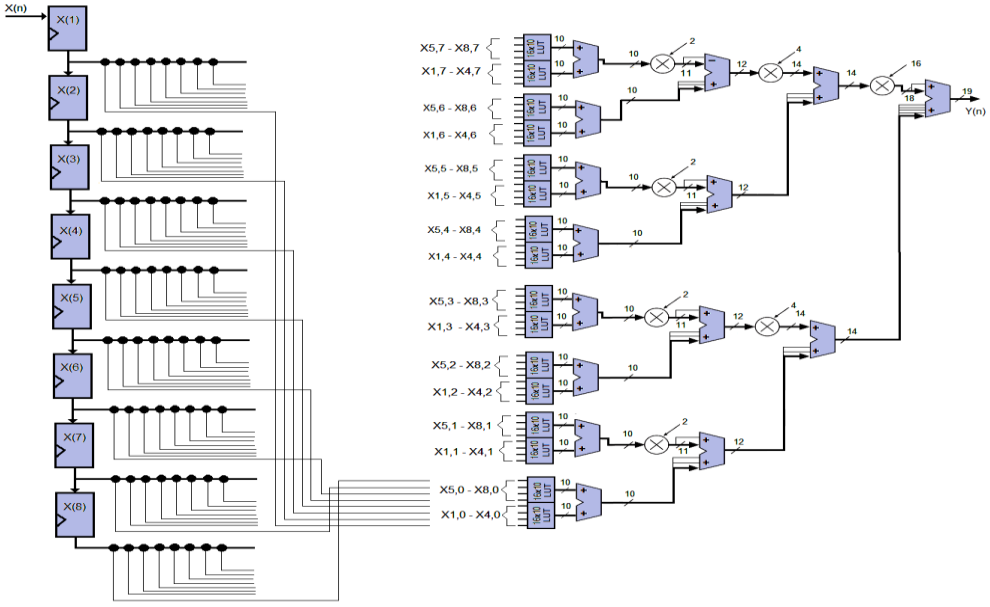
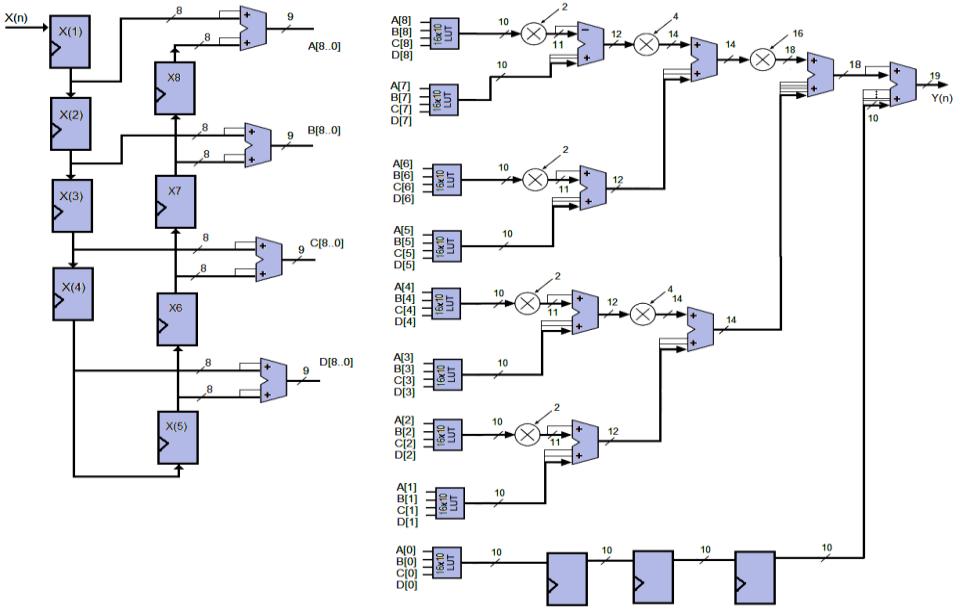


Рис. 4.49. Параллельный векторный умножитель четырех 2-разрядных сигналов на четыре 2-разрядные константы



а)

Рис. 4.50. Структура КИХ-фильтра 8 отводов 8 бит на распределенной параллельной арифметике: а) с несимметричными коэффициентами; б) с симметричными



б)

Рис. 4.50. Структура КИХ-фильтра 8 отводов 8 бит на распределенной параллельной арифметике: а) с несимметричными коэффициентами; б) с симметричными (продолжение)

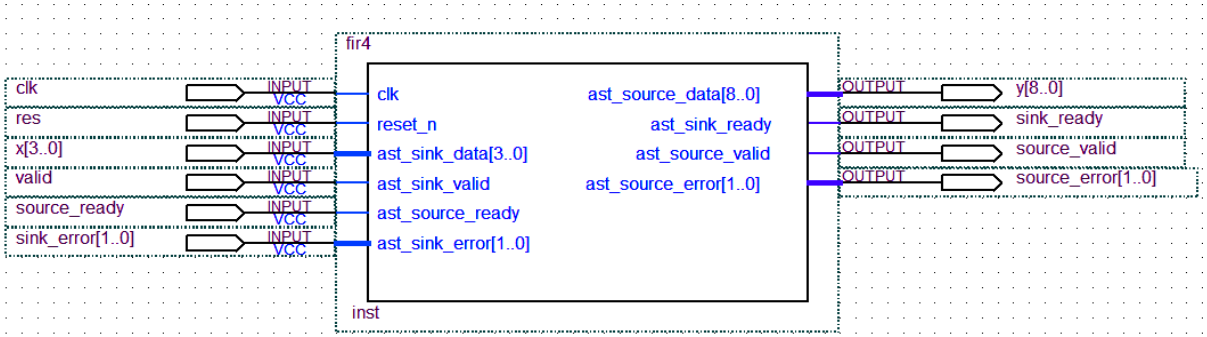


Рис. 4.51.Тестовая схема КИХ-фильтра с использованием мегаядра FIR Compiler на последовательной и параллельной распределенной арифметике

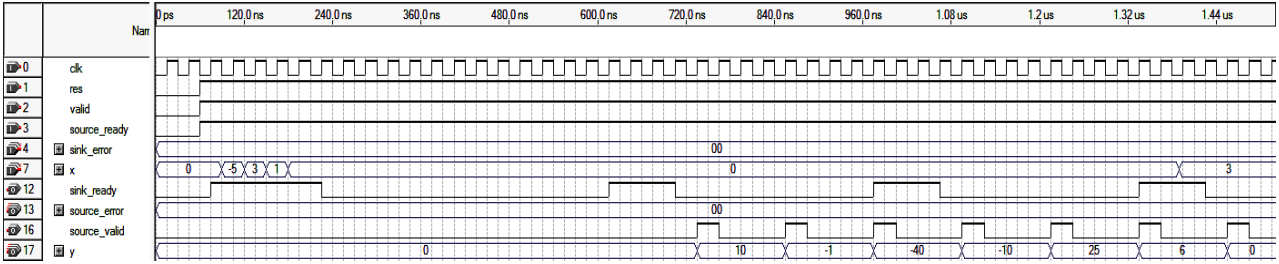


Рис. 4.52. Временные диаграммы работы КИХ-фильтра с использованием последовательной распределенной арифметики на мегаядре FIR Compiler

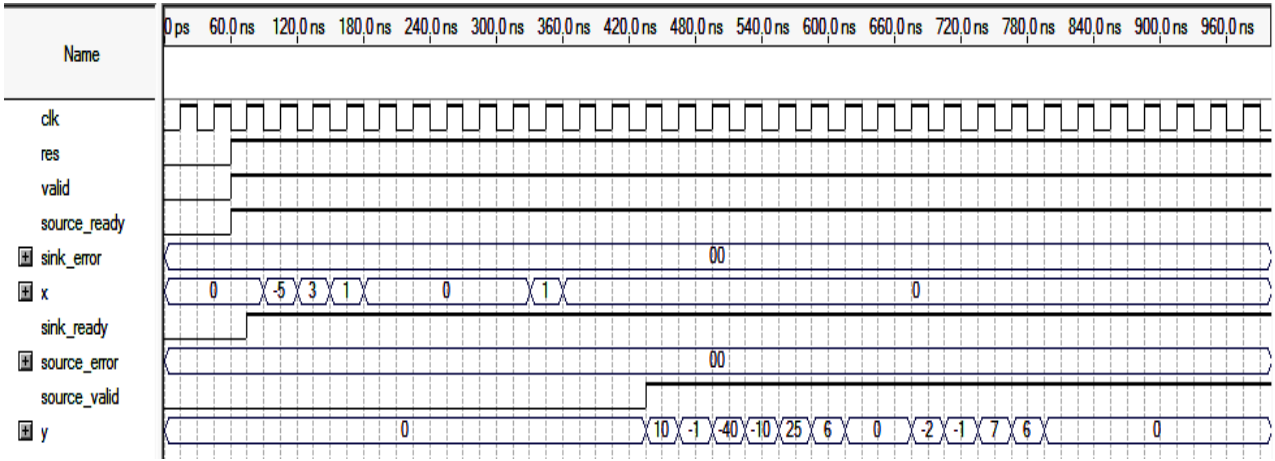


Рис. 4.53. Временные диаграммы работы КИХ-фильтра с использованием параллельной распределенной арифметики на мегаядре FIR Compiler

Входные данные на линии задержки представлены с 8-битной точностью параллельного кода. Для фильтра с симметричными коэффициентами требуются на выходах линии задержки 4 параллельных сумматоров, которые своими выходами непосредственно адресуют 4-входовые LUT. Для того, чтобы переполнения гарантированно не произошло, необходимы 9-разрядные сумматоры, что обеспечивается расширением знакового разряда на входах. Это приводит к увеличению числа LUT с 8 до 9. В случае фильтра с несимметричными коэффициентами восемь отводов линии задержки уже адресуют 16 таких таблиц (число адресных линий равно числу элементов в векторе размерностью K, т. е. вместо использования 8 LUT с восемью входами можно использовать 16 LUT с четырьмя входами).

Частичные произведения, представляющие комбинацию сумм 8-разрядных коэффициентов, хранящиеся в LUT, представлены с 10-битной точностью с запасом в два разряда. Поэтому в случае фильтра с несимметричными коэффициентами для суммирований значений с выходов LUT используются восемь 10-разрядных сумматоров. На входах последующих 12, 14 и 19-разрядных сумматоров требуется коррекция разрядности. Для фильтра с симметричными коэффициентами необходимы 12, 14, 18 и 19-разрядные сумматоры с соответствующей коррекцией на входах, а для получения 19-битной точности дополнительно требуется конвейер из трех регистров для суммирования значений выхода самой младшей LUT.

Для ускорения процесса разработки целесообразно воспользоваться мегаядрами. На рис. 4.51 приведена тестовая схема КИХ-фильтра с использованием мегаядра FIR Compiler САПР Quartus II компании Altera на последовательной и параллельной распределенной арифметике.

Предположим, что коэффициенты фильтра целочисленные со знаком известны и равны $C_0 = -2$, $C_1 = -1$,

$C_2 = 7$ и $C_3 = 6$. На вход КИХ-фильтра поступают входные отсчеты -5, 3, 1 и 0. Правильные значения на выходе фильтра: 10, -1, -40, -10, 26, 6 и т. д., т. е. согласно формуле $y = C_0x_0 + C_1x_1 + C_2x_2 + C_3x_3$.

На рис. 4.52 и рис. 4.53 показаны временные диаграммы работы КИХ-фильтра с использованием последовательной и параллельной распределенной арифметики. Анализ задействованных ресурсов ПЛИС серии Stratix III при реализации КИХ-фильтров на четыре отвода с использованием мегаядра FIR Compiler показан в табл. 3.2.

Анализ табл. 4.3 показывает, что при числе отводов равном четырем, существенной разницы между последовательной и параллельной арифметиками нет, т. к. для фильтра на последовательной арифметике требуется еще и управляющий автомат, который по числу задействованных триггеров может перекрыть число используемых АЛМ для выполнения комбинационных функций.

В структуре КИХ-фильтра на параллельной распределенной арифметике используется параллельный векторный перемножитель.

Несимметричность коэффициентов КИХ-фильтра на параллельной распределенной арифметике ведет к увеличению числа LUT.

Основным достоинством КИХ-фильтров на параллельной распределенной арифметике является повышенное быстродействие при возрастании числа задействованных ресурсов. Значительно снизить число используемых LUT позволяет последовательная распределенная арифметика.

Таблица 4.3

Анализ задействованных ресурсов ПЛИС серии Stratix III при реализации КИХ-фильтров на четыре отвода с использованием мегаядра FIR Compiler

Ресурсы ПЛИС серии Stratix III	Последовательная распределенная арифметика	Параллельная распределенная арифметика
1	2	3
Кол-во АЛМ для выполнения комбинационных функций	74	87
Кол-во АЛМ с памятью	4	4
АЛМ	79	79
Кол-во выделенных регистров	136	134
Аппаратные перемножители		
Кол-во АЛМ для выполнения комбинационных функций без использования регистров	13	17
Кол-во АЛМ под регистерные ресурсы	71	60
Кол-во АЛМ под комбинационные и регистерные ресурсы	65	74
Рабочая частота в наихудшем случае, МГц	400	400

4.5. Пример реализации КИХ-фильтра на параллельной распределенной арифметике в САПР Quartus II

Уравнение КИХ-фильтра со структурой 4 отвода 4 бита (прямая реализация) представляется как арифметическая сумма произведений: $P_{out} = C_1 d_1 + C_2 d_2 + C_3 d_3 + C_4 d_4$. В случае параллельной распределенной арифметики уравнение КИХ-фильтра на четыре отвода записывается в виде

$$P_{out} = 2^0 * P_0 + 2^1 * P_1 + 2^2 * P_2 - 2^3 * P_3. \quad (4.18)$$

Частичные произведения P_0, P_1, P_2 и P_3:

$$P_0 = C_1 d_1(0) + C_2 d_2(0) + C_3 d_3(0) + C_4 d_4(0) = \sum_{n=1}^{4} C_n d_n(0); \quad (4.19)$$

$$P_1 = C_1 d_1(1) + C_2 d_2(1) + C_3 d_3(1) + C_4 d_4(1) = \sum_{n=1}^{4} C_n d_n(1); \quad (4.20)$$

$$P_2 = C_1 d_1(2) + C_2 d_2(2) + C_3 d_3(2) + C_4 d_4(2) = \sum_{n=1}^{4} C_n d_n(2); \quad (4.21)$$

$$P_3 = C_1 d_1(3) + C_2 d_2(3) + C_3 d_3(3) + C_4 d_4(3) = \sum_{n=1}^{4} C_n d_n(3). \quad (4.22)$$

В случае параллельной распределённой арифметики для КИХ-фильтра на 4 отвода необходимо иметь 4 идентичных массива памяти, параллельно адресуемых всеми битами всех входных переменных и свертывающееся иерархическое дерево многоразрядных сумматоров, осуществляющих соответствующее суммирование частичных произведений P_0, P_1, P_2 и P_3. В данном случае результат формируется за один такт и тем самым достигается наибольшее быстродействие структуры.

На рис. 4.54 показана линия задержки КИХ-фильтра, а на рис. 4.55 – принцип подключения выходов линии задержки КИХ-фильтра на 4 отвода к 4-входовым LUT. Разрядность

входной шины данных $N = 4$. Входные данные на линии задержки представлены с 4-битной точностью параллельного кода. 4-входовая LUT обеспечивает 16 частичных произведений (на примере P_0, представляющих собой комбинации сумм коэффициентов фильтра с 8-битной точностью).

Заполнение 4-входовой LUT показано в табл. 4.4, а описание на языке VHDL демонстрирует пример 3. На рис. 4.56 показана структура КИХ-фильтра на 4 отвода 4 бита на распределенной параллельной арифметике. Фильтр состоит из четырех однотипных LUT, формирующих частичные произведения P_0, P_1, P_2 и P_3 согласно формулам 4.19–4.22. Для суммирования значений с выходов LUT в соответствии с их весом (формула (4.18)) используются два 12-разрядных и один 14-разрядный сумматоры с соответствующей коррекцией разрядности на входах для того, чтобы гарантированно предотвратить переполнение.

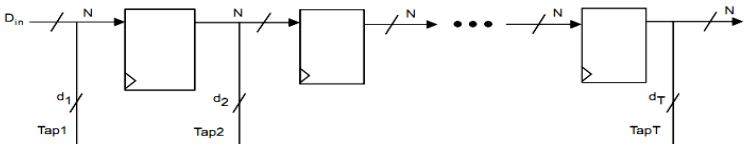


Рис. 4.54. Линия задержки КИХ-фильтра на регистрах

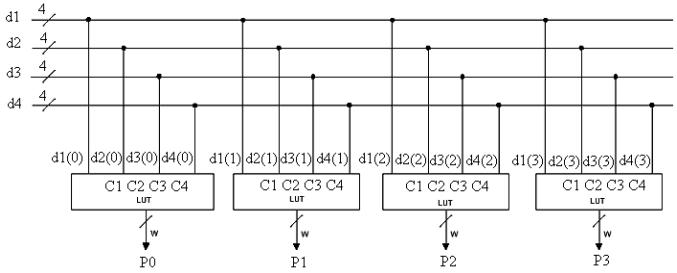


Рис. 4.55. Принцип подключения выходов линии задержки КИХ-фильтра на 4 отвода к 4-входовым LUT

Таблица 4.4

Вариант заполнения 4-входовой LUT на примере частичного произведения P_0

d4(0),d3(0),d2(0),d1(0)	Выход LUT-таблицы, P0
0000	0
0001	C1
0010	C2
0011	C2+C1
0100	C3
….	….
1111	C4+C3+C2+C1

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;
Entity PartialProd is
Port (D : in std_logic_vector(3 downto 0);
      P : out std_logic_vector(9 downto 0));
-- 4 * 8 bit coeff => 10 bit product
End PartialProd;
Architecture Behave of PartialProd is
constant c1 : std_logic_vector := "11111110"; -- coefficient for tap 1
-- -2D
constant c2 : std_logic_vector := "11111111"; -- coefficient for tap 2
-- -1D
constant c3 : std_logic_vector := "00000111"; -- coefficient for tap 3
-- 7D
constant c4 : std_logic_vector := "00000110"; -- coefficient for tap 4
-- 6D
-- Compute all the partial products and store them as constants.
constant v0 : std_logic_vector := sxt("0", 10);
constant v1 : std_logic_vector := sxt(c1, 10);
constant v2 : std_logic_vector := sxt(c2, 10);
constant v3 : std_logic_vector := v1 + v2;
constant v4 : std_logic_vector := sxt(c3, 10);
```

```
constant v5 : std_logic_vector := v4 + v1;
constant v6 : std_logic_vector := v4 + v2;
constant v7 : std_logic_vector := v4 + v3;
constant v8 : std_logic_vector := sxt(c4, 10);
constant v9 : std_logic_vector := v8 + v1;
constant v10 : std_logic_vector := v8 + v2;
constant v11 : std_logic_vector := v8 + v3;
constant v12 : std_logic_vector := v8 + v4;
constant v13 : std_logic_vector := v8 + v5;
constant v14 : std_logic_vector := v8 + v6;
constant v15 : std_logic_vector := v8 + v7;
Begin
prodeval: process (D)
begin
case(d) is
when "0000" => P <= v0;
when "0001" => P <= v1;
when "0010" => P <= v2;
when "0011" => P <= v3;
when "0100" => P <= v4;
when "0101" => P <= v5;
when "0110" => P <= v6;
when "0111" => P <= v7;
when "1000" => P <= v8;
when "1001" => P <= v9;
when "1010" => P <= v10;
when "1011" => P <= v11;
when "1100" => P <= v12;
when "1101" => P <= v13;
when "1110" => P <= v14;
when "1111" => P <= v15;
end case;
end process;
end behave;
```

Пример 3. Описание заполнения LUT на языке VHDL для КИХ-фильтра на 4 отвода

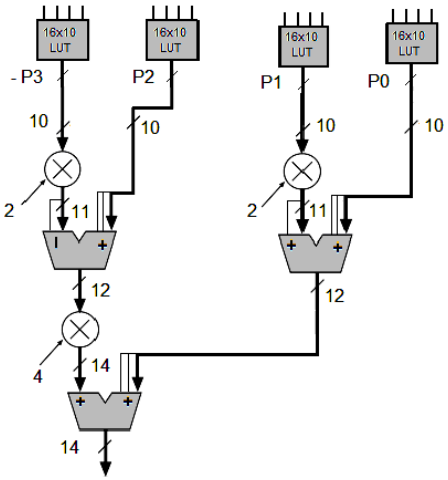


Рис. 4.56. Структура КИХ-фильтра на 4 отвода 4 бита на распределенной параллельной арифметике с указанием операции расширения знака

На рис. 4.57 показана функциональная модель КИХ-фильтра на 4 отвода 4 бита на распределенной параллельной арифметике в САПР ПЛИС Quartus II с использованием линии задержки на регистрах (рис. 4.57а) и линии задержки на базе встроенных блоков ОЗУ (рис. 4.58б).

Мегафункция ALTSHIFT_TAPS, используемая в качестве линии задержки, представляет собой двухпортовое ОЗУ. Свертывающее иерархическое дерево многоразрядных сумматоров выполнено на мегафункциях LPM_ADD_SUB. Для знакового разряда (P_3) необходимо сформировать дополнение до двух путем обращения P_3 с последующим прибавлением 1 к младшему разряду.

Обращение логически эквивалентно инверсии каждого бита в числе. Вентили Исключающее ИЛИ можно применить для избирательной инверсии в зависимости от значения управляющего сигнала. Прибавление 1 можно организовать

подключением входа переноса C_{in} одного из 12-разрядных сумматоров к шине питания.

Прохождение единичного импульса по структуре КИХ-фильтра в случае использования линии задержки на регистрах показано на рис. 4.59. На выходе фильтра видим коэффициенты фильтра $C_0 = -2$, $C_1 = -1$, $C_2 = 7$ и $C_3 = 6$.

На рис. 4.60 показаны временные диаграммы работы фильтра для случая, когда на вход КИХ-фильтра поступают входные отсчеты -5, 3, 1 и 0. Правильные значения на выходе фильтра: 10, -1, -40, -10, 26, 6 и т.д. На рис. 4.61 и 4.62 показаны временные диаграммы в случае использования линии задержки на базе встроенных блоков ОЗУ.

В тестировании принимают участие следующие структуры фильтров: классическая параллельная с использованием аппаратных ЦОС-боков (собран на мегафункции ALTMULT_ADD); систолическая структура; поведенческое описание на языке VHDL с использованием оператора цикла без привязки к какой-либо структуре, с использованием программных умножителей (мегафункция ALTMEMMULT) и фильтр на параллельной распределенной арифметике, реализованный с помощью мегаядра MegaCore FIR Compiler.

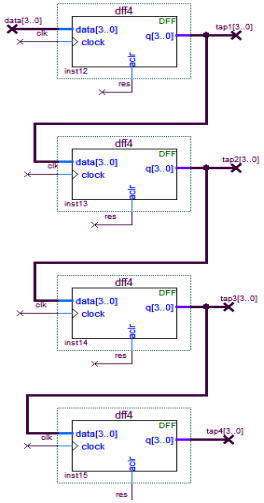
Рассмотренные основные структурные схемы параллельных КИХ-фильтров позволяют сделать вывод, что использование КИХ-фильтров на параллельной распределенной арифметике позволяет для ПЛИС серии Cyclone III получить рекордное быстродействие за счет использования «безумножительных» схем умножения (табл. 3.4), неснижаемое при увеличении числа отводов фильтра и точности представления входных данных. Это особенно актуально для проектов, использующих низкопроизводительные (бюджетные) серии ПЛИС. А также в том случае, если пользователь откажется от применения в проекте мегафункций умножителей, опционально

использующих либо аппаратные умножители, встроенные в ЦОС-блоки (ALTMULT_ADD, ALTMULT_ACCUM), либо программные (ALTMEMMULT).

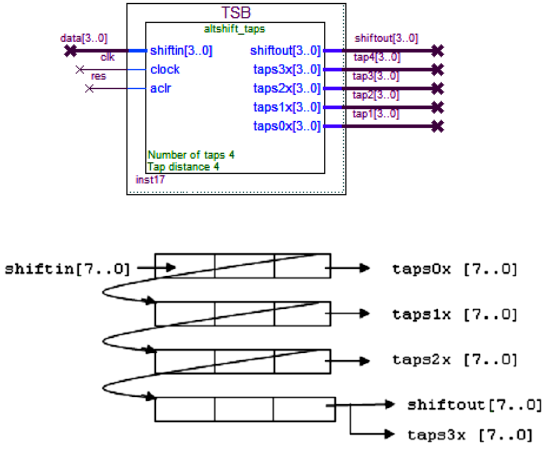
Использование кода языка VHDL и мегаядра FIR Compiler (структура фильтра задается пользователем опционально), в отличие от фильтров, структура которых разрабатывается «вручную», приводит к пониженному быстродействию, однако при этом необходимо учитывать, что рассматриваемые примеры сильно упрощены. Для точной оценки производительности фильтров необходимо пользоваться сравнительными таблицами из официальных документов производителей ПЛИС.

Для КИХ-фильтров с большим числом отводов и точностью представления коэффициентов характерно возрастание числа задействованных ресурсов ПЛИС (табл. 3.5), поэтому необходимо по возможности использовать симметричные фильтры.

Использование линии задержки на блочной памяти ПЛИС также позволяет сократить число используемых логических элементов. Последовательно распределенная арифметика снижает объем задействованных ресурсов ПЛИС, но ухудшает быстродействие и производительность фильтров.

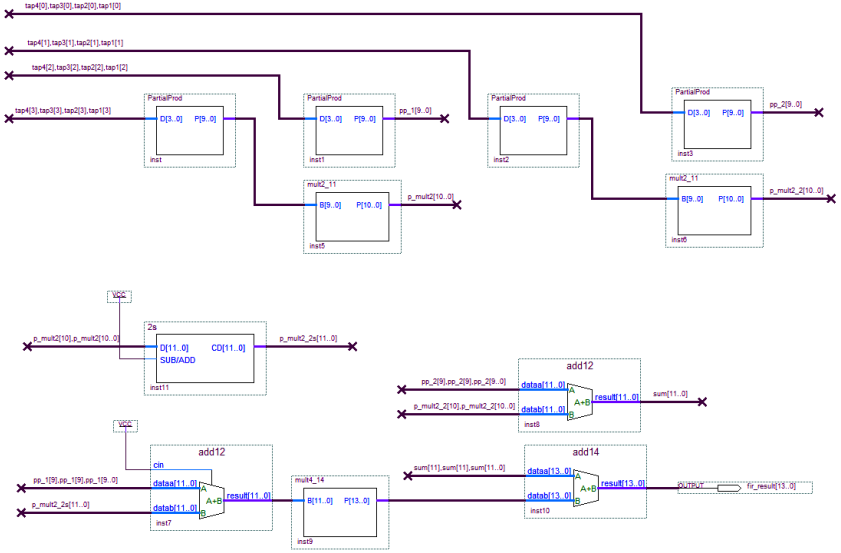


а)



б)

Рис. 4.57. КИХ-фильтр на 4 отвода 4 бита на распределенной параллельной арифметике в САПР ПЛИС Quartus II: а) линия задержки на регистрах; б) линия задержки на базе встроенных блоков ОЗУ; в) частичные произведения на базе LUT и свертывающееся иерархическое дерево многоразрядных сумматоров



в)

Рис. 4.58. КИХ-фильтр на 4 отвода 4 бита на распределенной параллельной арифметике в САПР ПЛИС Quartus II: а) линия задержки на регистрах; б) линия задержки на базе встроенных блоков ОЗУ; в) частичные произведения на базе LUT и свертывающееся иерархическое дерево многоразрядных сумматоров (продолжение)

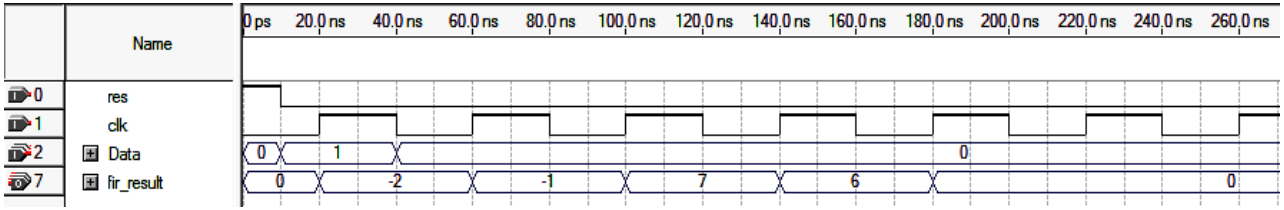


Рис. 4.59. Прохождение единичного импульса по структуре КИХ-фильтра в случае использования линии задержки на регистрах

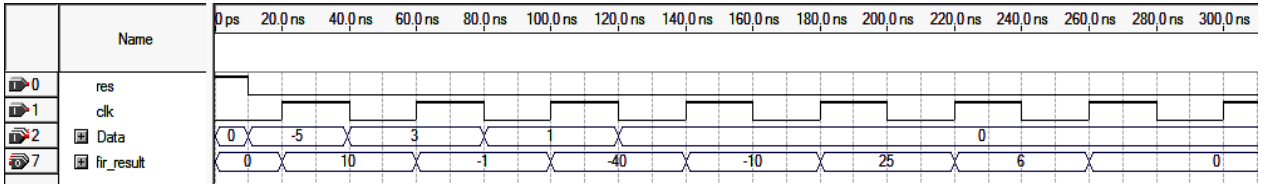


Рис. 4.60. Временные диаграммы работы фильтра в случае использования линии задержки на регистрах. На вход КИХ-фильтра поступают входные отсчеты -5, 3, 1 и 0. Правильные значения на выходе фильтра: 10, -1, -40, -10, 26, 6 и т.д.

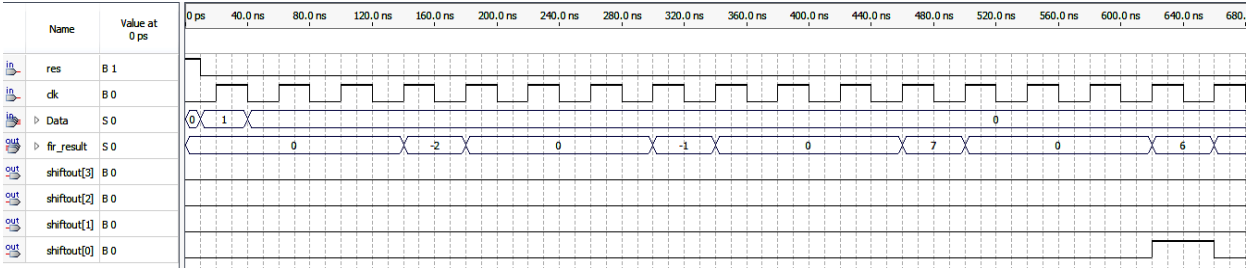


Рис. 4.61. Прохождение единичного импульса по структуре КИХ-фильтра в случае использования линии задержки на базе встроенных блоков ОЗУ

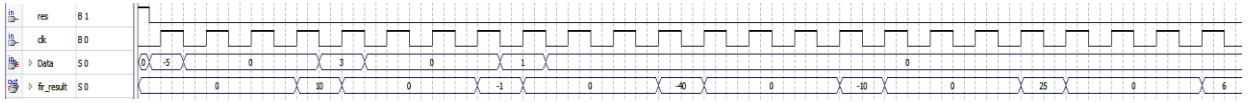


Рис. 4.62. Временные диаграммы работы фильтра в случае использования линии задержки на базе встроенных блоков ОЗУ

4.6. Пример реализации КИХ-фильтра на параллельной распределенной арифметике в Altera DSP Builder

На рис. 4.63 показаны разрабатываемая имитационная модель КИХ–фильтра, построенная по структурной схеме, приведенной на рис. 4.56, и КИХ-фильтр на последовательной распределенной арифметике (рис. 4.18).

Модели работают на разных скоростях. Зеленым цветом подсвечена модель КИХ-фильтра на параллельной, а красным модель на последовательной распределенной арифметики. Для краткости назовем эти модели PDA и SDA соответственно. Для модели PDA временной шаг симуляции в Matlab/Simulink задается 1 с, а для модели SDA – 0.25 с.

Шаг симуляции для модели SDA выбирается исходя из следующих соображений: профильтрованные значения на выходе фильтра появляются через четыре такта синхроимпульса. За интервал дискретизации входного сигнала процесс вычисления КИХ-фильтрами на PDA и SDA должен завершиться. Интервал дискретизации единичного сигнала (дельта-функция) и ЛЧМ-сигнала (Sample time) – 1 с, поэтому они так же подсвечены зеленым цветом.

Для корректной работы имитационной модели КИХ-фильтра на SDA необходимо использовать ФАПЧ для генерации дочернего синхросигнала. Для этого в модель добавлен блок Clock AlteraBlockset (Clock_Derived). В поле Base Clock Multiplicand Denomerator функционального блока Clock AlteraBlockset установим значение 4. С помощью функционального блока Tsamp устанавливается частота в 4 раза больше, чем у модели PDA. Имя дочернего синхросигнала Clock_Derived.

Дополнительно для проверки работы КИХ-фильтров PDA и SDA, построенных с помощью пакета расширения Altera DSP Builder, подключен функциональный блок Digital Filter (прямая форма) из библиотеки Simulik.

Входной сигнал умножается на масштабный множитель 2^3-1, профильтрованный сигнал также делится на это значение. Масштабный множитель подбирался экспериментально, исходя из 4-разрядной точности представления входных отсчетов сигнала. Модель состоит из линии задержки на функциональных блоках Delay, выполняющих функцию регистров, блоков Bus Splitter, выполняющих функцию разделения 4-разрядных отводов фильтра на битовые значения, генераторы булевых функций (LUT), три устройства сдвига данных Barrel Shifter вправо на 1 (2 шт) и на 2 разряда (что равносильно умножению на 2 и 4 соответственно), а также умножитель для учета знака числа.

На рис. 4.64 показаны результаты имитационного моделирования фильтрации ЛЧМ-сигнала КИХ-фильтрами с различной структурой: а) исходный сигнал; б) КИХ на PDA; в) DigitalFilter; г) КИХ на SDA. Совпадение результатов фильтрации подтверждает правильность работы имитационных моделей.

В дальнейшем от умножителя на константу –1 необходимо отказаться, т.к. при синтезе VHDL-кода будет использоваться умножитель ЦОС-блока. На рис. 4.65 показаны изменения в модели, которые необходимо сделать, чтобы полностью отказаться от использования умножителя для учета знака частичного произведения путем его замены на сумматор/вычитатель. Такая замена не влияет на логику работы устройства.

Битовые сигналы с выходов разделителей Bus Splitter подключены по следующему правилу: все младшие разряды всех 4 отводов подключены к 4-LUT (с названием Bit Level Sum of Product), формирующую частичное произведение P_0, все более старшие разряды всех 4 отводов подключены к 4-LUT, формирующую частичное произведение P_1, и так далее, пока не будет сформировано частичное произведение P_3.

Рассмотрим блок BaseClock. Для функционального моделирования в Altera-ModelSim задается синхросигнал с периодом 20 нс (поле Real-World Clock Period) и временной шаг симуляции в Matlab/Simulink 1 с (поле Simulink Sample Time). На рис. 4.66 представлено формирование содержимого LUT. Коэффициенты представлены с 10-разрядной точностью, а суммы коэффициентов с 12-разрядной точностью. Для такого фильтра требуется 4 идентичных таблицы.

Линия задержки проектируемого фильтра построена на регистрах из функциональных блоков Delay. Число уровней конвейеризации блока – один. На рис. 4.67 показан функциональный блок Bus Splitter, выполняющий разделение четырехразрядной шины на биты. Таких блоков в модели 4.

На рис. 4.68 показано 12-разрядное устройство сдвига данных вправо на один разряд, что равносильно умножению на два. Направление сдвига определяется портом derection. Дистанция сдвига задается портом distance (d=1). Нас рис. 4.69 представлено 14-разрядное устройство сдвига вправо на 2 разряда, что равносильно умножению на четыре. Суммирование результатов сдвига выполняет дерево сумматоров, которое показано на рис. 4.56.

Из имитационной модели КИХ-фильтра на PDA (рис. 4.63) извлечем с помощью функционального блока SignalCompiler VHDL-код и разработаем проект в САПР Quartus II ver.13.0 (рис. 4.70).

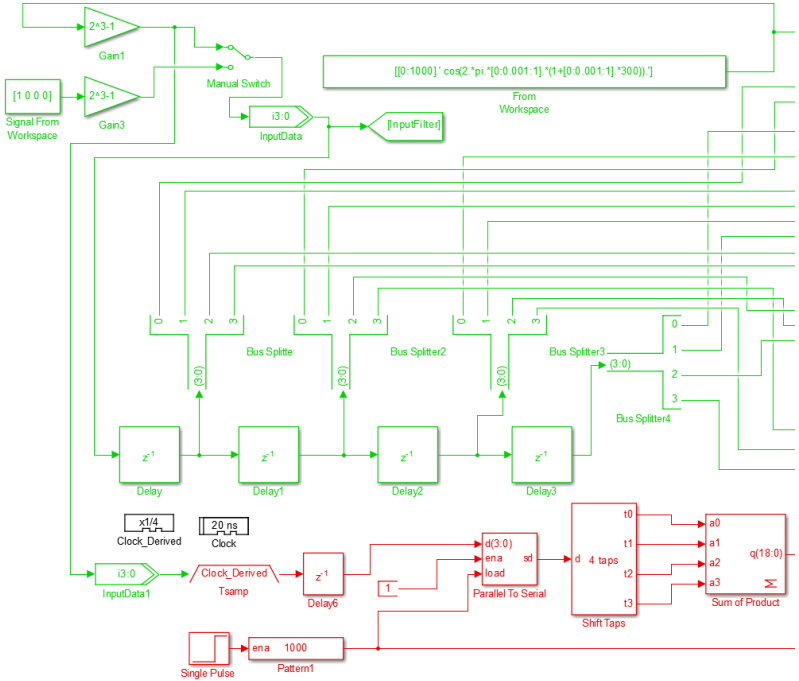
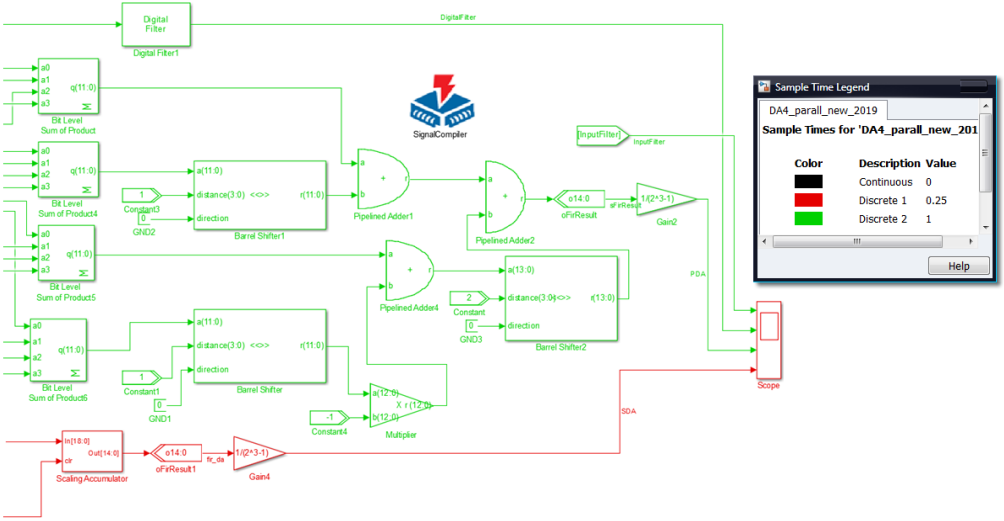


Рис. 4.63. (начало)



(продолжение)

Рис. 4.63. Имитационные модели КИХ-фильтра на параллельной (PDA) и последовательной (SDA) распределенной арифметике для фильтрации ЛЧМ-сигнала

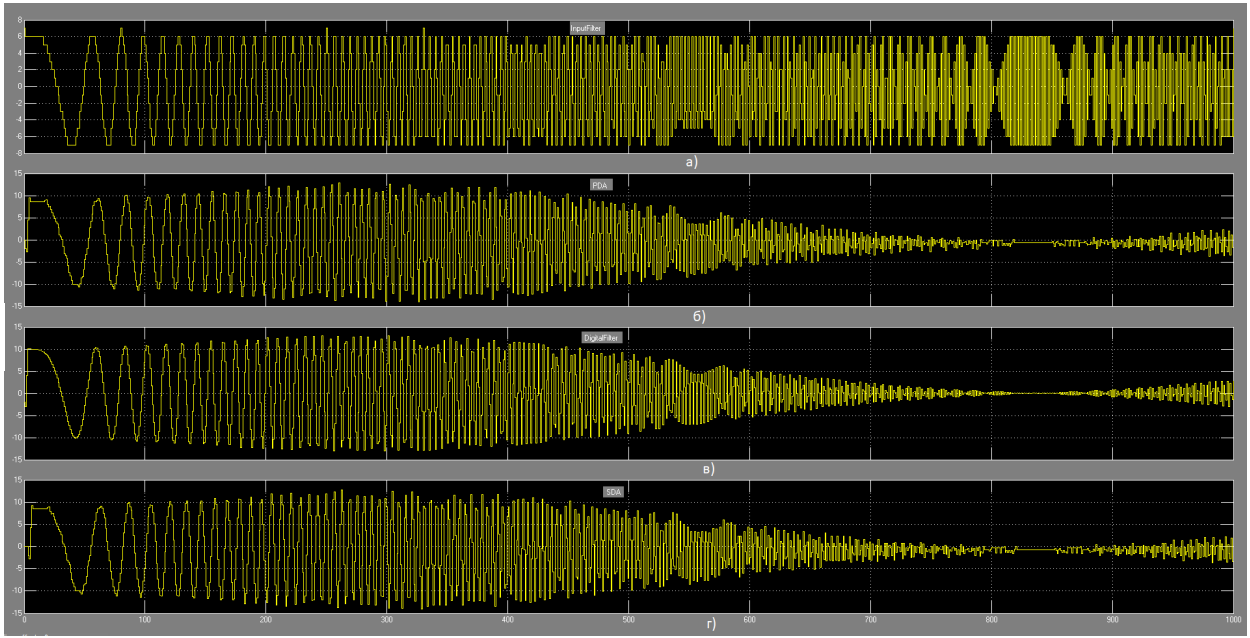


Рис. 4.64. Результаты имитационного моделирования фильтрации сигнала:
а) исходный ЛЧМ-сигнал; б) PDA; в) DigitalFilter; г) SDA

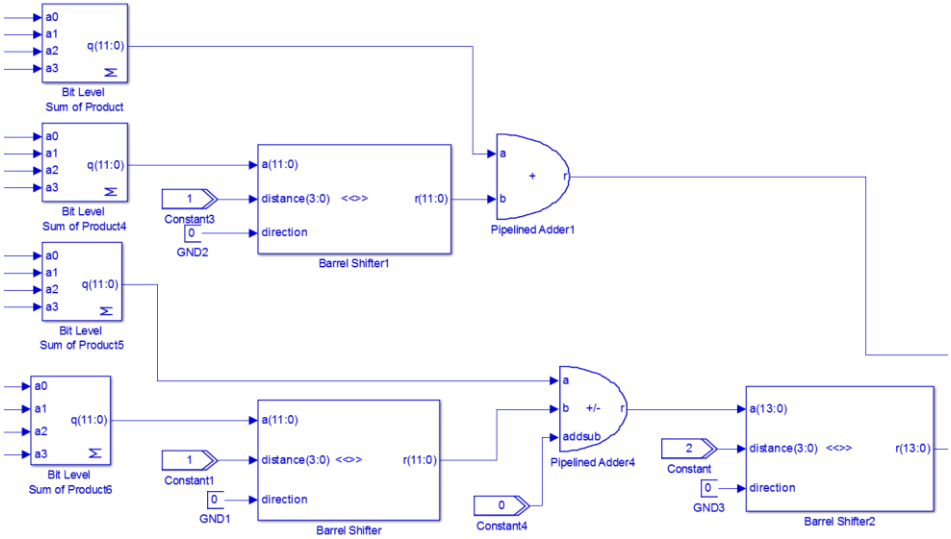


Рис. 4.65. Изменения в модели, которые необходимо сделать, чтобы полностью отказаться от использования умножителя для учета знака частичного произведения P_3 путем замены на сумматор/вычитатель

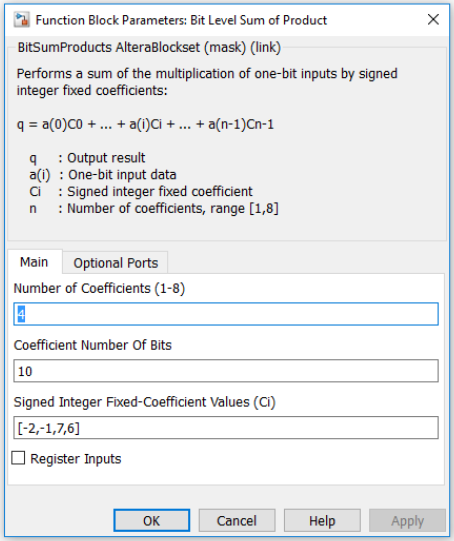


Рис. 4.66. Формирование содержимого LUT

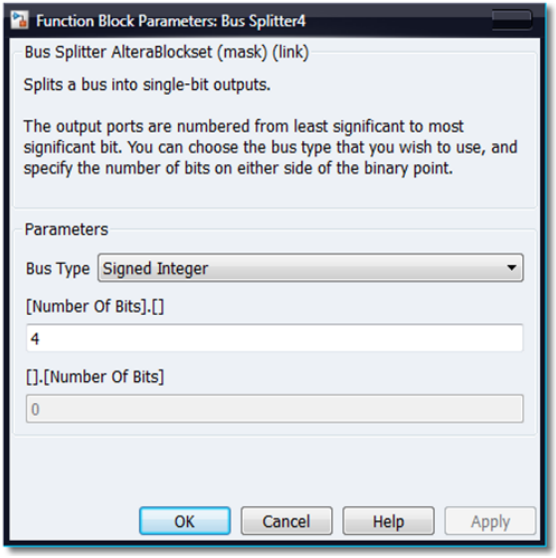


Рис. 4.67. Шинный разделитель

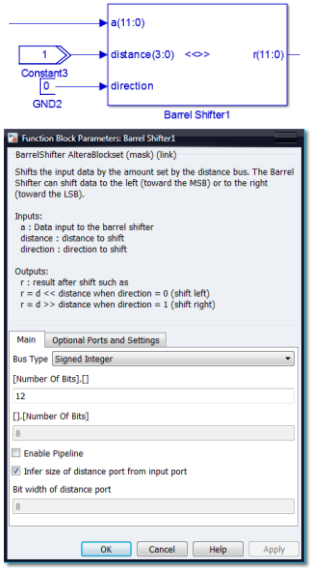


Рис. 4.68. Функциональный блок BarrelShifter, настроенный на сдвиг вправо на 1 разряд 12-разрядного числа, дистанция d=1

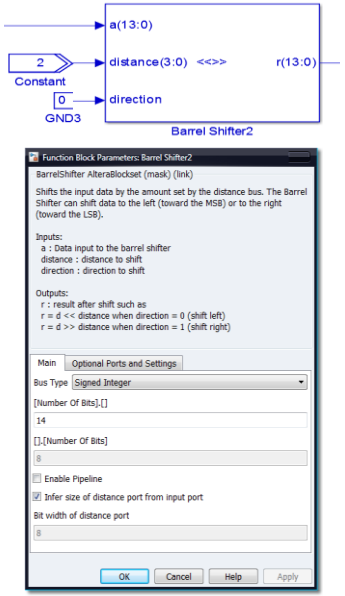


Рис. 4.69. Функциональный блок BarrelShifter, настроенный на сдвиг вправо на 2 разряда 14-разрядного числа, дистанция d=2

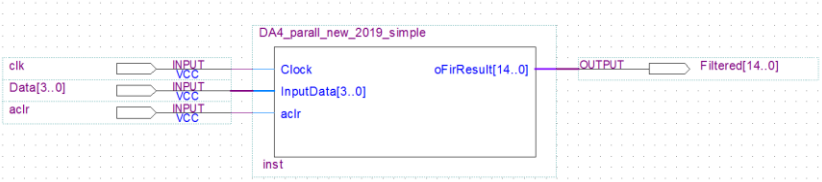


Рис. 4.70. Проект КИХ-фильтра, реализованный в САПР Quartus II по VHDL-коду, извлеченному в автоматическом режиме из имитационной модели

На рис. 4.71а показана оценка задействованных ресурсов на этапе анализа и синтеза в САПР при реализации проекта в базис ПЛИС серии Cyclone V типа 5CGXFC7C7F23C8 (содержит 156 ЦОС-блоков). Для реализации проекта КИХ-фильтра на PDA требуются 33 АЛМ из них: адаптивных LUT в режиме 4-LUT – 16 шт.; в режиме меньше или равно 3 LUT – 43 шт.; выделенных регистров АЛМ – 16 и ЦОС-блоков – 0 шт. Архитектурное планирование кристалла ПЛИС серии Cyclone V типа 5CGXFC7C7F23C8 показывает, что при реализации проекта задействуются логические ресурсы (АЛМ) и блочная память типа M10K (рис. 4.72).

Разработаем проект в САПР Quartus Prime v.15.0 (рис. 4.73) с использованием мегаядра FIR II (altera_fir_compiler_ii). Данное ядро позволяет быстро разрабатывать КИХ-фильтры с прямой формой с использованием аппаратных умножителей в ЦОС-блоках. Коэффициенты фильтра –2, –1, 7 и 6 загрузим из файла. Реализуем проект в такой же ПЛИС серии Cyclone V типа 5CGXFC7C7F23C8. На рис. 4.74 показаны настройки мегаядра FIR II САПР Quartus Prime v.15.0 (АЧХ фильтра), закладка “Коэффициенты”. На рис. 4.75 показано архитектурное планирование кристалла ПЛИС серии Cyclone V типа 5CGXFC7C7F23C8. Задействуются АЛМ и два ЦОС-блока (четыре умножителя).

Analysis & Synthesis Resource Usage Summary

	Resource	Usage
1	Estimate of Logic utilization (ALMs needed)	33
2		
3	◢ Combinational ALUT usage for logic	59
1	-- 7 input functions	0
2	-- 6 input functions	0
3	-- 5 input functions	0
4	-- 4 input functions	16
5	-- <=3 input functions	43
4		
5	Dedicated logic registers	16
6		
7	I/O pins	21
8	Total DSP Blocks	0
9	Maximum fan-out node	clk~input
10	Maximum fan-out	16
11	Total fan-out	264
12	Average fan-out	2.26

а)

	Resource	Usage
1	Estimate of Logic utilization (ALMs needed)	22
2		
3	⌄ Combinational ALUT usage for logic	13
1	-- 7 input functions	0
2	-- 6 input functions	0
3	-- 5 input functions	0
4	-- 4 input functions	0
5	-- <=3 input functions	13
4		
5	Dedicated logic registers	43
6		
7	I/O pins	22
8		
9	Total DSP Blocks	2
10		
11	Maximum fan-out node	clk~input
12	Maximum fan-out	45
13	Total fan-out	228
14	Average fan-out	2.24

б)

Рис. 4.71. Оценка задействованных ресурсов при реализации КИХ-фильтра на 4 отвода в базис ПЛИС серии Cyclone V типа 5CGXFC7C7F23C8: а) PDA в САПР Quartus II; б) мегаядро FIR II САПР Quartus Prime

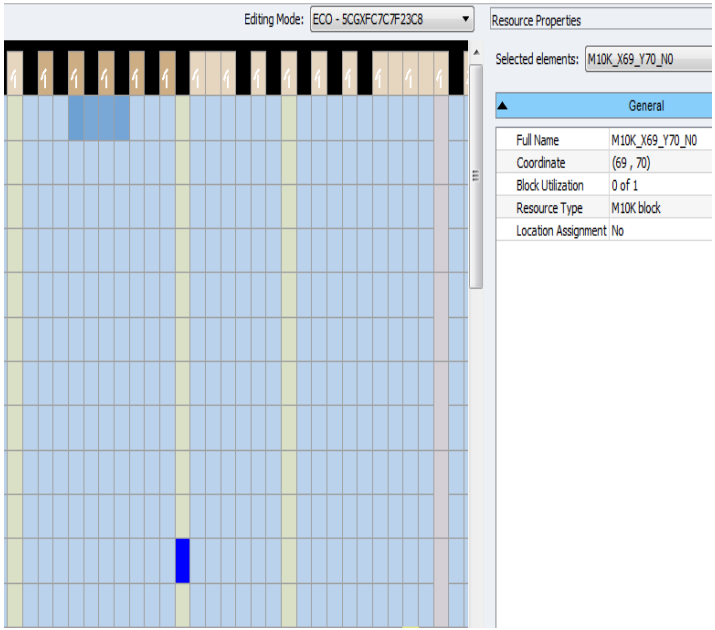


Рис. 4.72. Архитектурное планирование кристалла ПЛИС серии Cyclone V типа 5CGXFC7C7F23C8. Задействуются логические ресурсы и блочная память типа M10K (КИХ-фильтра на PDA)

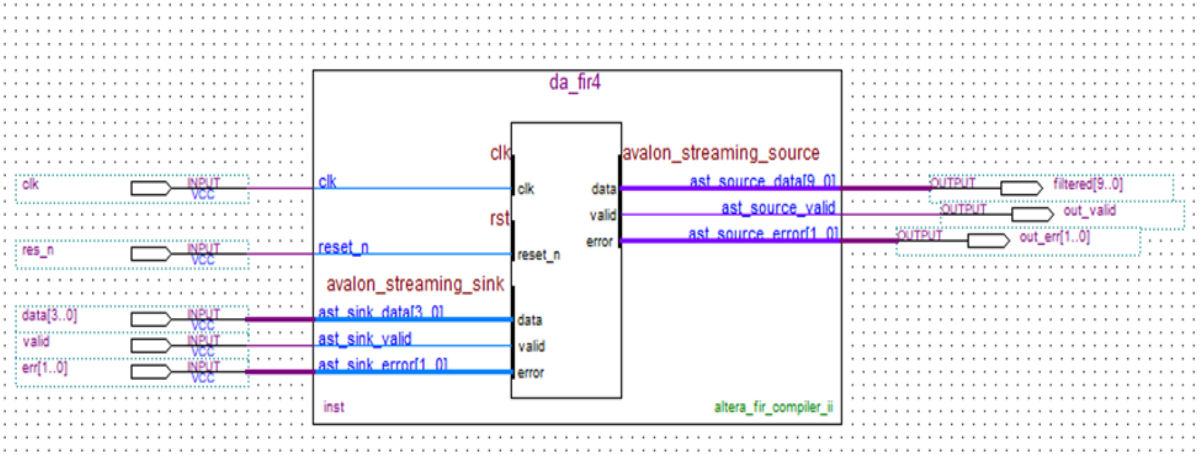


Рис. 4.73. Проект КИХ-фильтра с использованием мегаядра FIR II в САПР Quartus Prime v.15.0

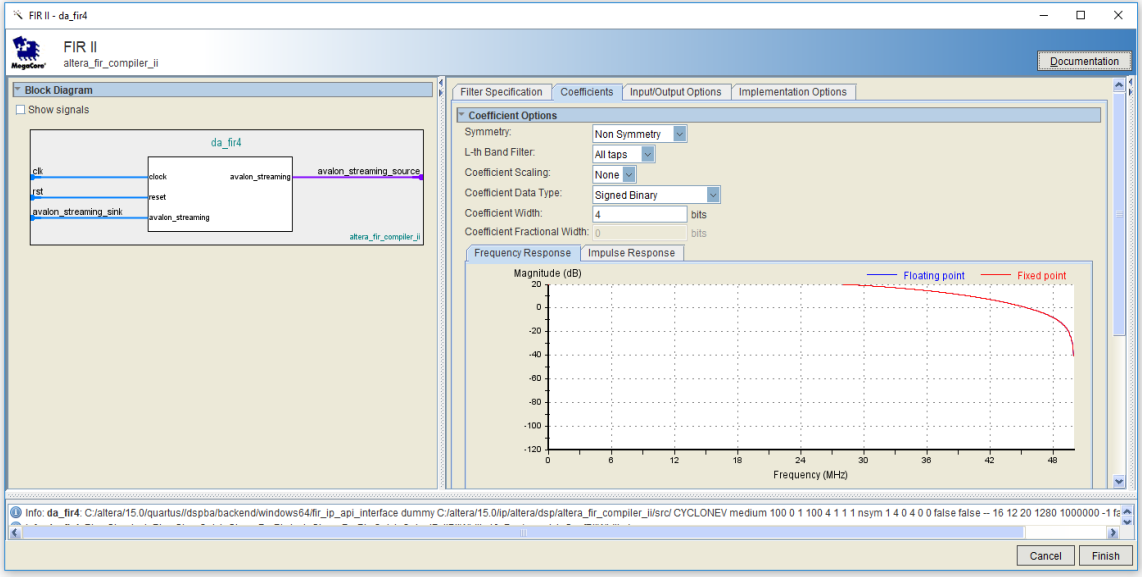


Рис. 4.74. Настройки мегаядра FIR II САПР Quartus Prime v.15.0 (АЧХ-фильтра)

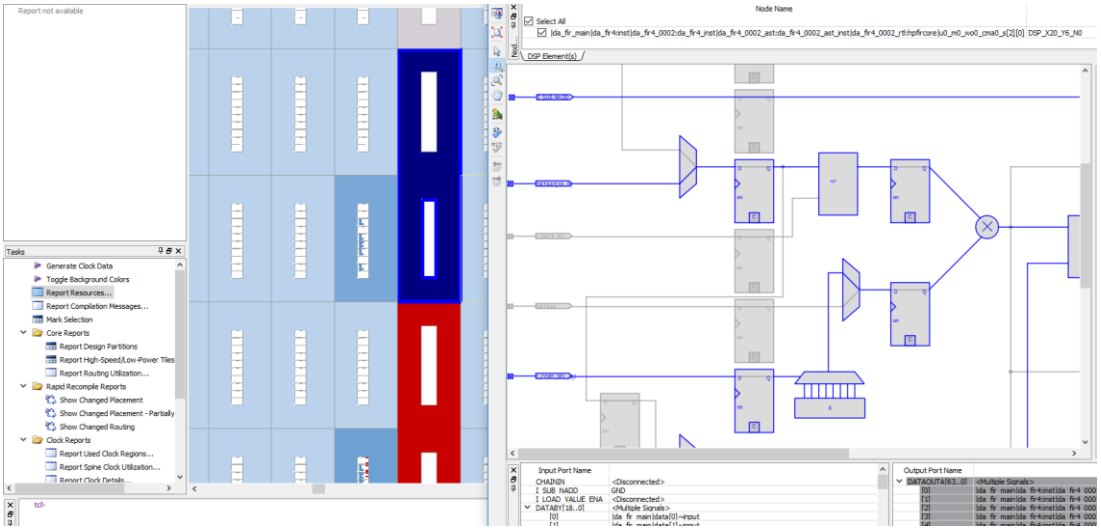


Рис. 4.75. Архитектурное планирование кристалла ПЛИС серии Cyclone V типа 5CGXFC7C7F23C8

Оценка задействованных ресурсов на этапе анализа и синтеза при реализации проекта в базис ПЛИС серии Cyclone V типа 5CGXFC7C7F23C8 с использованием мегаядра FIR II показывает, что для реализации проекта требуются 22 АЛМ из них: адаптивных LUT для комбинационной логики в режиме меньше или равно 3 LUT – 13 шт.; выделенных регистров АЛМ для последовательностной логики – 43 и ЦОС- блоков – 2 шт.

В табл. 4.5 приведены оценки максимальной рабочей частоты Fmax и рабочей частоты в наихудшем случае функциональных моделей КИХ-фильтра на SDA, PDA и на ЦОС-блоках (мегаядро FIR II) при использовании медленной временной модели при критическом напряжении питания ядра ПЛИС 1.1 В и крайней температуре 85 ^{0}C, полученные с помощью TimeQuest. Анализ таблицы показывает, что практически можно получить двукратный выигрыш по быстродействию в случае использования КИХ-фильтра на PDA.

Использование пакета расширения Altera DSP Builder системы визуально-имитационного моделирования Matlab/Simulink позволяет быстро разрабатывать имитационные модели сложных цифровых устройств, таких как КИХ-фильтры на последовательной и параллельной распределенной арифметике с извлечением VHD-кода. Мегаядра FIR Compiler и FIR II САПР Quartus II позволяют без значительных усилий реализовывать функциональные модели КИХ-фильтров в базис ПЛИС.

Наглядное сравнение показало, что быстродействие КИХ-фильтра на 4 отвода на параллельной распределенной арифметике оказалось примерно в два раза выше, чем у КИХ-фильтра на ЦОС-блоках (табл. 4.5).

Таблица 4.5

Оценка быстродействия КИХ-фильтра на 4 отвода, реализованного в базисе ПЛИС серии Cyclone V типа 5CGXFC7C7F23C8, временная модель Slow 1100 мВ и 85 ^{0}C

Оцениваемый параметр	КИХ-фильтр на SDA	КИХ-фильтр на PDA	КИХ-фильтр на ЦОС-блоках (мегаядро FIR II)
Максимальная рабочая частота Fmax, МГц	226.45	407.83	254.39
Рабочая частота в наихудшем случае Restricted Fmax, МГц	180.02	407.83	220.02

4.7. Пример проектирования КИХ-фильтров на распределенной арифметике в базисе ПЛИС с применением генератора параметризированных ядер XLogiCORE IP и функции FIR Compiler v5.0 САПР фирмы Xilinx ISE

В данном разделе предлагается рассмотреть вопрос проектирования КИХ-фильтров на последовательной распределенной арифметике с использованием генератора параметризированных ядер XLogiCORE IP и функции FIR Compiler v5.0 САПР фирмы Xilinx ISE. Выигрыш от использования распределенной арифметики заключается в том, что с ростом числа отводов производительность КИХ-фильтра остается постоянной за счет применения «безумножительных» схем умножения, при этом обеспечивается повышенное быстродействие, экономия по использованию встроенных ЦОС-блоков, а недостатком – повышенный расход логических ресурсов ПЛИС.

Генератор параметризированных ядер XLogiCORE IP FIR Compiler v5.0 предлагает на выбор три структуры фильтра: прямую форму систолического фильтра, в котором операции умножения и сложения выполняются параллельно с конвейеризацией; обратную и на распределенной арифметике. Функция FIR Compiler v5.0 не поддерживает современный протокол AXI4-Stream.

На рис. 4.76а показана структурная схема КИХ-фильтра на 4 отвода с использованием последовательной распределенной арифметики, применение которой, например для последовательного КИХ-фильтра, позволяет отказаться от использования четырех аппаратных умножителей на константу и сократить дерево сумматоров (рис. 4.76б), заменив их единственной таблицей перекодировок. На практике для реализации адресуемых массивов комбинаций весовых коэффициентов фильтра используют таблицы перекодировок (LUT) ПЛИС (рис. 4.76в).

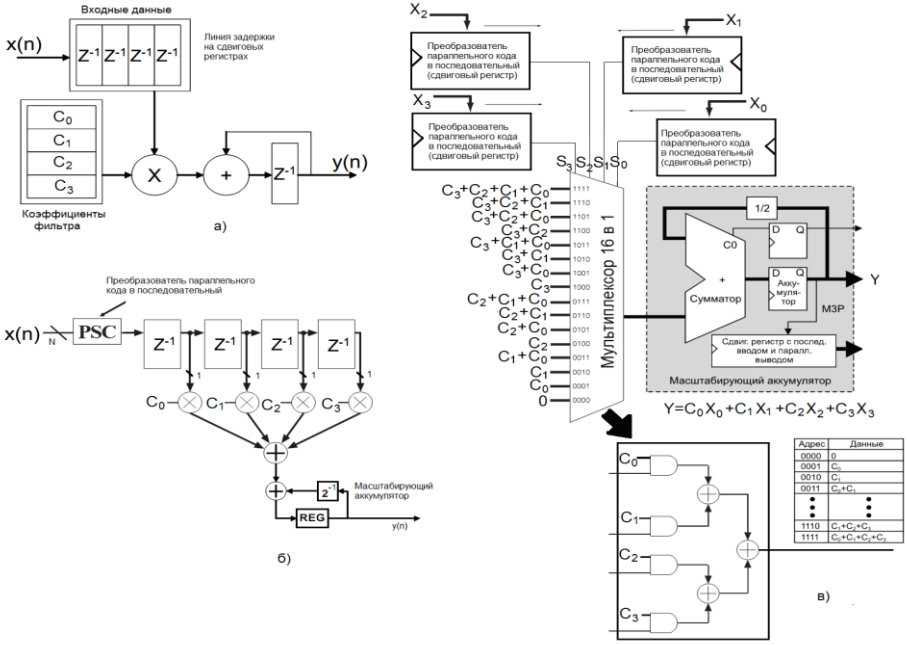


Рис. 4.76. Структуры КИХ-фильтров: а) один MAC-фильтр; б) последовательный КИХ-фильтр на 4 отвода; в) упрощенное представление структуры КИХ-фильтра на 4 отвода с использованием последовательной распределенной арифметики

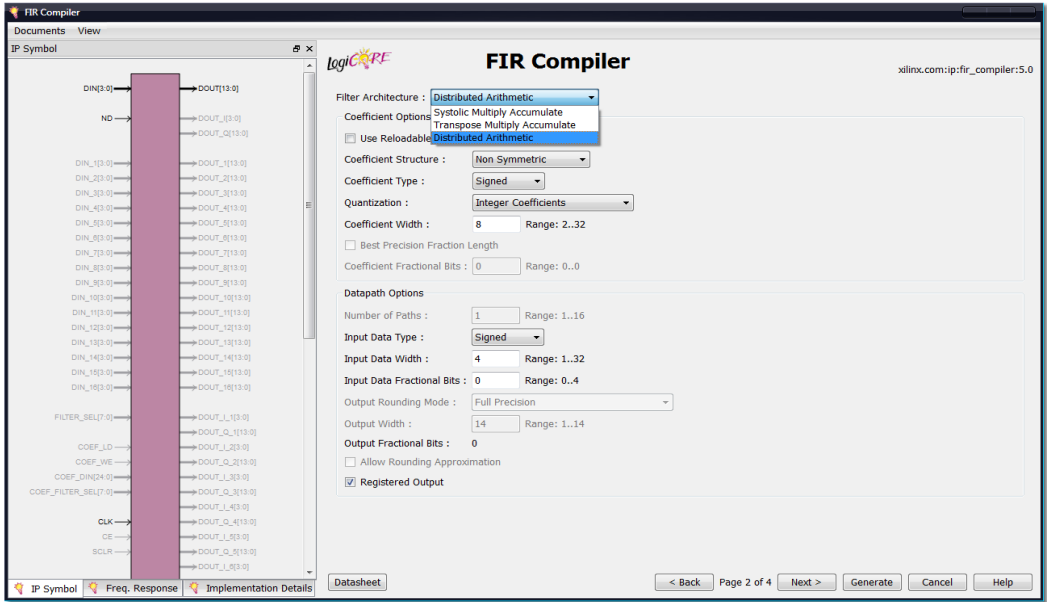


Рис. 4.77. Опции генератора параметризированных ядер XLogiCORE IP FIR Compiler Compiler v5.0. Выбирается структура фильтра на распределенной арифметике. Коэффициенты фильтра несимметричные, со знаком, целые, представляются с 8-битной точностью. Входные значения – целые со знаком, представляются с 4-битной точностью. Выходные значения представляются с 14-битной точностью

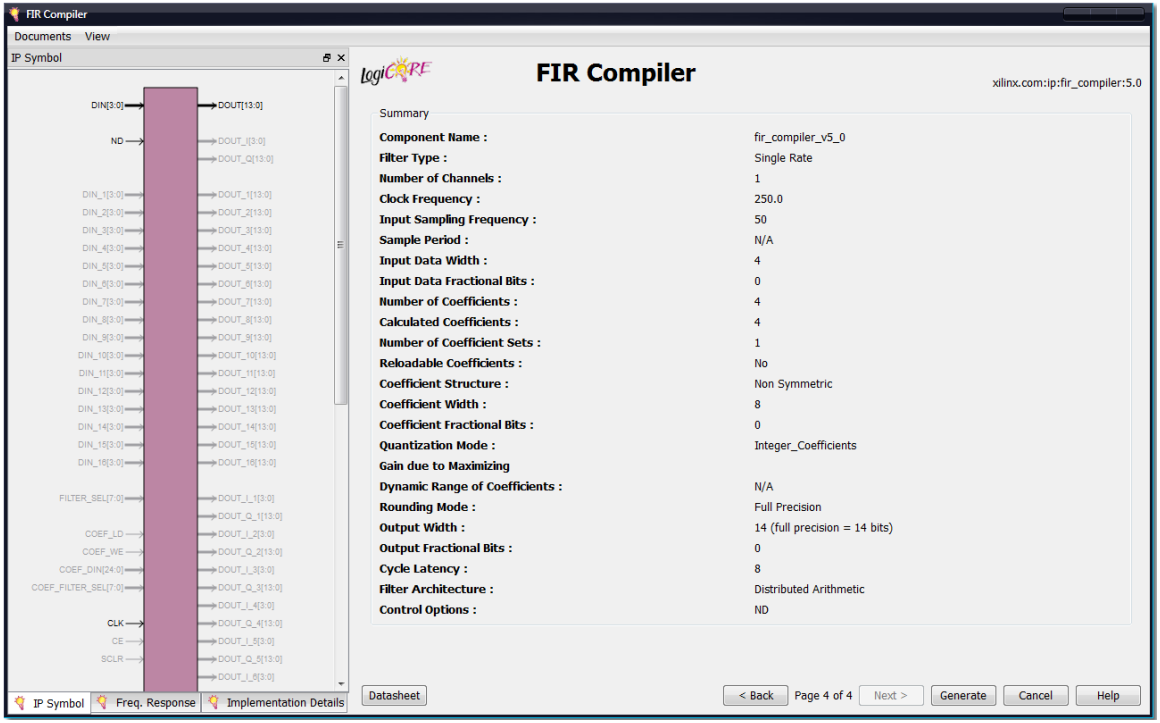


Рис. 4.78. Отчет о характеристиках КИХ-фильтра на 4 отвода

Выберем тип фильтра Single-Rate FIR и формат представления чисел с фиксированной запятой. В случае реализации КИХ-фильтра на распределенной арифметике генератор автоматически определяет тип арифметики: параллельная или последовательная. Степень параллелизма зависит от соотношения частоты взятия входных отчетов (fs) и частоты тактирования системы (fclk). Чем выше fs по отношению к fclk, тем меньше латентность фильтра L (задержка появления профильтрованного сигнала измеряется в тактах синхроимпульса).

Частота взятия входных отчетов fs=50 МГц, частота тактирования системы fclk=250 МГц. Коэффициенты фильтра такие же, разрядность представления коэффициентов – 8 бит. Предполагаем, что на вход фильтра поступают только целые значения, как со знаком, так и без, например -5, 3, 1 и 0. Разрядность представления значений входного сигнала, подлежащего фильтрации (B), – 4 бита, профильтрованного сигнала – 14 бит (рис. 4.77). Под дробную часть числа в обоих случаях отводим 0 бит (Input Data Fractional Bits и Output Fractional Bits).

Для размещения проекта в базис ПЛИС XC6SLX4 требуются 57 триггеров, тактируемых фронтом синхросигнала из общих логических ресурсов ПЛИС – LUT – 41, из них 30 используются как логические ресурсы, 8 LUT – для реализации сдвигового регистра, при этом максимальная частота составила 439 МГц.

На рис. 4.78 показан отчет о характеристиках спроектированного КИХ-фильтра на 4 отвода. Частота тактирования ядра фильтра установлена в 250 МГц, а входная частота дискретизации – 50 МГц. Разрядность входной шины данных – 4 бита. Структура коэффициентов фильтра – не симметричная, разрядность представления коэффициентов – 8 бит. Задана полная точность вычисления выходных значений

профильтрованного сигнала. Латентность фильтра 8 тактов синхрочастоты.

На рис. 4.79 представлен проект КИХ-фильтра на 4 отвода в САПР ПЛИС Xilinx ISE 14.2 с использованием генератора параметризированных ядер XLogiCORE IP FIR Compiler Compiler v5.0. Испытательный стенд для моделирования прохождения сигнала по структуре КИХ-фильтра на 4 отвода показывает пример 4. Моделирование прохождения сигнала по структуре КИХ-фильтра демонстрирует рис. 4.80. На вход фильтра подаются значения $-5, 3, 1$.

Разрешение на прием фильтром новых значений определяется высоким уровнем сигнала nd. Готовность фильтра прочитать новую порцию информации определяется выходными стробирующими импульсами сигнала rfd. Результат фильтрации определяется выходными стробирующими импульсами сигнала rdy. Смена профильтрованных значений на выходе фильтра осуществляется через 4 такта синхрочастоты (рис. 4.81).

Рассмотрим случай с параллельной распределенной арифметикой. Частота взятия входных отчетов fs = 250 МГц и частота тактирования системы fclk = 300 МГц. Функция FIR Compiler v5.0 исходя из соотношения частот автоматически определила латентность фильтра 5 тактов синхрочастоты.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;
ENTITY fir4_test_5 IS
END fir4_test_5;
```

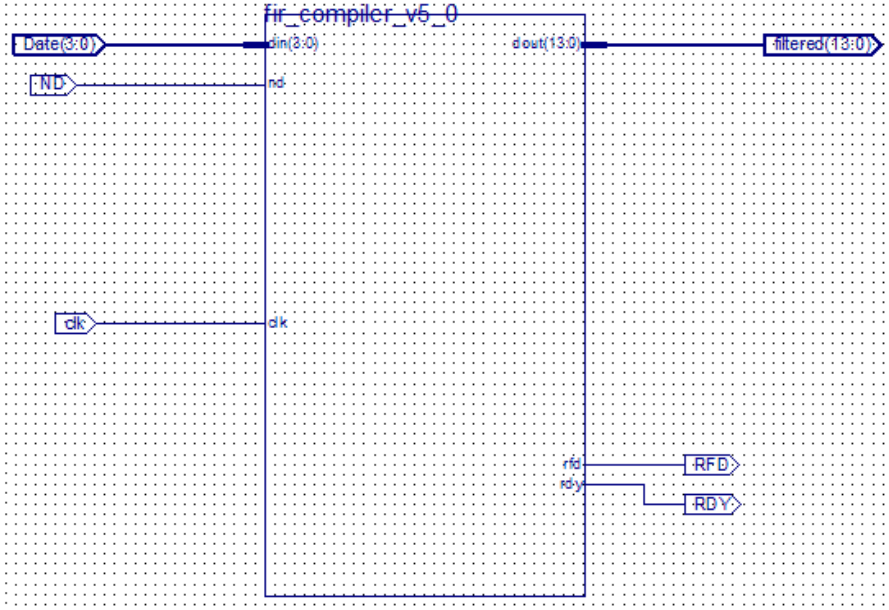


Рис. 4.79. Проект КИХ-фильтра на 4 отвода в САПР ПЛИС Xilinx ISE 14.2 с использованием генератора параметризированных ядер XLogiCORE IP FIR Compiler Compiler v5.0

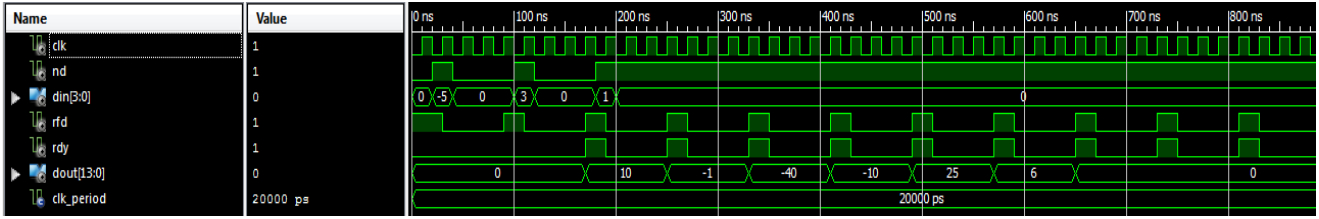


Рис. 4.80. Моделирование прохождения сигнала по структуре КИХ-фильтра на последовательной распределенной арифметике. На вход фильтра подаются значения -5, 3, 1. Правильные значения на выходе 10, -1, -40, -10, 25, 6. Латентность фильтра 8 тактов синхрочастоты

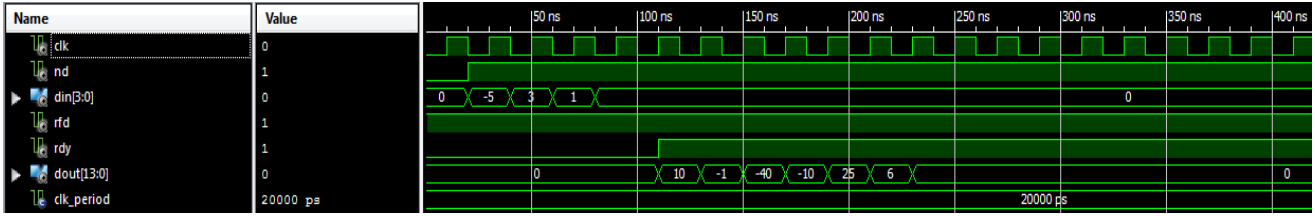


Рис. 4.81. Моделирование прохождения сигнала по структуре КИХ-фильтра на параллельной распределенной арифметике. На вход фильтра подаются значения -5, 3, 1. Правильные значения на выходе 10, -1, -40, -10, 25, 6. Латентность фильтра 5 тактов синхрочастоты

```
ARCHITECTURE behavior OF fir4_test_5 IS
   -- Component Declaration for the Unit Under Test (UUT)
   COMPONENT fir_compiler_v5_0
   PORT(
        clk : IN  std_logic;
        nd : IN  std_logic;
        rfd : OUT  std_logic;
        rdy : OUT  std_logic;
        din : IN  std_logic_vector(3 downto 0);
        dout : OUT  std_logic_vector(13 downto 0)
       );
   END COMPONENT;
  --Inputs
  signal clk : std_logic := '0';
  signal nd : std_logic := '0';
  signal din : std_logic_vector(3 downto 0) := (others => '0');
          --Outputs
  signal rfd : std_logic;
  signal rdy : std_logic;
  signal dout : std_logic_vector(13 downto 0);
  -- Clock period definitions
  constant clk_period : time := 20 ns;
BEGIN
          -- Instantiate the Unit Under Test (UUT)
  uut: fir_compiler_v5_0 PORT MAP (
        clk => clk,
        nd => nd,
        rfd => rfd,
        rdy => rdy,
        din => din,
        dout => dout
       );
  -- Clock process definitions
  clk_process :process
  begin
                    clk <= '0';
```

```
                wait for clk_period/2;
                clk <= '1';
                wait for clk_period/2;
    end process;
    -- Stimulus process
    stim_proc: process
    begin
      -- hold reset state for 100 ns.
      wait for 100 ns;
      wait for clk_period*10;
      -- insert stimulus here
      wait;
    end process;
tb : process
                begin
                        wait for 20 ns;
                        nd<= '1';
                        din <= "1011";
                        wait for 20 ns;
                        nd<= '0';
                        din <= "0000";
                        wait for 20 ns;
                        din <= "0000";
                        wait for 20 ns;
                        din <= "0000";
                        wait for 20 ns;
                        nd<= '1';
                        din <= "0011";
                        wait for 20 ns;
                        nd<= '0';
                        din <= "0000";
                        wait for 20 ns;
                        din <= "0000";
                        wait for 20 ns;
                        din <= "0000";
                        wait for 20 ns;
```

```
                    nd<= '1';
                    din <= "0001";
                    wait for 20 ns;
                    din <= "0000";
                    wait for 20 ns;
                    din <= "0000";
                    wait for 20 ns;
                    din <= "0000";
                    wait for 20 ns;
                    nd<= '1';
                    din <= "0000";
        wait;
END process;
END;
```

Пример 4. Испытательный стенд для моделирования прохождения сигнала по структуре КИХ-фильтра на четыре отвода

Для размещения проекта в базис ПЛИС XC6SLX4 требуется уже 111 триггеров, тактируемых фронтом синхросигнала из общих логических ресурсов ПЛИС, LUT – 88, из них 74 используются как логические ресурсы, 1 LUT функционирует как блок памяти и 1 LUT как сдвиговый регистр, при этом максимальная частота составила 438 МГц. Рис. 4.81 показывает моделирование прохождения сигнала по структуре КИХ-фильтра. На вход фильтра подаются значения -5, 3, 1. Правильные значения на выходе 10, -1, -40, -10, 25, 6. Латентность фильтра составила 5 тактов синхрочастоты.

В табл. 4.6 приведен анализ задействованных ресурсов ПЛИС XC6SLX4 при реализации КИХ-фильтров на 4 отвода с использованием функции FIR Compiler v6.3 и FIR Compiler Compiler v5.0 САПР Xilinx ISE 14.2. Из анализа табл. 4.2 следует, что наиболее быстродействующими являются фильтры на распределенной арифметике.

Рассмотрим размещение КИХ-фильтра в ресурсы ПЛИС Xilinx XC6SLX4 (рис. 4.82). Во всех случаях установлена целевая задача проектирования – сбалансированная.

Таблица 4.6

Анализ задействованных ресурсов ПЛИС XC6SLX4 при реализации КИХ-фильтров на 4 отвода с использованием функции FIR Compiler v6.3 и FIR Compiler v5.0 САПР Xilinx ISE 14.2

Ресурсы ПЛИС	Систоли-ческий	Распределенная арифметика	
		Последовательная	Параллель-ная
Триггеры логических блоков в секциях	48	57	111
Секций с LUT	33	41	88
LUT для выполнения комбинационных функций	22	30	74
LUT как блоки памяти	11	8	1
LUT как сдвиговые регистры	11	8	1
Кол-во ЦОС-блоков DSP48A1	1	–	–
Латентность фильтра	11	8	5
Рабочая частота, МГц	348	439	438

Кристалл ПЛИС разбит на 8 тактовых регионов (доменов). КИХ-фильтр с использованием систолической структуры и ЦОС-блока занимает три клоковых региона, а фильтры на последовательной и параллельной арифметике – по два. Последовательная занимает меньшее число ресурсов ПЛИС, но они более широко разбросаны по кристаллу, что может увеличивать задержки в трассировочных ресурсах, а параллельная при более значительном потреблении ресурсов более локализована. Области локализации для систолического

фильтра (рис. 4.82а) и фильтров с использованием распределенной арифметики (рис. 4.82а и 4.82б) показаны красными овалами.

В случае КИХ-фильтров на параллельной арифметике выходной сигнал (профильтрованные значения) формируется через каждый синхроимпульс, а для фильтра на последовательной арифметике с несимметричной – через B и через B+1 для фильтра с симметричной импульсной характеристикой, т.е. в нашем случае через 4 такта синхроимпульса.

Отказ от использования в функции FIR Compiler v6.3 структур фильтров на распределенной арифметике говорит о том, что в настоящее время идет ориентация на массовое использование ЦОС-блоков в ПЛИС, но в то же время ведущие разработчики САПР Xilinx и Altera сохранили возможность использования распределенной арифметики из-за ряда преимуществ. Например, структуры КИХ-фильтров на основе распределенной арифметики обладают рекордным быстродействием, которое не снижается с ростом числа отводов.

Такие решения особенно эффективны в низкобюджетных сериях ПЛИС, где существует недостаточное число встроенных аппаратных ЦОС-блоков. Фильтрам на распределенной арифметике присущи такие недостатки, как меньшая точность представления коэффициентов и входных отсчетов, например КИХ-фильтры на систолических структурах для ПЛИС серий Vitrex-5/6 позволяют иметь максимальную точность представления коэффициентов 49 бит против 32 бит, и их нельзя перегрузить в режиме онлайн. Распределенная арифметика не позволяет также реализовать полифазный банк фильтров и параллельную потоковую обработку информации.

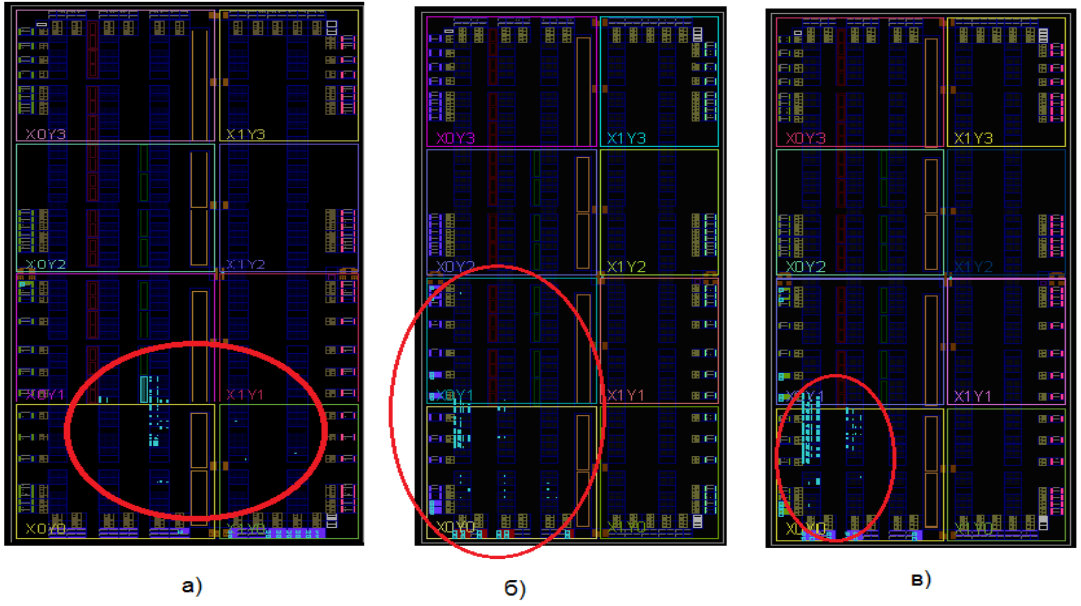


Рис. 4.82. Размещение КИХ-фильтра на четыре отвода в ресурсы ПЛИС XC6SLX4: а) систолическая структура с использованием ЦОС-блока; б) последовательная арифметика; в) параллельная арифметика

В заключении отметим, что распределенная арифметика представляет собой иной подход к реализации цифровых фильтров. Распределенная арифметика основывается на том факте, что коэффициенты фильтра известны, поэтому операция умножения $C_k \cdot x_k$ становится умножением на константу.

Основная идея заключается в замене всех сложных операций умножения и сложения простыми последовательностями табличных просмотров, сложений, вычитаний и сдвигов входных последовательностей данных.

Для этого необходима одна LUT-таблица, содержащая комбинацию сумм коэффициентов, являющихся константами, всех возможных вариантов на ее адресных входах, накапливающий (масштабирующего) сумматор и многоразрядный сдвиговый регистр.

Результаты показывают, что использование распределенной арифметики является наилучшим выбором для реализации КИХ-фильтров в базисе ПЛИС. Последовательная распределенная арифметика демонстрирует высокую эффективность в отношении потребляемых логических ресурсов ПЛИС, но ухудшает быстродействие и производительность фильтров.

Параллельная распределенная арифметика показывает высокое быстродействие КИХ-фильтров при их реализации в базис ПЛИС за счет использования «безумножительных» схем умножения, не снижаемое при увеличении его длины, а также точности представления входных данных. Что особенно актуально для проектов использующих отечественные ПЛИС, в которых отсутствуют аппаратные умножители или их количество ограничено, но обладающих большим количеством логических ресурсов.

5. КИХ-ФИЛЬТРЫ СИСТОЛИЧЕСКОЙ И ТРАНСПОНИРОВАННОЙ ФОРМЫ ДЛЯ РЕАЛИЗАЦИИ В БАЗИСЕ ПЛИС

5.1. Проектирование систолических КИХ-фильтров в базисе ПЛИС с использованием САПР Quartus II

Систолический КИХ-фильтр считается оптимальным решением для параллельных архитектур цифровых фильтров. В настоящее время входит в состав мегафункции (ALTMULT_ADD) САПР Quartus II, начиная с версий 11, 12 и 13, для работы с ЦОС-блоками серий Cyclon V, Arria V и Stratix V, выполненных по 28 нм технологическому процессу, и функции XtremeDSP™ Digital Signal Processing для ЦОС-блока DSP48 ПЛИС серии Virtex-4 Xilinx.

В фон-неймановских машинах данные, считанные из памяти, однократно обрабатываются в процессорном элементе (ПЭ), после чего снова возвращаются в память (рис. 5.1а). Авторы идеи систолической матрицы Кунг и Лейзерсон предложили организовать вычисления так, чтобы данные на своем пути от считывания из памяти до возвращения обратно пропускались через как можно большее число ПЭ (рис. 5.1б).

Если сравнить положение памяти в вычислительных системах со структурой живого организма, то по аналогии ей можно отвести роль сердца, множеству ПЭ – роль тканей, а поток данных рассматривать как циркулирующую кровь. Отсюда и происходит название систолическая матрица (систола – сокращение предсердий и желудочков сердца, при котором кровь нагнетается в артерии). Систолическая структура – это однородная вычислительная среда из процессорных элементов, совмещающая в себе свойства конвейерной и матричной обработки. Систолические структуры эффективны при выполнении матричных вычислений, обработке сигналов, сортировке данных и т. д.

Рассмотрим структуру систолического КИХ-фильтра на 4 отвода (рис. 5.2). В основе структуры лежит ритмическое прохождение двух потоков данных x_{in} и y_{in} навстречу друг другу.

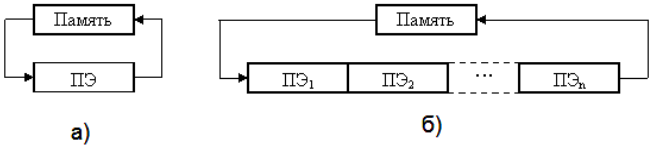


Рис. 5.1. Обработка данных в вычислительных системах:
а) фон-неймановского типа; б) систолической структуры

Последовательные элементы каждого потока разделены одним тактовым периодом, чтобы любой из них мог встретиться с любым элементом встречного потока. Вычисления выполняются параллельно в процессорных элементах, каждый из которых реализует один шаг в операции вычисления скалярного произведения (рис. 5.2). Значение y_{in}, поступающее на вход ПЭ, суммируется с произведением входных значений x_{in} и c_k. Результат выходит из ПЭ как $y_{out} = y_{in} + c_k * x_{in}$. Значение x_{in}, кроме того, для возможного последующего использования остальной частью массива транслируется через ПЭ без изменений и покидает его в виде x_{out}. Таким образом, на каждый отвод фильтра приходится один ПЭ. На рис. 5.2 показана структура систолического КИХ-фильтра на 4 отвода. Потоки сигналов y_{in}, представляющего накопленный результат вычисления скалярного произведения, и входного x_{in} распространяются слева направо. На входах x_{in} и выходах суммы каждого ПЭ добавлены регистерные элементы, что позволяет накопленному результату и входному значению оставаться в синхронизации.

Для получения конечного результата используется сеть сумматоров, которая последовательно суммирует скалярные произведения. Фильтр состоит из двух однотипных процессорных элементов ПЭ1 и ПЭ2. На вход y_{in} ПЭ1 необходимо подать сигнал логического нуля, а на входной линии x_{in} используется один регистр, а не два, как у ПЭ2.

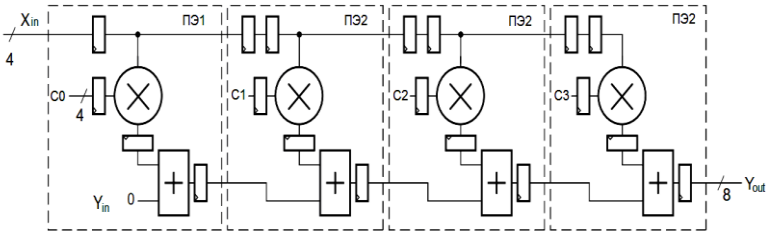


Рис. 5.2. Систолический КИХ-фильтр на 4 отвода, составленный из четырех секций DSP48 ПЛИС Virtex-4 фирмы Xilinx

Используя представленные выше соображения, разработаем модель систолического фильтра на 4 отвода $y = C_0x_0 + C_1x_1 + C_2x_2 + C_3x_3$ в САПР ПЛИС Quartus II в базисе ПЛИС серии Stratix III. Предположим, что коэффициенты фильтра целочисленные со знаком известны и равны $C_0 = -2$, $C_1 = -1$, $C_2 = 7$ и $C_3 = 6$. На вход КИХ-фильтра поступают входные отсчеты -5, 3, 1 и 0. Правильные значения на выходе фильтра: 10, -1, -40, -10, 26, 6 и т.д., т.е. согласно модели. В качестве умножителей используется мегафункция LPM_MULT. Регистры выполнены на мегафункциях LPM_FF. Систолический КИХ-фильтр на 4 отвода с процессорными элементами ПЭ1 и ПЭ2 в САПР ПЛИС Quartus II показан на рис. 5.3. На рис. 5.4 и 5.5 показаны схемы процессорных элементов ПЭ1 и ПЭ2, а на рис. 5.6 – временные диаграммы работы систолического КИХ-фильтра на 4 отвода.

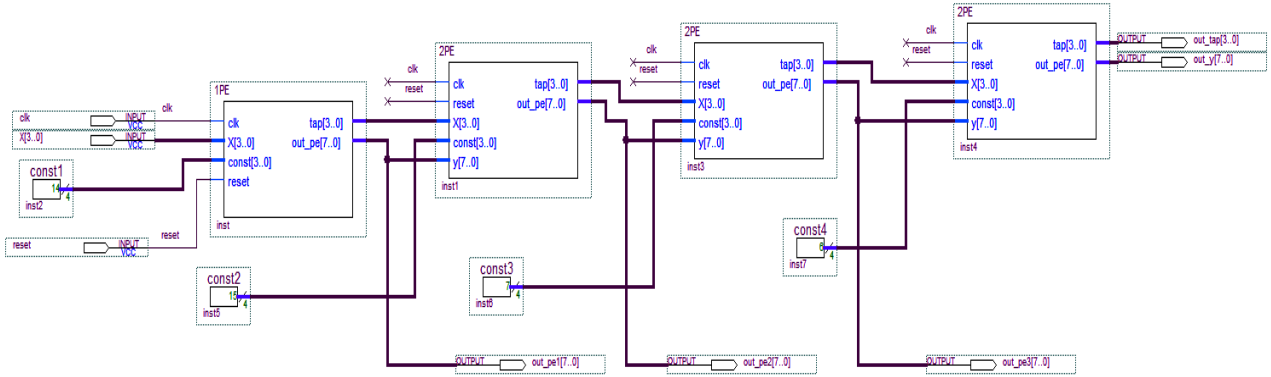


Рис. 5.3. Систолический КИХ-фильтр на четыре отвода в САПР ПЛИС Quartus II с процессорными элементами ПЭ1 и ПЭ2

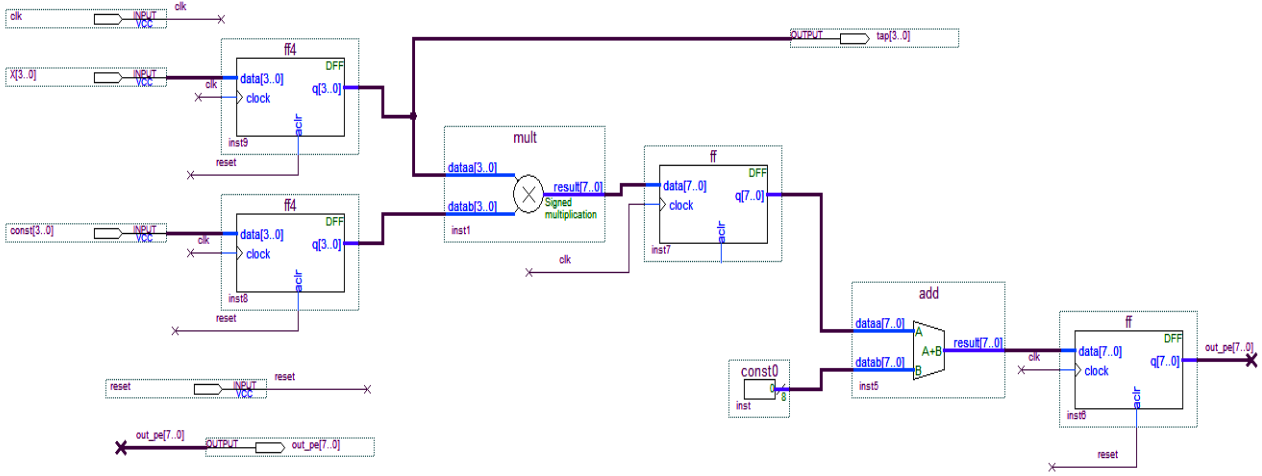


Рис. 5.4. Первый процессорный элемент ПЭ1

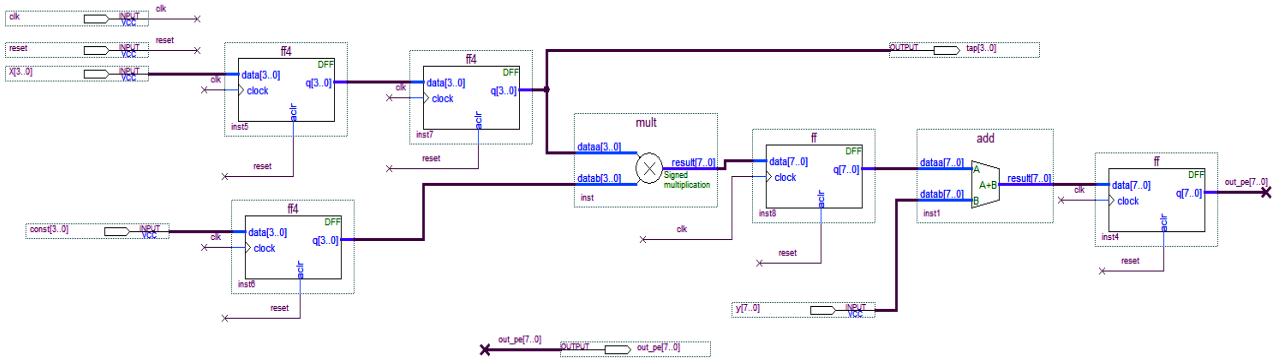


Рис. 5.5. Второй процессорный элемент ПЭ2

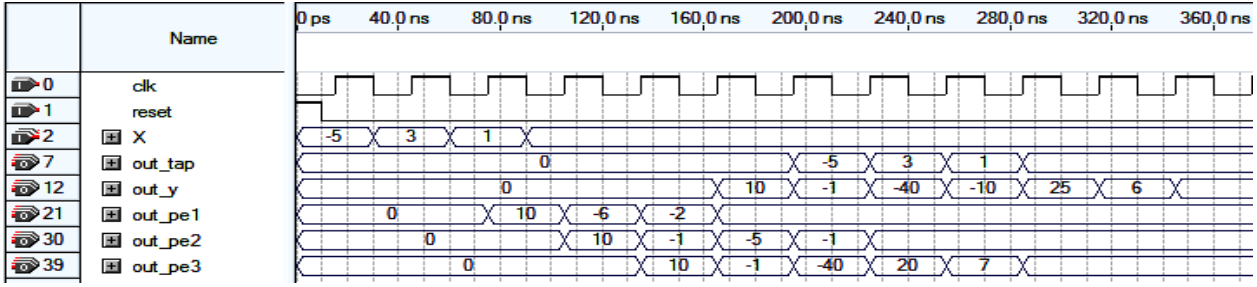


Рис. 5.6. Временные диаграммы работы систолического КИХ-фильтра на 4 отвода с процессорными элементами ПЭ1 и ПЭ2

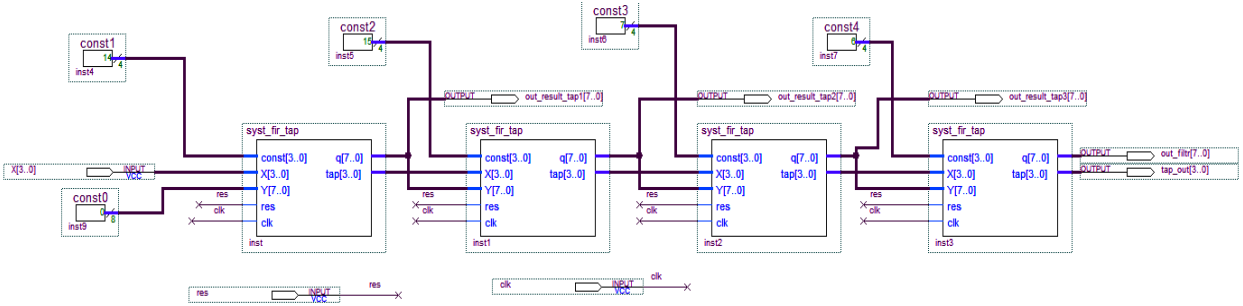


Рис. 5.7. Систолический КИХ-фильтр на 4 отвода в САПР ПЛИС Quartus II с однотипными процессорными элементами

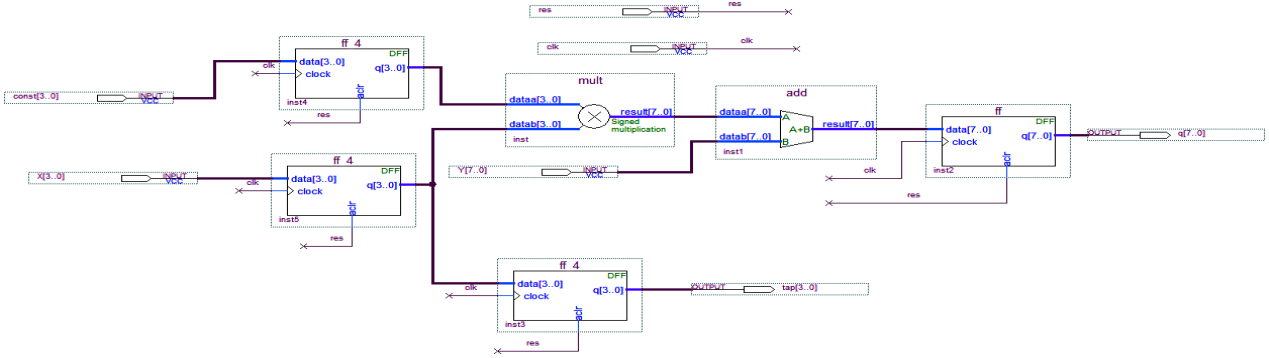


Рис. 5.8. Однотипный процессорный элемент

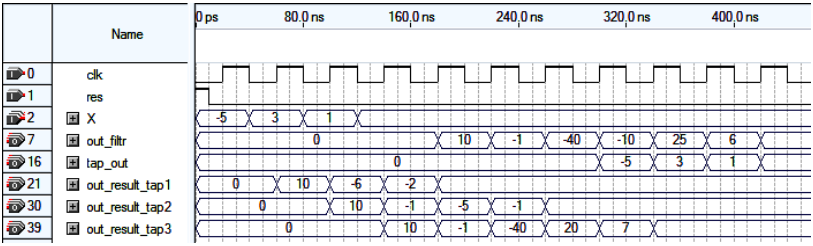


Рис. 5.9. Временные диаграммы работы систолического КИХ-фильтра на 4 отвода с однотипными процессорными элементами

На входах перемножителя сигналов, выполненных на мегафункции ALT_MULT в процессорном элементе 1PE, необходимы по одному 4-разрядному регистру для процесса согласования вычислений. А на одном из входов перемножителя сигналов в процессорном элементе 2PE необходимы два 4-разрядных регистра, первый из которых играет роль линии задержки.

Упростить структуру систолического КИХ-фильтра позволяет использование однотипного процессорного элемента (рис. 5.7 и рис.5.8). Правильность функционирования такого решения подтверждает диаграмма на рис. 5.9.

Пример 1 демонстрирует код языка VHDL КИХ-фильтра с использованием прямой реализации по формуле $y = C_0x_0 + C_1x_1 + C_2x_2 + C_3x_3$. По коду были получены временные диаграммы, показанные на рис. 5.10. Сравнивая временные диаграммы на рис. 5.6, 5.9 и 5.10, видим, что фильтры работают корректно.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
package coeffs is
  type coef_arr is array (0 to 3) of signed(3 downto 0);
  constant coefs: coef_arr:= coef_arr'("1110", "1111", "0111", "0110");
  end coeffs;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.coeffs.all;
entity fir_var is
 port (clk, reset, clk_ena: in std_logic;
       date: in signed (3 downto 0);
       q_reg: out signed (9 downto 0));
end fir_var;
architecture beh of fir_var is
begin
process(clk,reset)
      type shift_arr is array (3 downto 0)
      of signed (3 downto 0);
      variable shift: shift_arr;
      variable tmp: signed (3 downto 0);
      variable pro: signed (7 downto 0);
      variable acc: signed (9 downto 0);
   begin
if reset='0' then
for i in 0 to 3 loop
shift(i):= (others => '0');
end loop;
q_reg<= (others => '0');
elsif(clk'event and clk = '1') then
if clk_ena='1' then
                        shift(0):=date;
                        pro := shift(0) * coefs(0);
                        acc := conv_signed(pro, 10);
                         for i in 2 downto 0 loop
                        pro := shift(i+1) * coefs(i+1);
                        acc := acc + conv_signed(pro, 10);
                        shift(i+1):= shift(i);
                        end loop;
end if;
end if;
          q_reg<=acc;
end process;
```

end beh;

Пример 1. Код языка VHDL КИХ-фильтра на четыре отвода

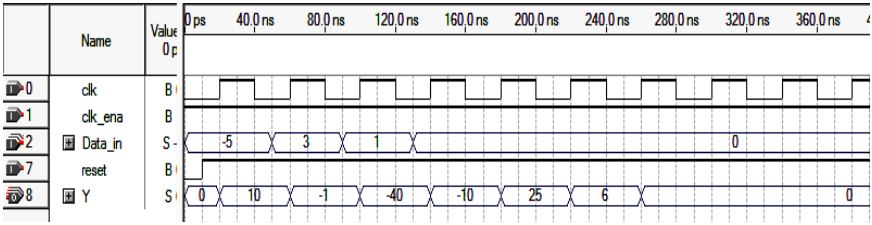


Рис. 5.10. Временные диаграммы работы КИХ-фильтра на 4 отвода по коду языка VHDL

В качестве примера рассмотрим индустриальные ПЛИС предпоследнего поколения фирмы Altera серии Cyclon V и Stratix V. В указанных сериях появились ЦОС-блоки с переменной точностью, одна из особенностей которых – это реализация систолических КИХ-фильтров.

Каждый ЦОС-блок серии Stratix V позволяет реализовать два перемножителя двух 18-разрядных сигналов или один перемножитель двух 32-разрядных сигналов, причем встроенные в выходные цепи ЦОС-блоков многоразрядные сумматоры позволяют организовать первый и второй уровень в дереве многоразрядных сумматоров в случае проектирования параллельных КИХ-фильтров.

На рис. 5.11 показана прямая форма КИХ-фильтра с несимметричными коэффициентами на 16 отводов, коэффициенты представлены с 18-битной точностью, с использованием ЦОС-блоков ПЛИС серии Stratix V. В ЦОС-блоках используются два уровня суммирования по отношению к внешнему дереву многоразрядных сумматоров, каскадирующая шина, и, как вариант, линия задержки может быть сконфигурирована из внутренних входных регистров.

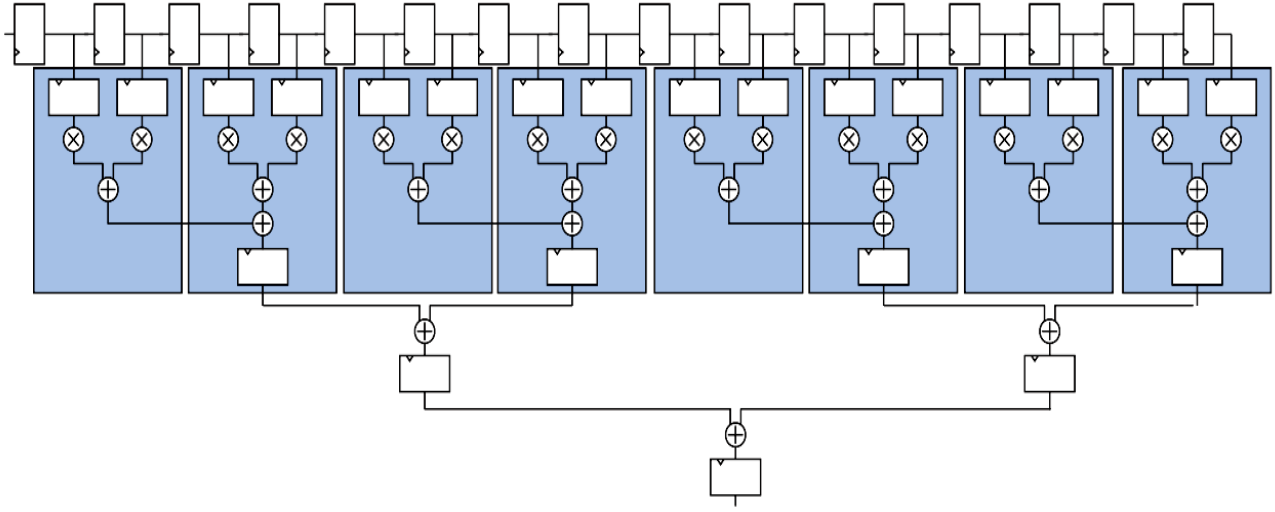
На рис. 5.12 показана прямая форма КИХ-фильтра на 8 отводов с несимметричными коэффициентами с

использованием ЦОС-блоков ПЛИС серии Stratix V. Для реализации фильтров с большим числом отводов необходимы внешние линия задержки и дерево многоразрядных сумматоров по отношению к ЦОС-блокам, при этом в самом ЦОС-блоке осуществляется первый уровень суммирования значений с отводов фильтра, предварительно умноженных на его коэффициенты.

Использование ЦОС-блоков при проектировании КИХ-фильтра прямой формы позволяет вдвое сократить число многоразрядных сумматоров, но оставшаяся часть дерева сумматоров реализуется на внутренних ресурсах ПЛИС, что отрицательно сказывается на общем числе задействованных ресурсов, и, как следствие, снижается быстродействие фильтра за счет увеличения задержек в трассировочных каналах самой ПЛИС.

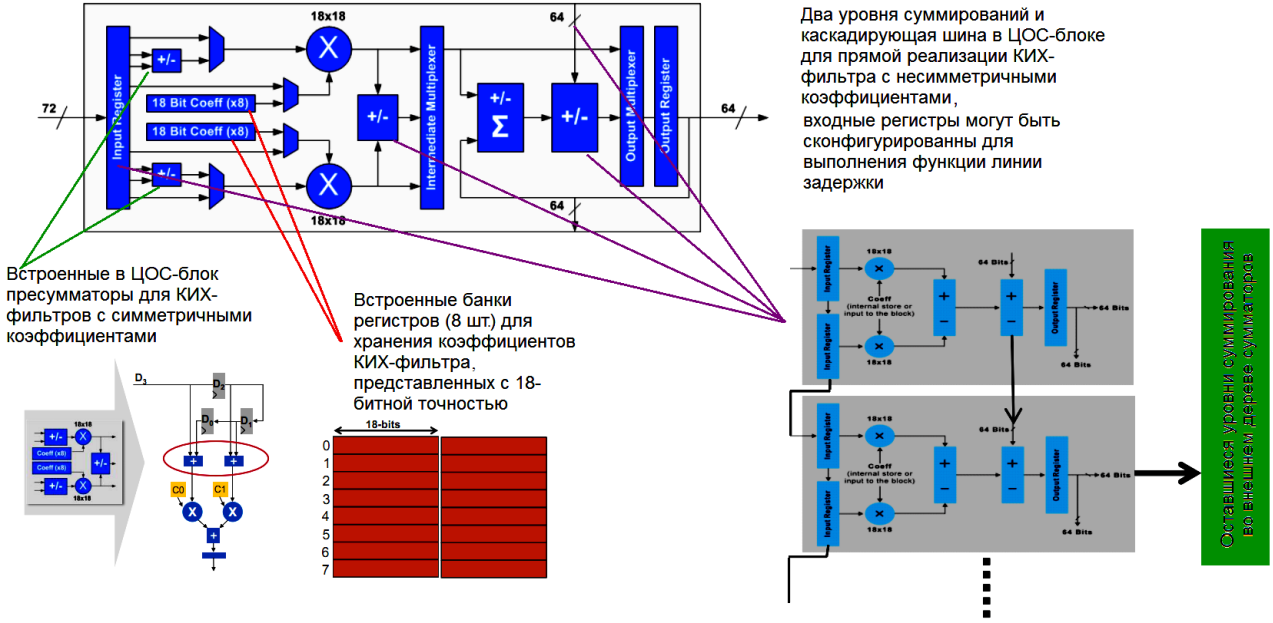
Существенно уменьшить число используемых ресурсов позволяет систолический КИХ-фильтр, в ЦОС-блоках которого реализуются операции умножения и сложения с конвейеризацией, т. е. отпадает необходимость в дереве многоразрядных сумматоров, однако он может иметь большую латентность по сравнению с прямой реализацией фильтра (рис. 5.13). Мегафункция ALTMULT_ADD САПР Quartus II начиная с версии 11.0 для ПЛИС серий Arria V, Cyclone V и Stratix V позволяет конфигурировать ЦОС-блоки для организации систолических фильтров.

На рис. 5.14 представлена структура ЦОС-блока с переменной точностью ПЛИС серии Cyclon V, на которой выделен процессорный элемент, а на рис. 5.15 показано включение систолического регистра в мегафункции ALTMULT_ADD для ПЛИС серии Cyclon V.



а)

Рис. 5.11. а) прямая форма КИХ-фильтра с несимметричными коэффициентами на 16 отводов, коэффициенты представлены с 18-битной точностью с использованием ЦОС-блоков ПЛИС серии Stratix V; б) режимы ЦОС-блока для прямой реализации КИХ-фильтра с симметричными и несимметричными коэффициентами



б)

Рис. 5.11. а) прямая форма КИХ-фильтра с несимметричными коэффициентами на 16 отводов, коэффициенты представлены с 18-битной точностью с использованием ЦОС-блоков ПЛИС серии Stratix V; б) режимы ЦОС-блока для прямой реализации КИХ-фильтра с симметричными и несимметричными коэффициентами (продолжение)

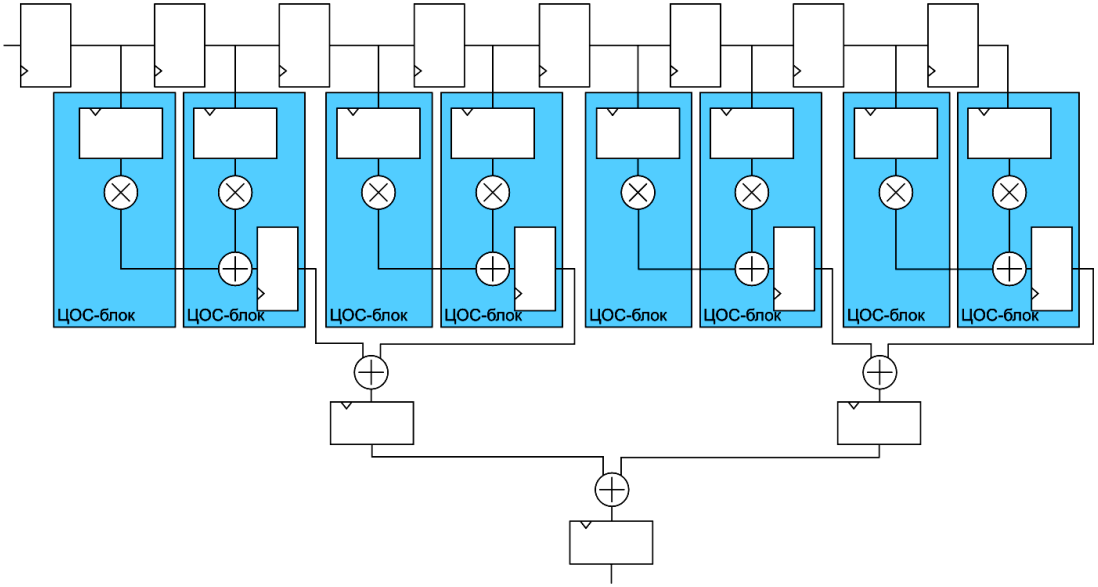


Рис. 5.12. Прямая реализация КИХ-фильтра с несимметричными коэффициентами на 8 отводов, коэффициенты представлены с 27-битной точностью (высокоточный режим) с использованием ЦОС-блоков ПЛИС серии Stratix V. В ЦОС-блоках используется один уровень суммирования по отношению к внешнему дереву многоразрядных сумматоров

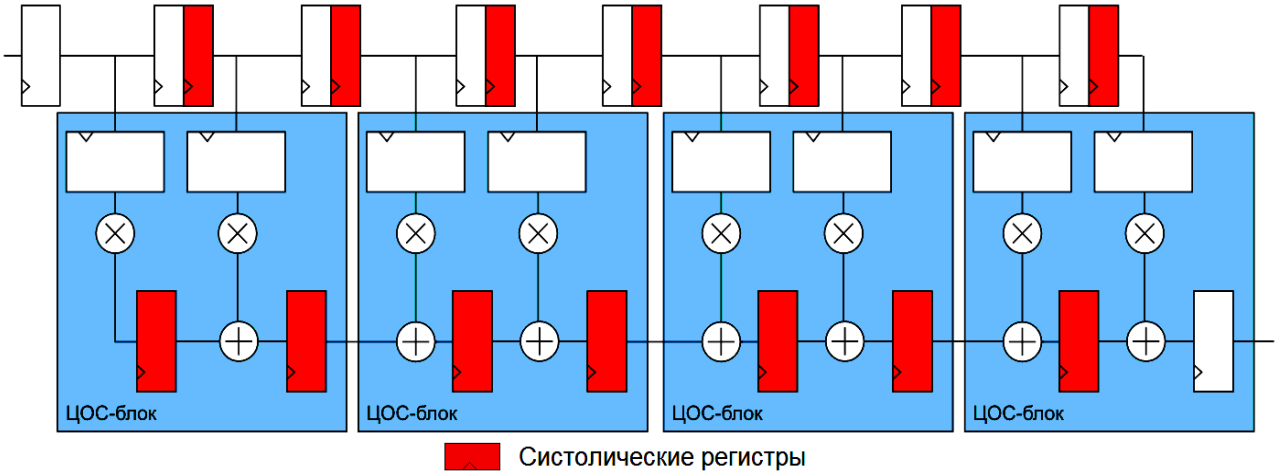


Рис. 5.13. Систолический КИХ-фильтр на 8 отводов с 18-битной точностью представления коэффициентов с использованием ЦОС-блоков ПЛИС серии Stratix V

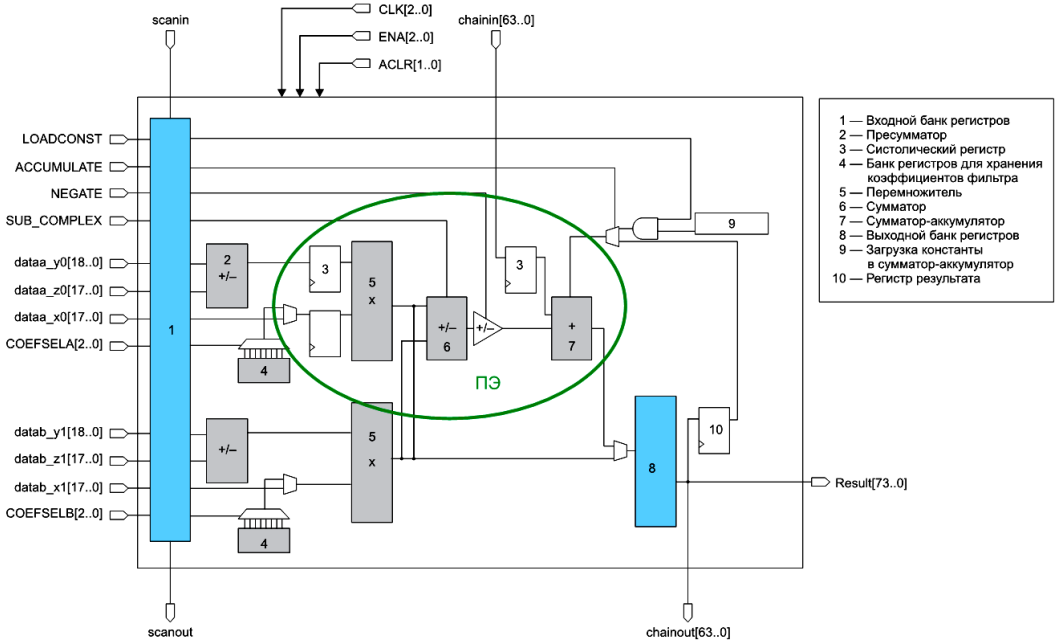
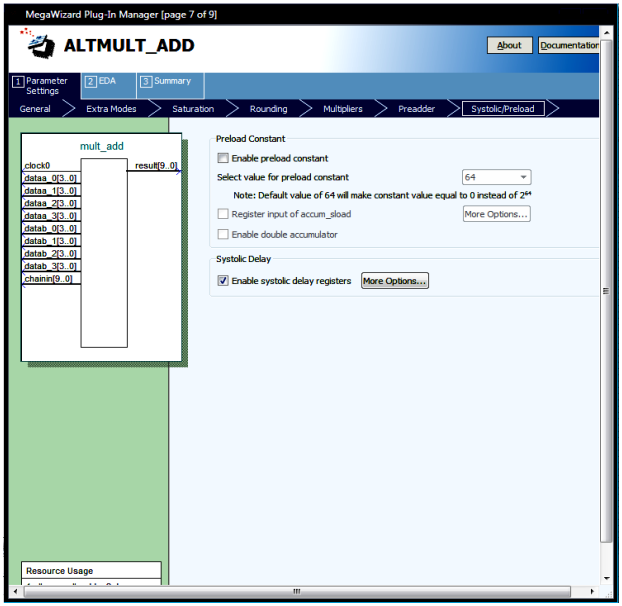


Рис. 5.14. Систолические регистры, подключаемые опционально с помощью мегафункции ALTMULT_ADD в ЦОС-блоке с переменной точностью ПЛИС серии Cyclon V



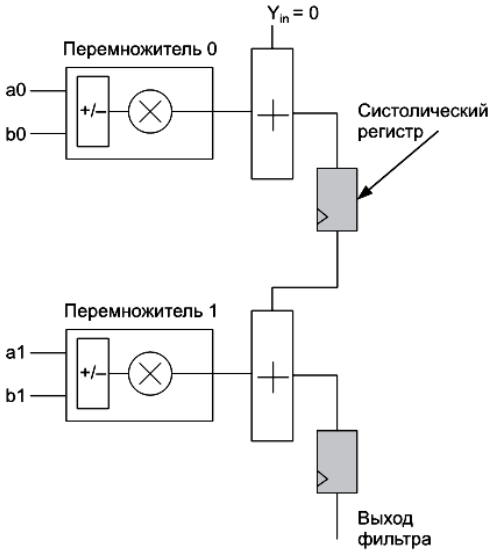


Рис. 5.15. Включение систолического регистра в последовательную цепь сумматоров в мегафункции ALTMULT_ADD для ПЛИС серии Cyclon V

Таблица 5.1

Реализация параллельных КИХ-фильтров на 4 отвода в базис ПЛИС серии Stratix III

Ресурсы ПЛИС	Систолический КИХ-фильтр с однотипными процессорными элементами без использования встроенных ЦОС-блоков	Четыре мегафункции умножения с накоплением ALTMULT_ACCUM (внешняя линия задержки и дерево сумматоров)	Мегафункция умножения и сложения ALTMULT_ADD (внешняя линия задержки)
Максимальная частота/ частота в наихудшем случае, МГц	432/400	429/400	514/400
Число LUT для реализации комбинационныъх функций	59	18	0
Адаптивных логических модулей (ALM)	44	18	8
Число выделенных регистров для реализации последовательностной логики	65	12	12
Встроенные ЦОС-блоки, 18×18 бит	–	16	4

Реализации параллельных КИХ-фильтров на 4 отвода в базис ПЛИС серии Stratix III представлены в табл. 5.1. Все рассмотренные варианты фильтров реализованы на мегафункциях. Первый вариант реализован без использования мегафункциями ЦОС-блоков (рис. 5.7), второй вариант задействует лишь по одному умножителю в блоке из четырех возможных мегафункций ALTMULT_ACCUM, а третий – 4 умножителя из четырех возможных в блоке мегафункции ALTMULT_ADD. Приведенная таблица показывает оптимальное использование ресурсов ЦОС-блоков мегафункцией ALTMULT_ADD за счет использования четырех перемножителей и встроенного дерева сумматоров в блоке, реализующих первые и вторые уровни суммирования. Во всех случаях частота работы фильтров ограничивается величиной 400 МГц.

5.2. Проектирование систолических КИХ-фильтров в базисе ПЛИС с использованием системы цифрового моделирования ModelSim-Altera

Кратко рассмотрим особенности проектирования цифровых фильтров на примере систолического КИХ-фильтра в САПР ПЛИС Quartus II версии 11.1 Web Edition. Начиная с версии 10.0, из САПР Quartus II исключен векторный редактор, а моделирование предлагается вести с использованием различных симуляторов высокоуровневых языков описания аппаратурных средств, например Active-HDL, Riviera-Pro, ModelSim и др. В качестве свободно распространяемого симулятора с ограниченными возможностями пользователю предлагается использовать систему моделирования ModelSim-Altera Free.

Рассмотрим уравнение КИХ-фильтра (нерекурсивного цифрового фильтра с конечно-импульсной характеристикой),

которое представляется как арифметическая сумма произведений:

$$y = \sum_{k=0}^{K-1} c_k \cdot x_k \,, \qquad (5.1)$$

где $y-$ отклик цепи; x_k- k-я входная переменная; c_k- весовой коэффициент k-й входной переменной, который является постоянным для всех n; $K-$ число отводов фильтра.

Разработаем модель систолического фильтра на 4 отвода $y = C_0x_0 + C_1x_1 + C_2x_2 + C_3x_3$ в САПР ПЛИС Quartus II версии 11.1 в базисе ПЛИС серии Cyclone II с однотипными процессорными элементами (рис. 5.16 и 5.17). ПЛИС серии Cyclone II выбраны из соображений, что в САПР ПЛИС Quartus II Web Edition не поддерживаются ПЛИС серий Cyclon V и Stratix V.

Дадим произвольное имя файлу верхнего уровня проектной иерархии – poly_syst_main.bdf. Предположим, что коэффициенты фильтра целочисленные со знаком известны и равны $C_0 = -2$, $C_1 = -1$, $C_2 = 7$ и $C_3 = 6$. На вход КИХ-фильтра поступают входные отсчеты -5, 3, 1 и 0. Правильные значения на выходе фильтра: 10, -1, -40, -10, 26, 6 и т.д., т.е. согласно формуле.

На рис. 5.18 показаны временные диаграммы работы систолического КИХ-фильтра, реализованного в базисе ПЛИС Stratix III на 4 отвода с однотипными процессорными элементами в САПР ПЛИС Quartus II версии 9.1.

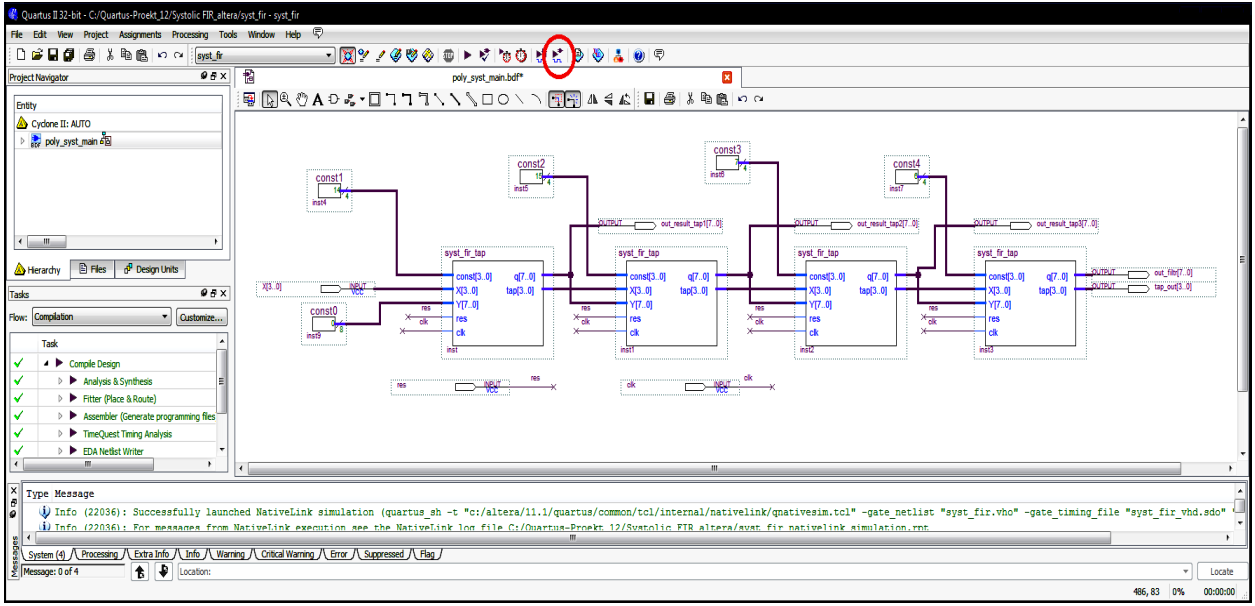


Рис. 5.16. Систолический КИХ-фильтр на 4 отвода в САПР ПЛИС Quartus II версии 11.1 с однотипными процессорными элементами

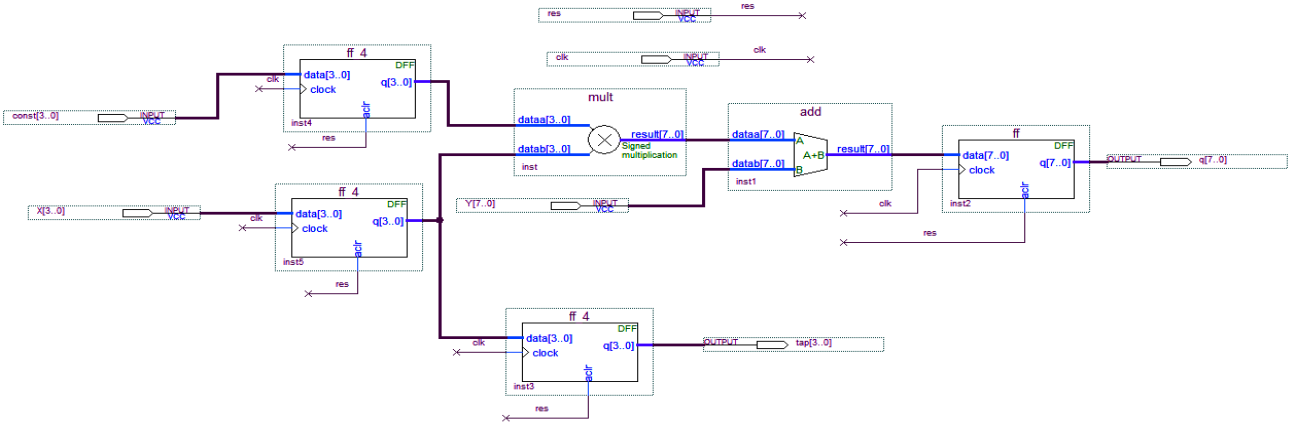


Рис. 5.17. Однотипный процессорный элемент

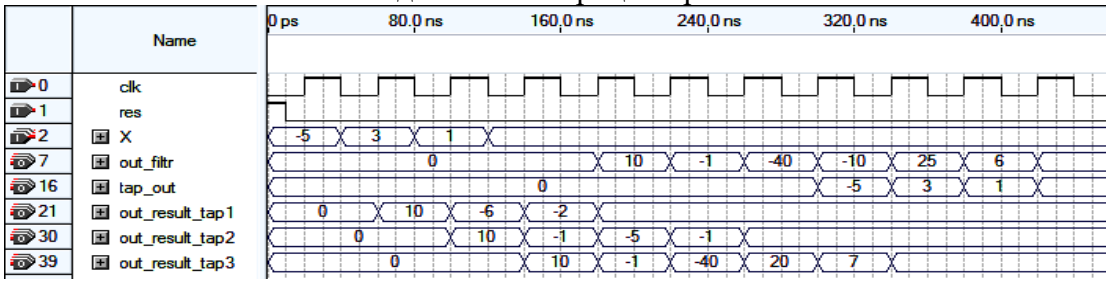


Рис. 5.18. Временные диаграммы работы систолического КИХ-фильтра на 4 отвода с однотипными процессорными элементами в САПР ПЛИС Quartus II версии 9.1

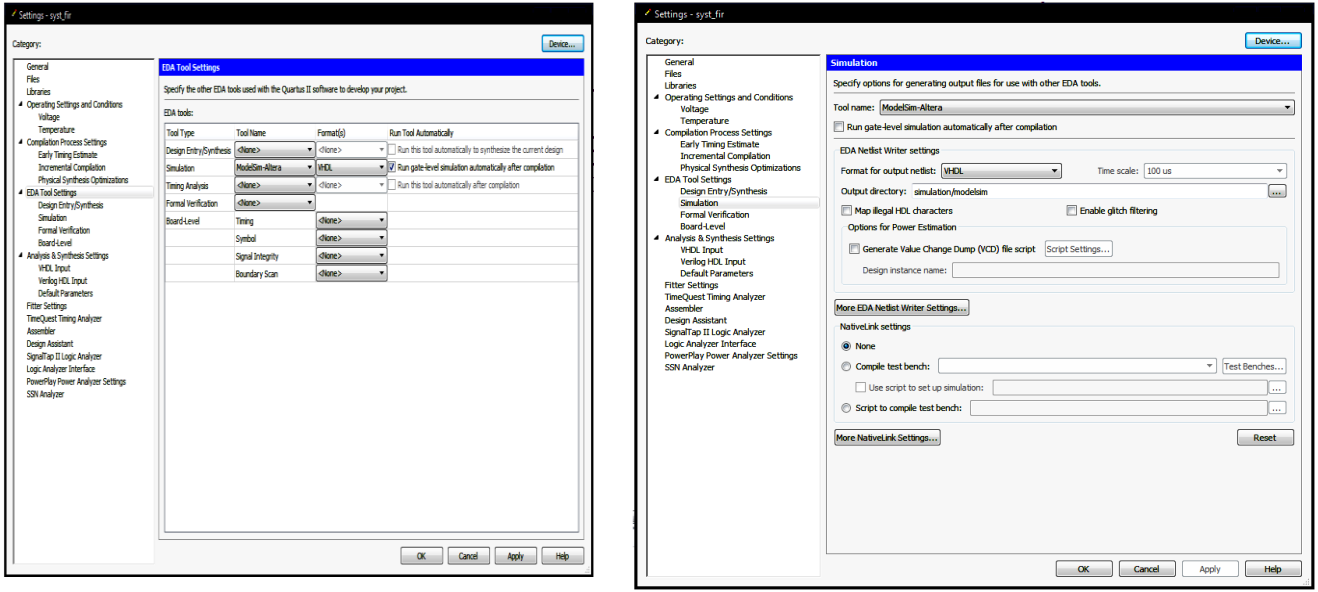


Рис. 5.19. Настройки закладки EDA Tool Settings САПР ПЛИС Quartus II версии 11.1

Рассмотрим моделирование КИХ-фильтра с использованием симулятора ModelSim-Altera STARTER EDITION. Предварительно его необходимо подключить. Это осуществляется с помощью меню Assigments/settings/EDA Tool settings. В поле Tool name САПР ПЛИС Quartus II версии 11.1 необходимо выбрать симулятор ModelSim-Altera, а в поле Format Netlist Writer settings указать выходной формат списка межсоединений цепей – язык VHDL (рис. 5.19).

Предварительно необходимо создать текстовый сценарий функционирования систолического КИХ-фильтра с использованием структурного стиля языка VHDL («тестбенч» или испытательный стенд). В качестве промежуточного шаблона, который необходимо отредактировать, можно взять файл poly_syst_main.vht. Для этого необходимо его сформировать следующими действиями: меню Processing/Start/Start Test Bench Template Writer (пример 2). Отредактируем объект poly_syst_main_vhd_tst, который основан на компоненте poly_syst_main, и сохраним его под именем test_poly_syst_main.vhd (пример 3).

```
-- Vhdl Test Bench template for design  :  poly_syst_main
-- Simulation tool : ModelSim-Altera (VHDL)

LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY poly_syst_main_vhd_tst IS
END poly_syst_main_vhd_tst;
ARCHITECTURE poly_syst_main_arch OF poly_syst_main_vhd_tst IS
-- constants
-- signals
SIGNAL clk : STD_LOGIC;
SIGNAL out_filtr : STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL out_result_tap1 : STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL out_result_tap2 : STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL out_result_tap3 : STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL res : STD_LOGIC;
```

```
SIGNAL tap_out : STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL X : STD_LOGIC_VECTOR(3 DOWNTO 0);
COMPONENT poly_syst_main
        PORT (
        clk : IN STD_LOGIC;
        out_filtr : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        out_result_tap1 : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        out_result_tap2 : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        out_result_tap3 : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        res : IN STD_LOGIC;
        tap_out : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
        X : IN STD_LOGIC_VECTOR(3 DOWNTO 0)
        );
END COMPONENT;
BEGIN
        i1 : poly_syst_main
        PORT MAP (
-- list connections between master ports and signals
        clk => clk,
        out_filtr => out_filtr,
        out_result_tap1 => out_result_tap1,
        out_result_tap2 => out_result_tap2,
        out_result_tap3 => out_result_tap3,
        res => res,
        tap_out => tap_out,
        X => X
        );
init : PROCESS
-- variable declarations
BEGIN
      -- code that executes only once
WAIT;
END PROCESS init;
always : PROCESS
-- optional sensitivity list
-- (       )
-- variable declarations
BEGIN
```

```
    -- code executes for every event on sensitivity list
WAIT;
END PROCESS always;
END poly_syst_main_arch;
```

Пример 2. Шаблон теста систолического КИХ-фильтра на языке VHDL (poly_syst_main.vht)

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;
ENTITY test_poly_syst_main IS
END test_poly_syst_main;

ARCHITECTURE behavior OF test_poly_syst_main IS
COMPONENT poly_syst_main
PORT(
res :  IN  STD_LOGIC;
clk :  IN  STD_LOGIC;
X :  IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
out_filtr :  OUT  STD_LOGIC_VECTOR(7 DOWNTO 0);
out_result_tap1 :  OUT  STD_LOGIC_VECTOR(7 DOWNTO 0);
out_result_tap2 :  OUT  STD_LOGIC_VECTOR(7 DOWNTO 0);
out_result_tap3 :  OUT  STD_LOGIC_VECTOR(7 DOWNTO 0);
tap_out :  OUT  STD_LOGIC_VECTOR(3 DOWNTO 0));
END COMPONENT;
--Inputs
SIGNAL clk :  std_logic := '0';
SIGNAL res :  std_logic := '1';
SIGNAL X_in:  STD_LOGIC_VECTOR(3 DOWNTO 0) := "1011";
--Outputs
SIGNAL out_systol_filtr : std_logic_VECTOR(7 DOWNTO 0);
SIGNAL out_result_tap1 :  std_logic_vector(7 downto 0);
SIGNAL out_result_tap2 :  std_logic_vector(7 downto 0);
SIGNAL out_result_tap3 :  std_logic_vector(7 downto 0);
SIGNAL tap_out :  std_logic_vector(3 downto 0);
BEGIN
```

```
        uut: poly_syst_main PORT MAP(
                clk => clk,
                res => res,
                X => X_in,
                out_filtr => out_systol_filtr,
                out_result_tap1 => out_result_tap1,
                out_result_tap2 => out_result_tap2,
                out_result_tap3 => out_result_tap3,
                tap_out => tap_out
        );
  process
   begin
                clk <= '0';
                wait for 50 ns;
                clk <= '1';
                wait for 50 ns;
   end process;
process
   begin
                wait for 125 ns;
                res <= '0';
   end process;
tb : process
         begin
                wait for 100 ns;
                X_in <= "1011";
                wait for 100 ns;
                X_in <= "0011";
                wait for 100 ns;
                X_in <= "0001";
                wait for 100 ns;
                X_in <= "0000";
        wait;
        END process;
END;
```

Пример 3. Тест систолического КИХ-фильтра на языке VHDL (test_poly_syst_main.vhd)

После компиляции в САПР Quartus II при нажатии кнопки EDA Gate Level Simulation (моделирование на уровне вентилей, временная модель «Slow Model») автоматически должен запуститься симулятор ModelSim-Altera (на рис. 5.16 пиктограмма кнопки отмечена кружком). Возможны два варианта. Рассмотрим первый вариант с созданием «тестбенча». Для этого необходимо откомпилировать файл test_poly_syst_main.vhd с помощью меню Compile симулятора ModelSim. После компиляции в рабочей библиотеке work должны появиться два объекта poly_syst_main и test_poly_syst_main (рис. 5.20). Двойным щелчком мыши по объекту test_poly_syst_main автоматически запускаются различные вспомогательные окна (рис. 5.21).

Ставим курсор в окно Objects, нажимаем на правую кнопку мыши меню Add/To Wave/Signals in Region и в окне Wave появляется список сигналов проекта. Далее целесообразно настроить окно Wave, в котором отображаются временные диаграммы работы фильтра. Выбираем Simulate\Runtime Options. В поле система счисления (Default Radix) нажимаем радиокнопку Decimal, что позволяет перейти от двоичной системы счисления, представленной в тестбенче, к десятичной со знаком. В поле Default Run задаем шаг моделирования 100 ns. Последовательно нажимая на пиктограмму кнопки Run с шагом 100 нс, получим временные диаграммы работы КИХ-фильтра (рис. 5.21). Сравниваем полученные результаты с временными диаграммами на рис. 5.18, убеждаемся в правильности работы систолического КИХ-фильтра на 4 отвода в базисе ПЛИС Cyclone II.

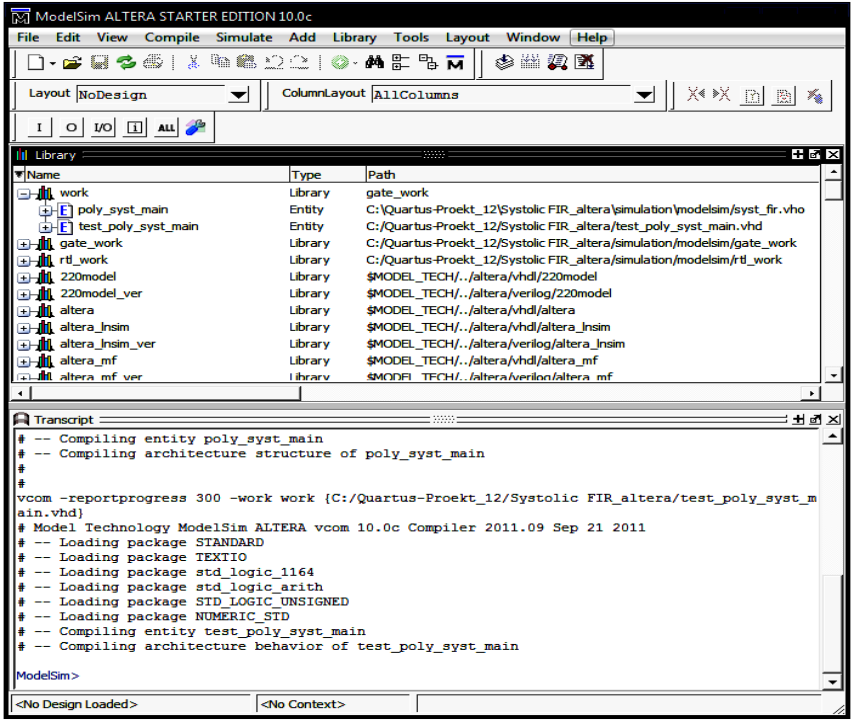


Рис. 5.20. Создается рабочая библиотека work, в которую помещаются два объекта poly_syst_main и test_poly_syst_main

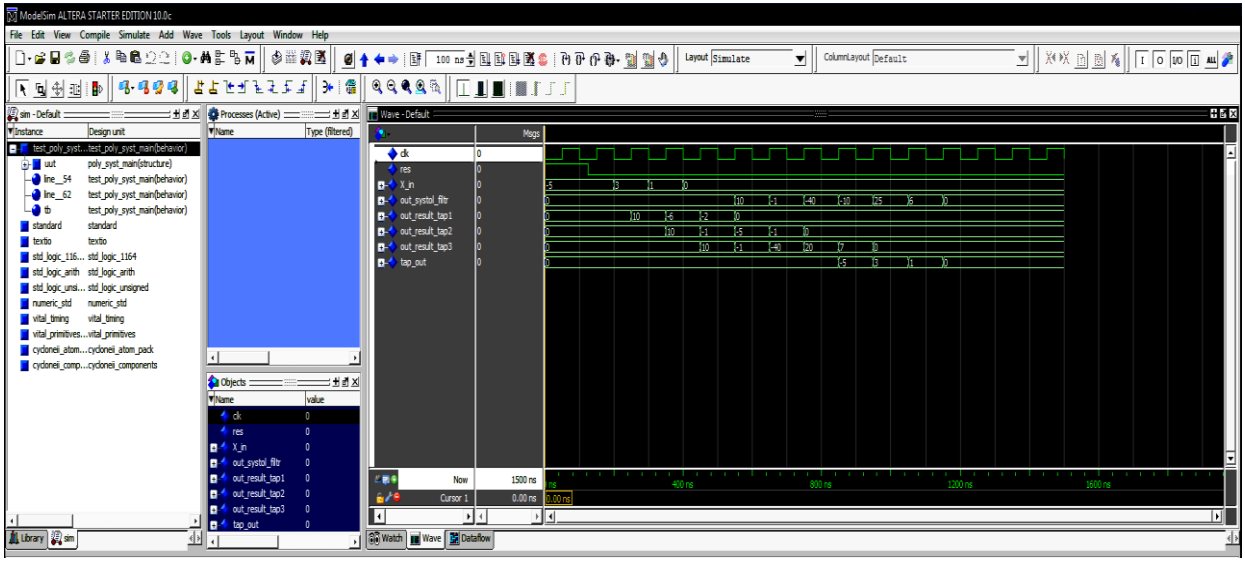


Рис. 5.21. Временные диаграммы работы систолического КИХ-фильтра на 4 отвода с однотипными процессорными элементами в симуляторе ModelSim-Altera версии 10.c

Рассмотрим второй вариант без использования тестбенча. Для этого будем использовать файл poly_syst_main.vhd, а задание на моделирование сформируем непосредственно в векторном редакторе (окно Wave) с помощью специальных инструментов Clock и Force. Двойным щелчком мыши по объекту poly_syst_main автоматически запускаются различные окна. Ставим курсор в окно Objects, как и в первом варианте, нажимаем на правую кнопку мыши меню Add/To Wave/Selected Signals и выбираем только интересующие нас сигналы.

В окне Wave выбираем синхросигнал clk и нажимаем на правую кнопку мыши, выбираем меню Clock. В окне Define Clock в полях задается период синхросигнала – 100 нс, коэффициент заполнения – 50, уровни логической единицы и нуля, радиокнопкой выбираем активным передний фронт синхросигнала (рис. 5.22).

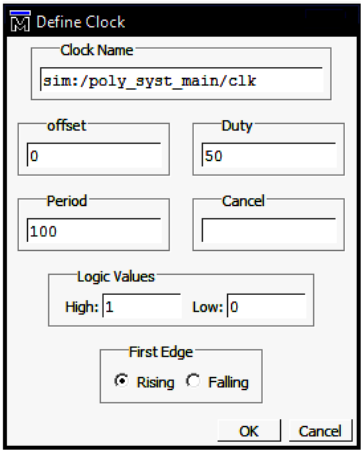


Рис. 5.22. Окно настройки тактового синхросигнала clk

Сигнал асинхронного сброса res (активный – высокий уровень) внутренних регистров процессорных элементов зададим равным нулю с помощью меню Force (действие над сигналом). Радиокнопкой отмечаем вид действия над сигналом

Freeze («замороженный»), в поле Value зададим логический ноль, т. е. на всем временном промежутке моделирования сигнал сброса res будет неактивным (рис. 5.23).

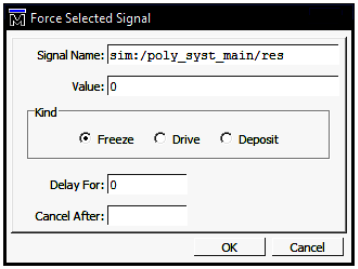


Рис. 5.23. Настройка сигнала сброса res

Далее выбираем сигнал X и с помощью меню Force задаем десятичное число -5 (рис. 5.24). Затем нажимаем на пиктограмму кнопки Run для осуществления моделирования с шагом 100 ns. Далее будем изменять только значения, поступающие на вход X с помощью меню Force, и последовательно нажимать на кнопку Run до получения нужных откликов (пример 4 и рис. 5.25). Задание для моделирования (пример 4) можно использовать, повторно сохранив в текстовый файл.

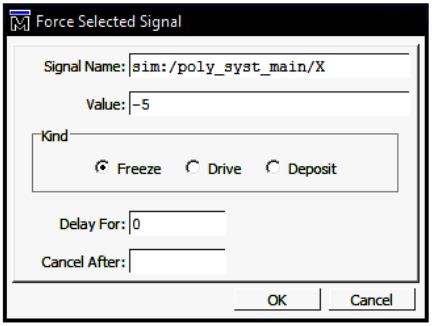


Рис. 5.24. Настройка сигнала X

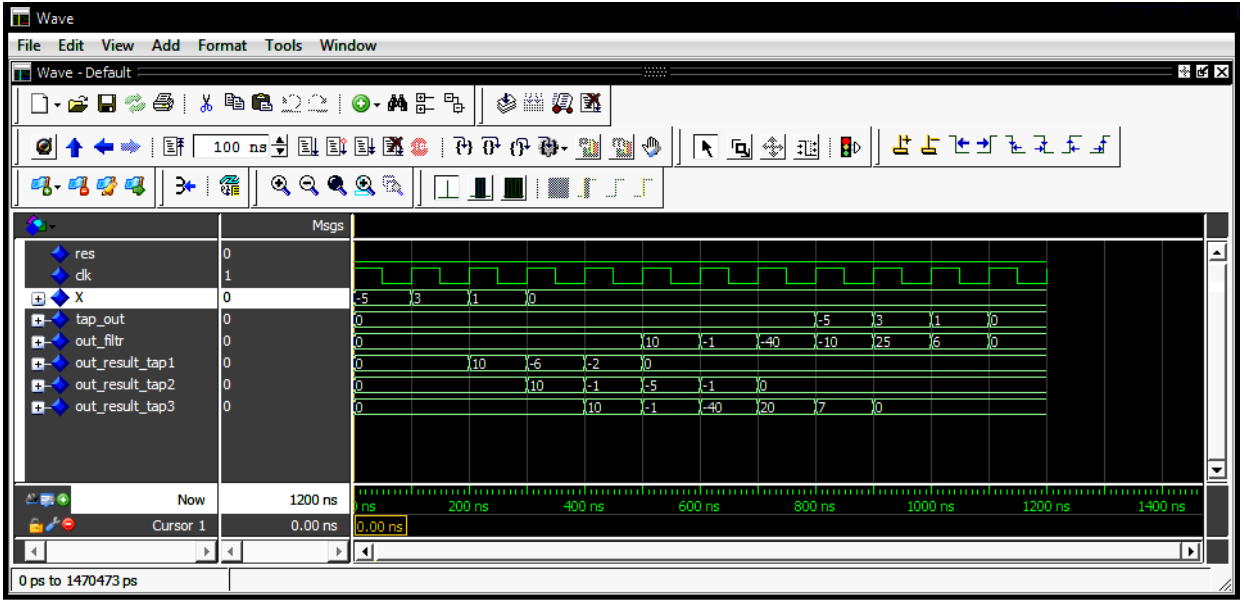


Рис. 5.25. Окно Wave с результатами моделирования КИХ-фильтра

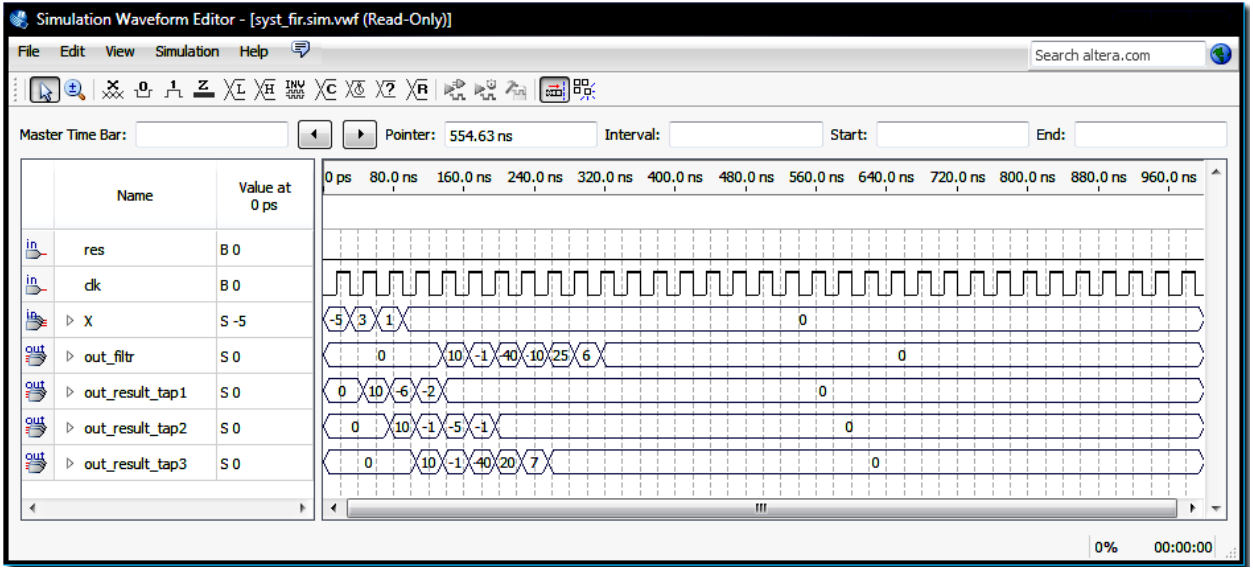


Рис. 5.26. Временные диаграммы работы систолического КИХ-фильтра на четыре отвода с однотипными процессорными элементами в САПР ПЛИС Quartus II версии 13.0

```
force -freeze sim:/poly_syst_main/res 0 0
force -freeze sim:/poly_syst_main/clk 1 0, 0 {50000 ps} -r {100 ns}
force -freeze sim:/poly_syst_main/X -5 0
run
force -freeze sim:/poly_syst_main/X 3 0
run
force -freeze sim:/poly_syst_main/X 1 0
run
force -freeze sim:/poly_syst_main/X 0 0
```

Пример 4. Задание для моделирования из окна Transcript

В версии 13.0 САПР ПЛИС Quartus реализован встроенный векторный редактор с собственной системой моделирования.

Создать векторный файл можно с помощью меню File/New/Verification/Debugging Files/University Program VWF. Запуск моделирования осуществляется непосредственно из окна Simulation Waveform Editor с помощью меню Simulation/Run Functional Simulation.

На рис. 5.26 показаны временные диаграммы работы систолического КИХ-фильтра на 4 отвода в САПР ПЛИС Quartus II версии 13.0. Проект размещен в ПЛИС серии MAX II.

Использование текстового сценария на языке VHDL совместно с симулятором ModelSim-Altera Free позволяет пользователю отлаживать сложные проекты в кратчайшие сроки.

5.3. Пример проектирования систолических КИХ-фильтров в базисе ПЛИС с применением генератора параметризированных ядер XLogiCORE IP и функции FIR Compiler v6.3

Рассмотрим пример проектирования КИХ-фильтра в базисе ПЛИС фирмы Xilinx с использованием САПР ISE Design Suite версии 14.2. Для ускорения процесса разработки проекта КИХ-фильтра воспользуемся генератором параметризированных ядер XLogiCORE IP и функцией FIR Compiler v6.3. Выберем бюджетную ПЛИС Spartan-6 XC6SLX4 с поддержкой протокола AXI, содержащую 8 ЦОС-блоков DSP48A1, располагающихся в двух секциях по четыре в каждой.

На рис. 5.27 показан проект КИХ-фильтра в САПР ПЛИС Xilinx ISE 14.2 с использованием генератора параметризированных ядер XLogiCORE IP FIR Compiler v6.3.

Настройка функции FIR Compiler v6.3 осуществляется в несколько шагов. Задаются варианты считывания значений коэффициентов КИХ-фильтра: из текстового файла (coe-файл) или представление в виде вектора значений, а также параметры спецификации фильтра (рис. 5.28).

По известным коэффициентам происходит построение АЧХ КИХ-фильтра в автоматическом режиме. Задаются границы полосы пропускания и задерживания (подавления), неравномерность АЧХ в полосе пропускания и минимальное затухание в полосе задерживания.

Выбирается одноканальная структура фильтра типа Single-Rate FIR (входная частота дискретизации равна выходной частоте дискретизации). Частота дискретизации определяется как fclk/N для несимметричного и как fclk/N+1 для симметричного фильтра, где fclk – частота тактирования ядра фильтра; N – разрядность входной шины данных (точность представления входных значений подлежащих

фильтрации). Частота тактирования ядра фильтра установлена в 250 МГц, а входная частота дискретизации – 50 МГц.

Проектирование КИХ-фильтра осуществляется в формате с фиксированной запятой. На рис. 5.28 показан учет эффектов квантования. Коэффициенты фильтра несимметричные, целочисленные со знаком $C_0 = -2$, $C_1 = -1$, $C_2 = 7$ и $C_3 = 6$, разрядность представления коэффициентов – 4 бита. Предполагаем, что на вход фильтра поступают только целые значения, как со знаком, так и без, например –5, 3, 1, 0. Разрядность представления значений входного сигнала подлежащего фильтрации – четыре бита, профильтрованного сигнала – 8 бит. Под дробную часть числа в обоих случаях отводим 0 бит (Input Data Fractional Bits и Output Fractional Bits) (рис. 5.29).

На рис. 5.30 показан выбор структуры фильтра. Используем прямую форму систолического фильтра, в котором операции умножения и сложения выполняются параллельно с конвейеризацией (рис. 5.31). В каждой секции доступны 4 ЦОС-блока, располагающихся в столбец, а для реализации КИХ-фильтра требуется 1 ЦОС-блок.

Применение систолических КИХ-фильтров в проектах пользователя позволяет существенно уменьшить число используемых ресурсов и повысить быстродействие. Поддержка систолических структур также обеспечивается мегафункцией (ALTMULT_ADD) САПР Altera Quartus II для работы с ЦОС-блоками ПЛИС серий Cyclon V, Arria V и Stratix V.

Также широко распространена на практике транспонированная реализация дискретного фильтра (direct transposed form II). Транспонированная схема позволяет эффективно распараллелить вычисления. Функция FIR Compiler v6.3 поддерживает такие структуры фильтров.

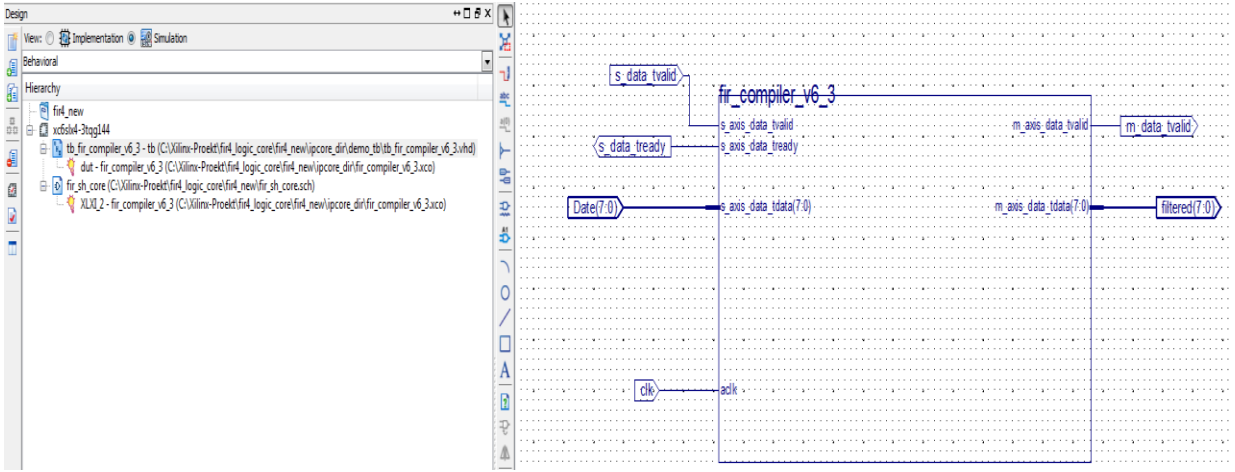


Рис. 5.27. Проект КИХ-фильтра в САПР ПЛИС Xilinx ISE 14.2 с использованием генератора параметризированных ядер XLogiCORE IP FIR Compiler v6.3. Верхний уровень иерархии – схемный файл (sch-файл)

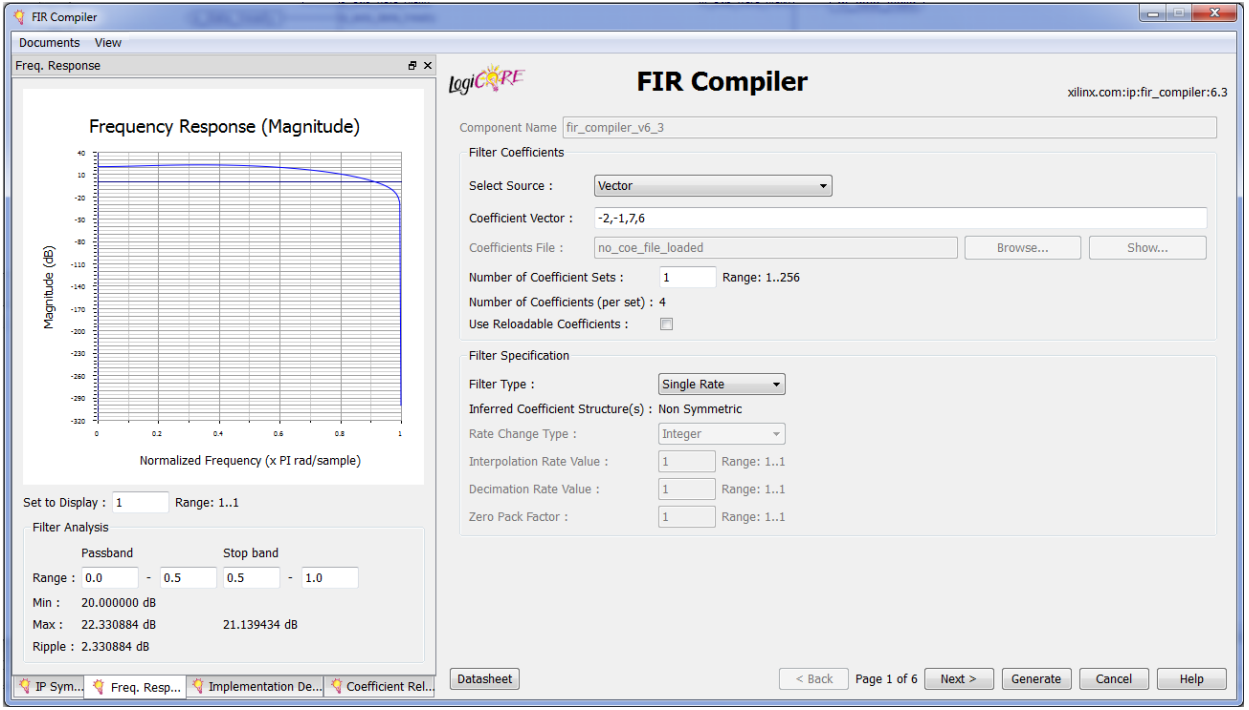


Рис. 5.28. Настройки функции FIR Compiler

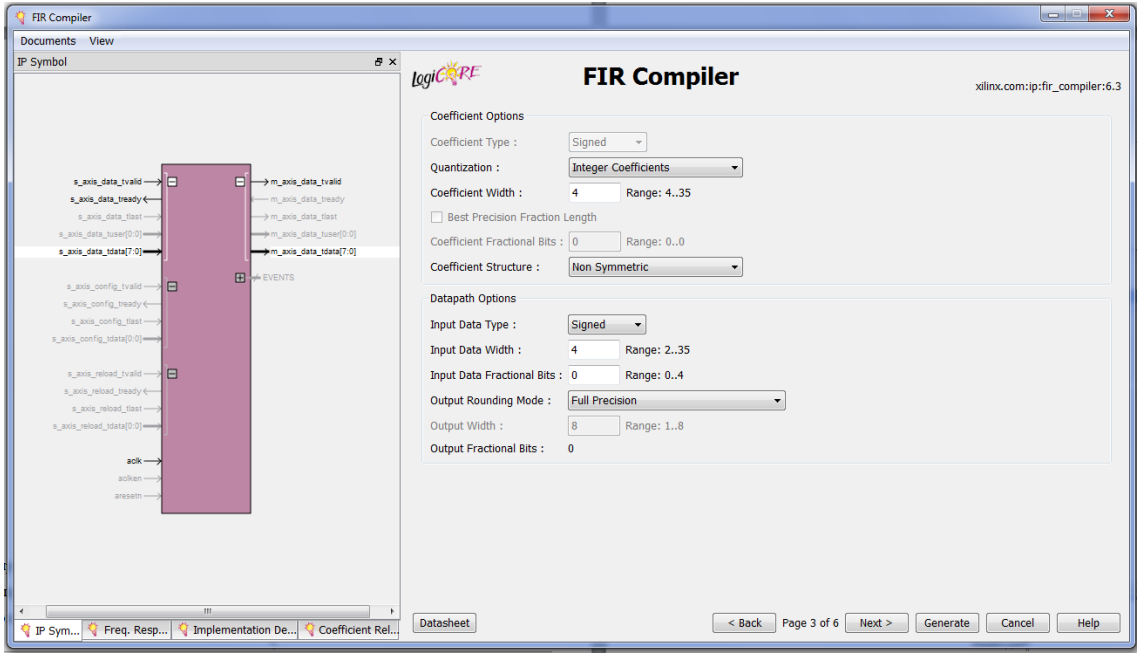
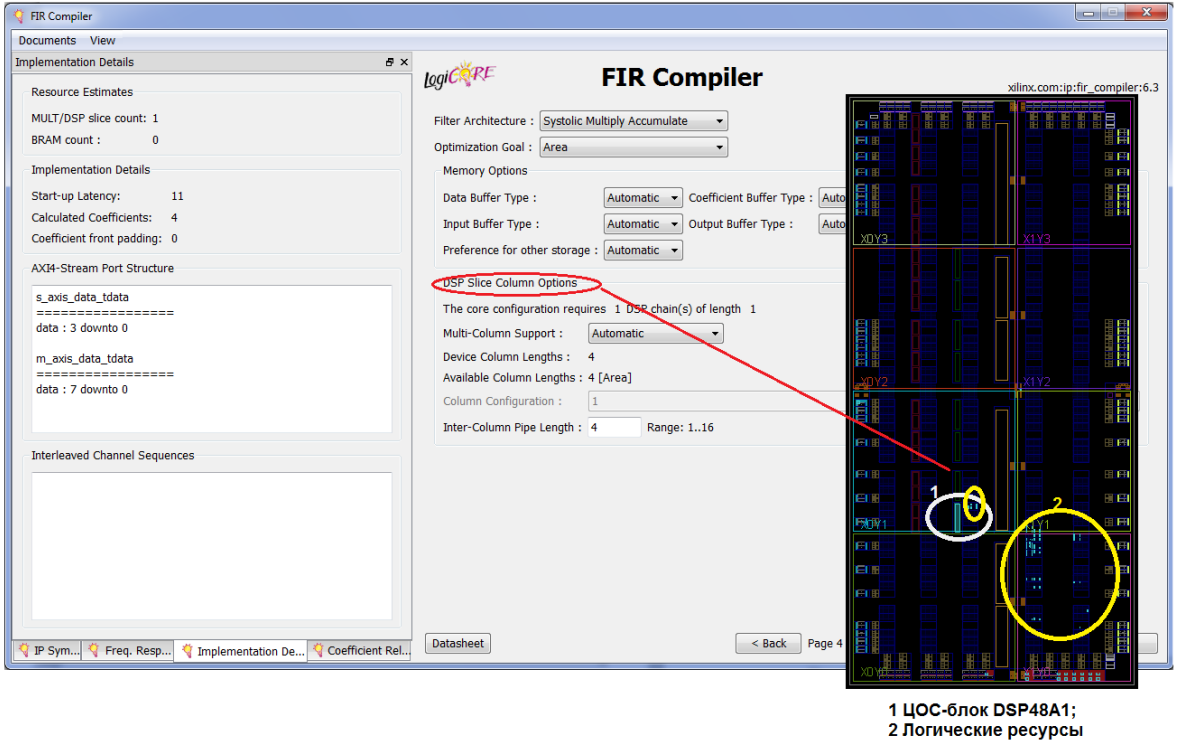


Рис. 5.29. Учет эффектов квантования коэффициентов КИХ-фильтра, точности представления входных и выходных значений (сигналов). Опции, позволяющие настроить форматы представления коэффициентов и входных/выходных значений фильтра



1 ЦОС-блок DSP48A1;
2 Логические ресурсы

Рис. 5.30. Выбор архитектуры фильтра и настройка секций ЦОС-блоков

Они получили название Transpose Multiply-Accumulate architecture (транспонированная структура, основанная на операциях умножения и накопления).

При реализации фильтра в транспонированной форме можно одновременно выполнять все операции умножения, но для получения результата фильтрации необходимо дождаться окончания выполнения всех операций сложения.

Для симуляции проекта в автоматическом режиме необходимо сгенерировать тестбенч (файл теста на языке VHDL для заданий значений входных сигналов или Test Bench-файл). На рис. 5.32 показан симулятор ISim САПР ПЛИС Xilinx ISE 14.2. Демонстрируется прохождение единичного импульса по структуре фильтра (импульсная характеристика КИХ-фильтра на 4 отвода, полученная с использованием тестбенча, сгенерированного в автоматическом режиме). Латентность фильтра 11 тактов синхрочастоты. Для размещения проекта в базис ПЛИС XC6SLX4 требуются 48 триггеров, тактируемых фронтом синхросигнала из общих логических ресурсов ПЛИС, и один ЦОС-блок DSP48A1.

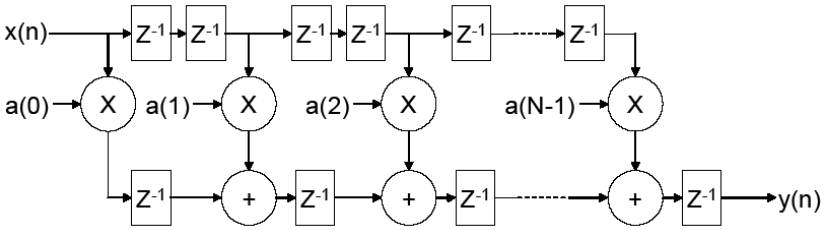


Рис. 5.31. Прямая форма систолического КИХ-фильтра с конвейеризацией

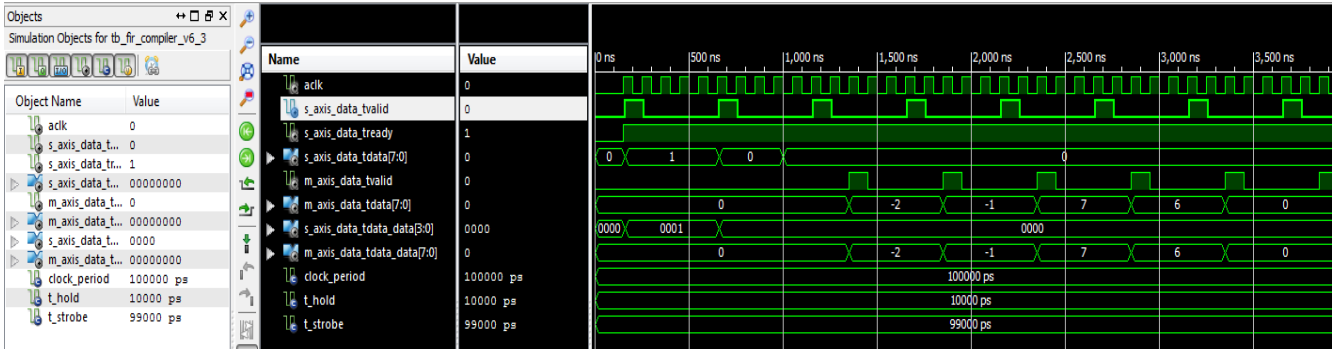


Рис. 5.32. Симулятор ISim САПР ПЛИС Xilinx ISE 14.2. Импульсная характеристика КИХ-фильтра на 4 отвода, полученная с использованием тестбенча, сгенерированного в автоматическом режиме

5.4 Пример проектирования транспонированных КИХ-фильтров в базисе ПЛИС с использованием САПР Quartus II

Рассмотрим пример проектирования КИХ-фильтра с использованием транспонированной формы, которую можно рассматривать как разновидность параллельного фильтра.

Уравнение фильтрации запишем в следующем виде: $y = C_0x_0 + C_1x_1 + C_2x_2 + C_3x_3$. Предположим что, коэффициенты фильтра известны $C_0 = -2$, $C_1 = -1$, $C_2 = 7$ и $C_3 = 6$.

На рис. 5.33 показана транспонированная реализация дискретного фильтра (direct transposed form II) на четыре отвода. Транспонированная схема позволяет эффективно распараллелить вычисления и поэтому наиболее часто применяется. Например, в САПР ПЛИС Xilinx ISE 14.4 функция FIR Compiler v6.3, входящая в состав генератора параметризированных ядер XLogiCORE IP, поддерживает такие структуры фильтров (рис. 5.34). Они получили название Transpose Multiply-Accumulate architecture (транспонированная структура, основанная на операциях умножения и накопления).

При реализации фильтра в транспонированной форме можно одновременно выполнять все операции умножения, но для получения результата фильтрации необходимо дождаться окончания выполнения всех операций сложения. Такие фильтры в технической документации фирмы Xilinx получили название Transpose Multiply-Accumulate (транспонированная реализация, основанная на операции умножения и накопления). Они предполагают использование аппаратных умножителей, входящих в состав ЦОС-блоков ПЛИС.

Мегафункция fir_compiler_v13_1 САПР ПЛИС Altera Quartus II поддерживает структуры фильтров на последовательной и параллельной распределенной арифметике, модификацию последовательной арифметики

(Multibit Serial Structure, несколько КИХ-фильтров на последовательной распределенной арифметике, объединенные склеивающей логикой) и структуры с использованием аппаратных умножителей (Multicycle variable, MCV).

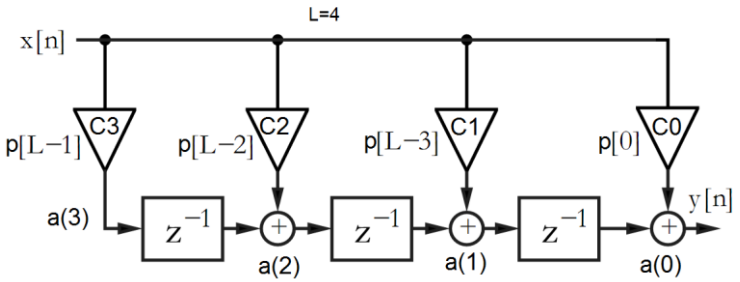


Рис. 5.33. Транспонированная форма КИХ-фильтра на четыре отвода

Генератор параметризированных ядер Xilinx XLogiCORE IP FIR Compiler v5.0 предлагает на выбор три структуры фильтров: прямую форму систолического КИХ-фильтра, в котором операции умножения и сложения выполняются параллельно с конвейеризацией; транспонированную и на распределенной арифметике.

В отличие от мегафункции Altera fir_compiler_v13_1 функция Xilinx FIR Compiler v5.0 в автоматическом режиме определяет какая структура фильтра на распределенной арифметике наиболее эффективна (последовательная или параллельная) для конкретной серии ПЛИС в зависимости от соотношения частоты взятия входных отчетов fs и частоты тактирования системы fclk.

Структуры КИХ-фильтров на основе распределенной арифметики обладают рекордным быстродействием, которое не снижается с ростом числа отводов. Такие решения особенно эффективны в низкобюджетных сериях ПЛИС, где существует недостаточное число встроенных аппаратных ЦОС-блоков.

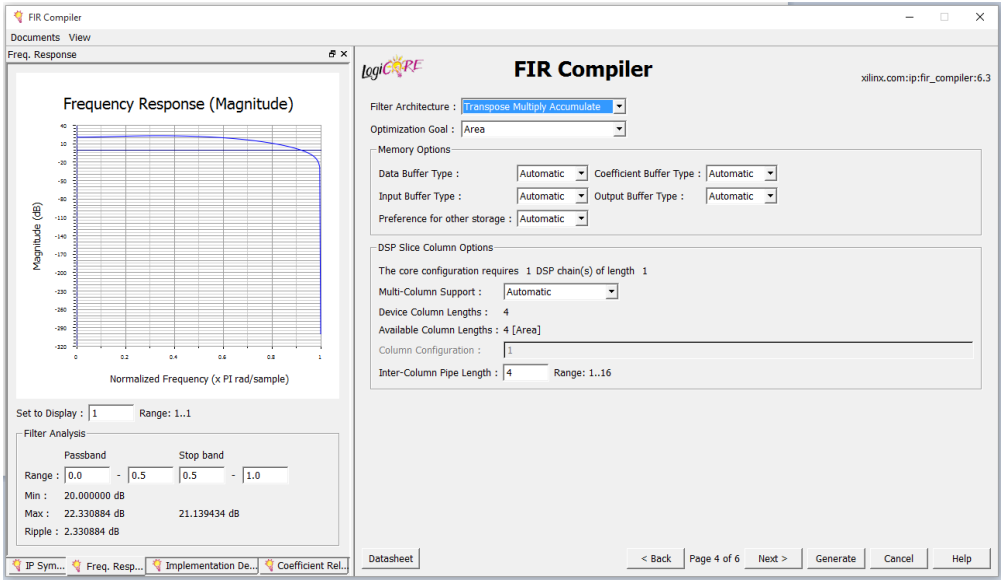


Рис. 5.34. Генератор параметризированных ядер XLogiCORE IP FIR Compiler v6.3 САПР ПЛИС Xilinx ISE 14.4. Выбор архитектуры фильтра и настройка секций ЦОС-блоков

Рассмотрим КИХ-фильтр транспонированной формы с перегружаемыми коэффициентами (обозначения сигналов, используемых в проекте, указаны на рис. 5.33). Первый оператор process используется для загрузки коэффициентов фильтра и дискретных значений сигнала. При ***Load_x*** равным логическому нулю происходит последовательная загрузка коэффициентов фильтра в регистр ***c*** (далее, последующий сдвиг), а при ***Load_x*** равным логической единице происходит загрузка сигнала подлежащего фильтрации в регистр ***x***.

Второй оператор process формирует сумму произведений (SOP). Для первого отвода фильтра устанавливается y<=a(0). Третий оператор process формирует произведения (***p***). Входной сигнал ***x_in*** и коэффициенты фильтра ***c_in*** представляются с 9-битной точностью (W1), произведения (сигнал ***p***, W2=2*W1) вычисляются с точностью 18 бит, суммирование (сигнал ***a***, W3=W2+log2(L), где L=4, L – число отводов фильтра) осуществляется с точностью 19–бит с учетом возможного переполнения по формуле $9+9+\log_2(4)-1=19$. Внутренние сигналы ***c, p*** и ***a*** представляют собой матрицы разной размерности.

На рис. 5.35 показан проект КИХ-фильтра на четыре отвода в САПР Quartus II ver.13.1 с использованием кода языка VHDL (пример 1), а на рис. 5.36 импульсная характеристика. Последовательно загружаются коэффициенты –2, –1, 7 и 6, далее подается единичный импульс на вход data и на следующем такте синхроимпульса на выходе фильтра наблюдаем импульсную характеристику.

```
LIBRARY ieee; -- Using predefined packages
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_signed.ALL;
-- ------------------------------------------------------
ENTITY fir_gen IS ------> Interface
GENERIC (W1 : INTEGER := 9; -- Input bit width
```

```
W2 : INTEGER := 18;-- Multiplier bit width 2*W1
W3 : INTEGER := 19;-- Adder width = W2+log2(L)-1
W4 : INTEGER := 11;-- Output bit width
L : INTEGER := 4 -- Filter length
);
PORT ( clk : IN STD_LOGIC; -- System clock
reset : IN STD_LOGIC; -- Asynchron reset
Load_x : IN STD_LOGIC; -- Load/run switch
x_in : IN STD_LOGIC_VECTOR(W1-1 DOWNTO 0);
-- System input
c_in : IN STD_LOGIC_VECTOR(W1-1 DOWNTO 0);
-- Coefficient data input
y_out : OUT STD_LOGIC_VECTOR(W4-1 DOWNTO 0));
END fir_gen; -- System output
-- ------------------------------------------------------
ARCHITECTURE fpga OF fir_gen IS
SUBTYPE SLVW1 IS STD_LOGIC_VECTOR(W1-1 DOWNTO 0);
SUBTYPE SLVW2 IS STD_LOGIC_VECTOR(W2-1 DOWNTO 0);
SUBTYPE SLVW3 IS STD_LOGIC_VECTOR(W3-1 DOWNTO 0);
TYPE A0_L1SLVW1 IS ARRAY (0 TO L-1) OF SLVW1;
TYPE A0_L1SLVW2 IS ARRAY (0 TO L-1) OF SLVW2;
TYPE A0_L1SLVW3 IS ARRAY (0 TO L-1) OF SLVW3;
SIGNAL x : SLVW1;
SIGNAL y : SLVW3;
SIGNAL c : A0_L1SLVW1 ; -- Coefficient array
SIGNAL p : A0_L1SLVW2 ; -- Product array
SIGNAL a : A0_L1SLVW3 ; -- Adder array
BEGIN
Load: PROCESS(clk, reset, c_in, c, x_in)
BEGIN ------> Load data or coefficients
IF reset = '1' THEN -- clear data and coefficients reg.
x <= (OTHERS => '0');
FOR K IN 0 TO L-1 LOOP
c(K) <= (OTHERS => '0');
END LOOP;
ELSIF rising_edge(clk) THEN
IF Load_x = '0' THEN
c(L-1) <= c_in; -- Store coefficient in register
FOR I IN L-2 DOWNTO 0 LOOP -- Coefficients shift one
```

```
c(I) <= c(I+1);
END LOOP;
ELSE
x <= x_in; -- Get one data sample at a time
END IF;
END IF;
END PROCESS Load;
SOP: PROCESS (clk, reset, a, p)-- Compute sum-of-products
BEGIN
IF reset = '1' THEN -- clear tap registers
FOR K IN 0 TO L-1 LOOP
a(K) <= (OTHERS => '0');
END LOOP;
ELSIF rising_edge(clk) THEN
FOR I IN 0 TO L-2 LOOP -- Compute the transposed
a(I) <= (p(I)(W2-1) & p(I)) + a(I+1); -- filter adds
END LOOP;
a(3) <= p(3)(W2-1) & p(3); -- First TAP has
END IF; -- only a register
y <= a(0);
END PROCESS SOP;
-- Instantiate L multipliers
MulGen: FOR I IN 0 TO L-1 GENERATE
p(i) <= c(i) * x;
END GENERATE;
y_out <= y(W4-1 DOWNTO 0);
END fpga;
```

Пример 1. VHDL-код КИХ-фильтра транспонированной формы на четыре отвода

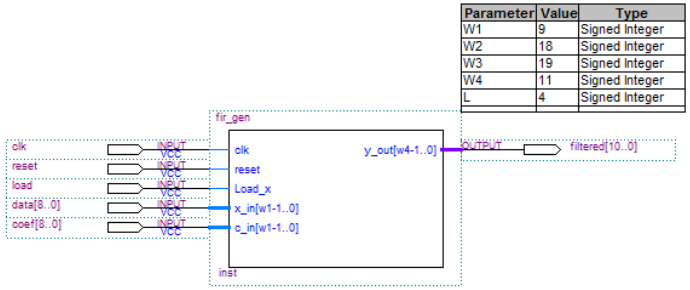


Рис. 5.35. Проект КИХ-фильтра на четыре отвода в САПР Quartus II ver.13.1 с использованием VHDL-кода (пример 1)

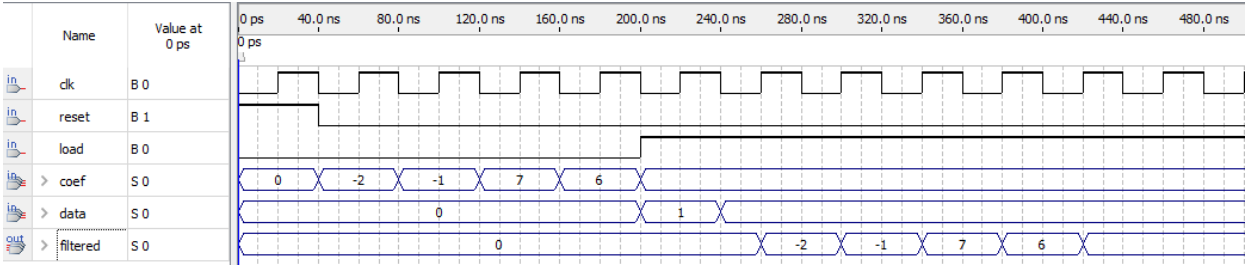


Рис. 5.36. Функциональное моделирование (импульсная характеристика КИХ-фильтра) с использованием встроенного векторного редактора

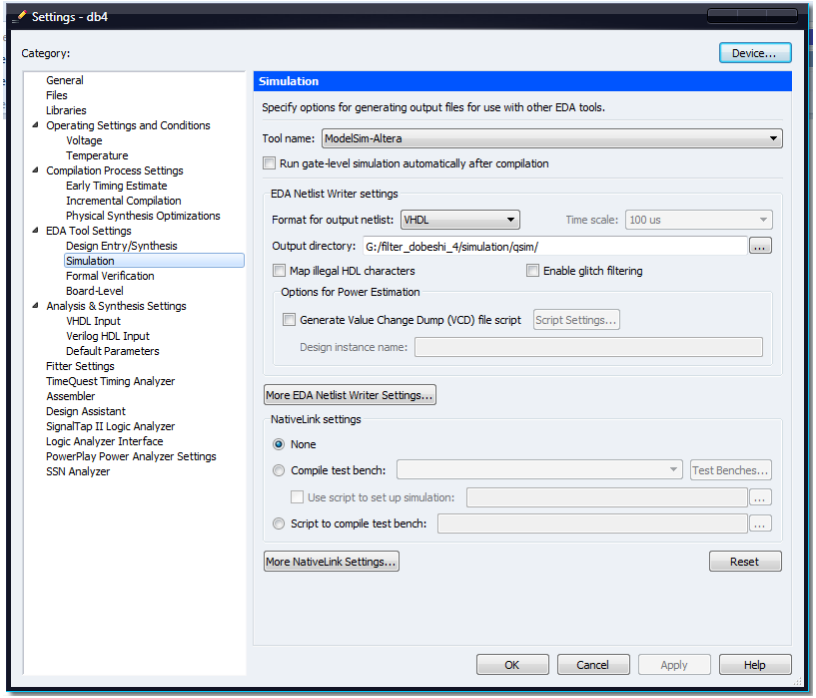


Рис. 5.37. Настройки САПР Quartus II, меню EDA Tool Settings/Simulation

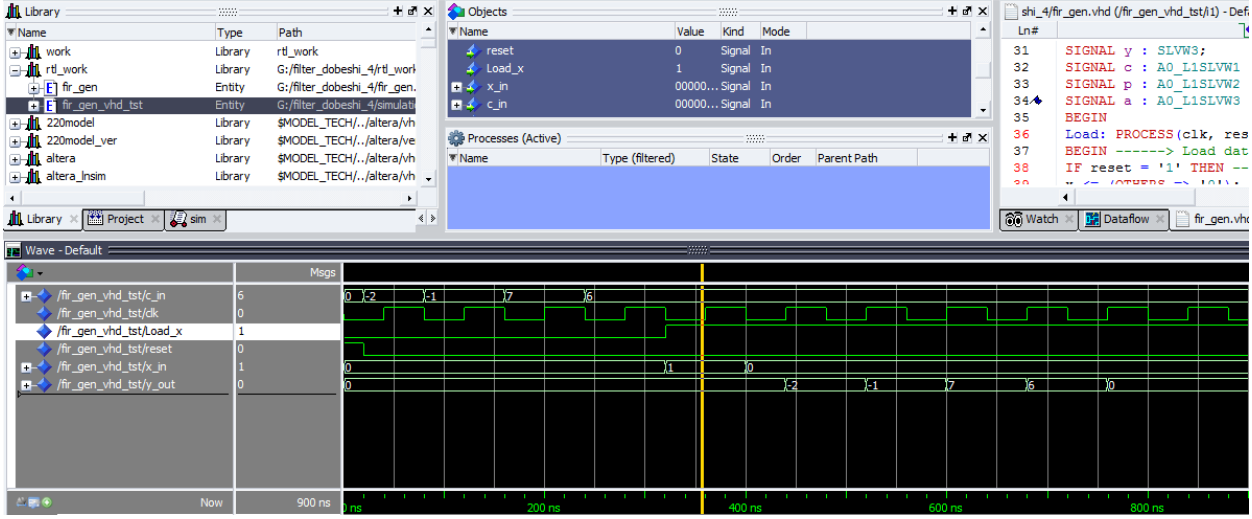


Рис. 5.38. Функциональное моделирование (импульсная характеристика) в системе цифрового моделирования ModelSim Altera

В случае если применяется САПР Quartus II ver.12.1 без поддержки использования встроенного векторного редактора или симуляция на уровне регистровых передач (RTL-симуляция) с использованием инструмента ModelSim-Altera, то необходимо предварительно разработать испытательный стенд на языке VHDL.

Для этого необходимо зайти в меню Processing/Start/Start Test Bench Template Writer и создать шаблон испытательного стенда. В нашем случае будет создан файл с именем проекта fir_gen с расширением vht, который в дальнейшем должен быть отредактирован в ручном режиме.

На рис. 5.37 показаны настройки САПР Quartus II меню EDA Tool Settings/Simulation для RTL-симуляции. Выбирается инструмент симуляции (ModelSim). Указывается выходной формат нетлиста (VHDL) для симуляции и каталог.

Пример 2 демонстрирует испытательный стенд для функционального моделирования импульсной характеристики КИХ-фильтра в системе цифрового моделирования ModelSim Altera. На рис. 5.38 показано функциональное моделирование в системе ModelSim Altera.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY fir_gen_vhd_tst IS
END fir_gen_vhd_tst;
ARCHITECTURE fir_gen_arch OF fir_gen_vhd_tst IS
-- constants
-- signals
SIGNAL c_in : STD_LOGIC_VECTOR(8 DOWNTO 0);
SIGNAL clk : STD_LOGIC;
SIGNAL Load_x : STD_LOGIC:= '0';
SIGNAL reset : STD_LOGIC:= '1';
SIGNAL x_in : STD_LOGIC_VECTOR(8 DOWNTO 0);
SIGNAL y_out : STD_LOGIC_VECTOR(10 DOWNTO 0);
COMPONENT fir_gen
        PORT (
        c_in : IN STD_LOGIC_VECTOR(8 DOWNTO 0);
```

```
        clk : IN STD_LOGIC;
        Load_x : IN STD_LOGIC;
        reset : IN STD_LOGIC;
        x_in : IN STD_LOGIC_VECTOR(8 DOWNTO 0);
        y_out : OUT STD_LOGIC_VECTOR(10 DOWNTO 0)
        );
END COMPONENT;
BEGIN
        i1 : fir_gen
        PORT MAP (
-- list connections between master ports and signals
        c_in => c_in,
        clk => clk,
        Load_x => Load_x,
        reset => reset,
        x_in => x_in,
        y_out => y_out
        );
clk_process : PROCESS
BEGIN
                clk <= '0';
                wait for 40 ns;
                clk <= '1';
                wait for 40 ns;
END PROCESS clk_process;
res_process : process
  begin
                wait for 20 ns;
                reset <= '0';
  end process res_process;
ld : PROCESS
BEGIN
  Load_x <= '0';
  wait for 320 ns;
  Load_x <= '1';
wait;
END PROCESS ld;
coef : process
          begin
                c_in <= "000000000";
                wait for 20 ns;
                c_in <= "111111110";
```

```
                wait for 60  ns;
                c_in <= "111111111";
                wait for 80 ns;
                c_in <= "000000111";
                wait for 80 ns;
                c_in <= "000000110";
        wait;
        END process coef;
delta : process
         begin
                x_in <= "000000000";
                wait for 320 ns;
                x_in <= "000000001";
                wait for 80 ns;
               x_in <= "000000000";
        wait;
        END process delta;
END fir_gen_arch;
```

Пример 2. Испытательный стенд для функционального моделирования КИХ-фильтра в системе цифрового моделирования ModelSim Altera

Проект может быть размещен в базис ПЛИС серии Cyclone III EP3C5F256C6 и занимает логических ячеек – 69, аппаратных умножителей с размерностью операндов 9×9 – 4, рабочая частота в наихудшем случае с использованием TimeQuest анализа для модели slow 1100 мВ 85 ^{0}C оценивается величиной 196.7 МГц.

6. ИСПОЛЬЗОВАНИЕ СИСТЕМЫ ВИЗУАЛЬНО-ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ M*ATLAB*/SIMULINK ДЛЯ ПРОЕКТИРОВАНИЯ КИХ-ФИЛЬТРОВ В САПР ISE DESIGN SUITE

6.1. Проектирование КИХ-фильтров в системе Xilinx System Generator САПР ISE Design Suite

System Generator IDS 14.4 – инструмент для разработки и отладки высокопроизводительных систем цифровой обработки сигналов в базисе ПЛИС фирмы Xilinx в системе визуально-имитационного моделирования M*atlab*/Simulink (версия 8.0.0.783 (R2012b)). Программный пакет обеспечивает высокоуровневое представление проекта, абстрагированное от конкретной аппаратной платформы, которое автоматически компилируется в ПЛИС Xilinx. System Generator является частью технологии XtremeDSP фирмы Xilinx.

System Generator сокращает время симуляции проектов за счет hardware-in-the-loop и HDL co-simulation. System Generator автоматически транслирует ЦОС-системы из M*atlab*/Simulink описаний в высокооптимизированные VHDL-описания для ПЛИС Xilinx и создает испытательные стенды. Методология «Hardware-in-the-loop» существенно ускоряет цикл проектирования, поскольку позволяет верифицировать проекты в ПЛИС непосредственно из системы M*atlab*/Simulink. «HDL co-simulation» позволяет пользователям импортировать HDL-код и симулировать всю систему в целом.

Рассмотрим разработку имитационной модели КИХ-фильтра на распределенной арифметике без использования встроенных ЦОС-блоков (тип фильтра – Single-Rate FIR) с применением функционального блока FIR Compiler v5.0, являющегося аналогом функции FIR Compiler v5.0 САПР Xilinx ISE, получаемой с помощью генератора параметризированных ядер XLogiCORE IP (рис. 6.1).

Для верификации функционального блока FIR Compiler v5.0 используются фильтры DF2T (транспонированная реализация дискретного фильтра) и Digital Filter (прямая форма). Блок FIR Compiler v5.0 является параметризированным (рис. 6.2). На рис. 6.2 показаны настройки блока. Согласно настройкам блока реализуется КИХ-фильтр на параллельной распределенной арифметике. Результаты имитационного моделирования представлены на рис. 6.3.

Для представления чисел со знаком в формате с фиксированной запятой Xilinx System Generator использует нотацию FIX, а для беззнаковых – UFIX. Формат FIX можно рассматривать как пару чисел M.N, где M – общее число двоичных разрядов; N – число разрядов дробной части. Входной сигнал представляется в формате FIX_16_8, коэффициенты фильтра – в формате FIX_8_4, профильтрованный сигнал FIX_26_12 (рис. 6.1). С помощью блока System Generator создадим в автоматическом режиме проект фильтра и испытательный стенд. Проект разместим в базис ПЛИС серии Spartan-6 xc6slx4-3tqg144.

На рис. 6.4 и рис. 6.5 показано функциональное моделирование с использованием моделирующей программы, сгенерированной в автоматическом режиме для случая, когда используется функциональный блок FIR Compiler v5.0. Входной сигнал, подлежащий фильтрации, умножается на масштабный множитель 256, а коэффициенты фильтра масштабируются на 16 согласно выбранному формату представления чисел.

Для получения правильного результата фильтрации необходимо предусмотреть деление на 4096. Уравнение фильтрации будет выглядеть следующим образом:

$$y/4096 = (C_0 * 16)(x_0 * 256) + (C_1 * 16)(x_1 * 256) +$$
$$+ (C_2 * 16)(x_2 * 256) + (C_3 * 16)(x_3 * 256).$$

На рис. 6.6 показана имитационная модель КИХ-фильтра на 4 отвода с перегружаемыми коэффициентами с использованием блока FIR Compiler v5.0, а на рис. 6.7 – результаты имитационного моделирования.

Рассмотрим разработку имитационной модели систолического КИХ-фильтра (тип фильтра – Single-Rate FIR) с использованием блока FIR Compiler v6.3, являющегося аналогом функции FIR Compiler v6.3 САПР Xilinx ISE, получаемой с помощью генератора параметризированных ядер XLogiCORE IP (рис. 6.8). На рис. 6.9 показан формат представления входного сигнала и закладка «реализация» блока FIR Compiler 6.3. Входной сигнал представляется в формате FIX_16_8, а коэффициенты фильтра – в формате FIX_4_0. Профильтрованный сигнал представляется с 20-битной точностью. На рис. 6.10 показано имитационное моделирование.

С помощью блока System Generator создадим в автоматическом режиме проект фильтра и испытательный стенд. На рис. 6.11 показано функциональное моделирование с использованием моделирующей программы, сгенерированной в автоматическом режиме (период синхросигнала 100 нс).

Входной сигнал, подлежащий фильтрации умножается на масштабный множитель 256, а коэффициенты фильтра не масштабируются согласно выбранному формату представления чисел. Для получения правильного результата фильтрации необходимо предусмотреть деление на 256. Уравнение фильтрации будет выглядеть следующим образом:

$$y/256 = C_0(x_0 * 256) + C_1(x_1 * 256) + C_2(x_2 * 256) + C_3(x_3 * 256).$$

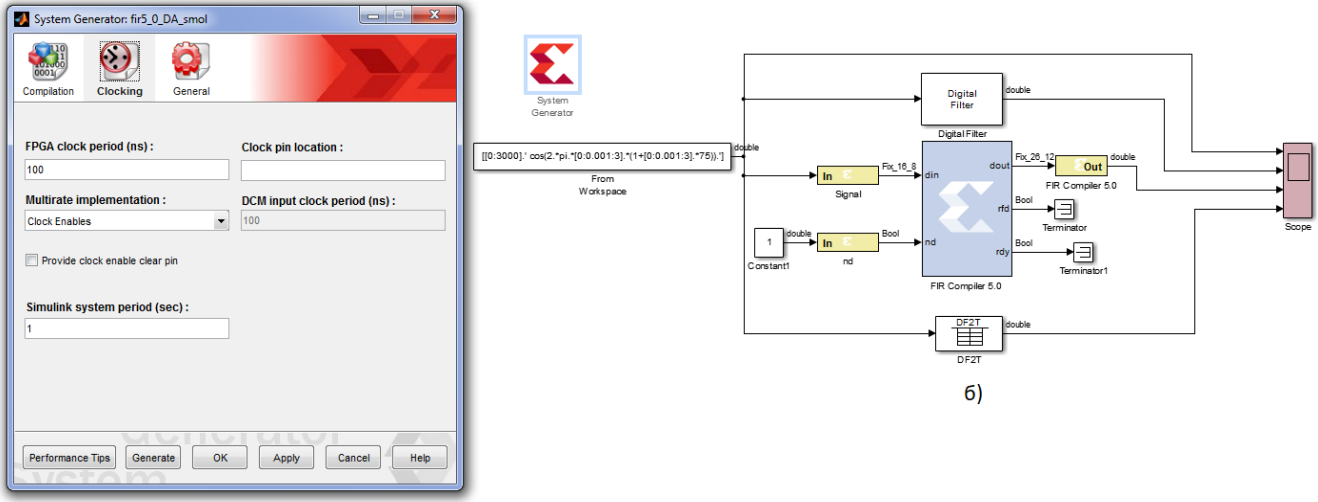
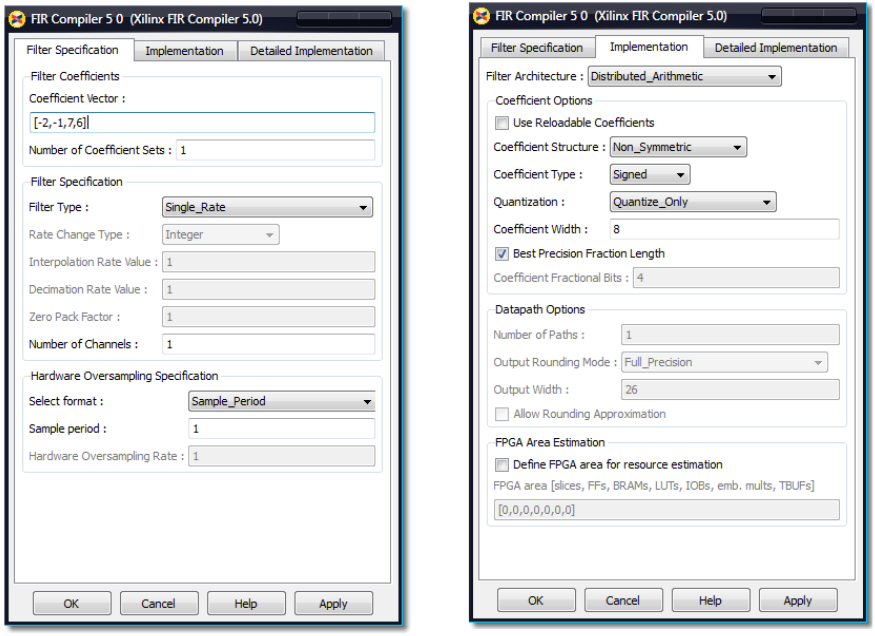


Рис. 6.1. а) Задание частоты тактирования системы (фильтра) в САПР ISE (100 нс) и периода симуляции в Simulink (1 с); б) модель КИХ-фильтра на 4 отвода с использованием блока FIR Compiler v5.0



а) б)

Рис. 6.2. Настройки блока FIR Compiler 5.0: а) закладка “спецификация” фильтра; б) закладка “реализация” фильтра. Коэффициенты фильтра несимметричные, со знаком, квантованные, представлены в формате FIX_8_4

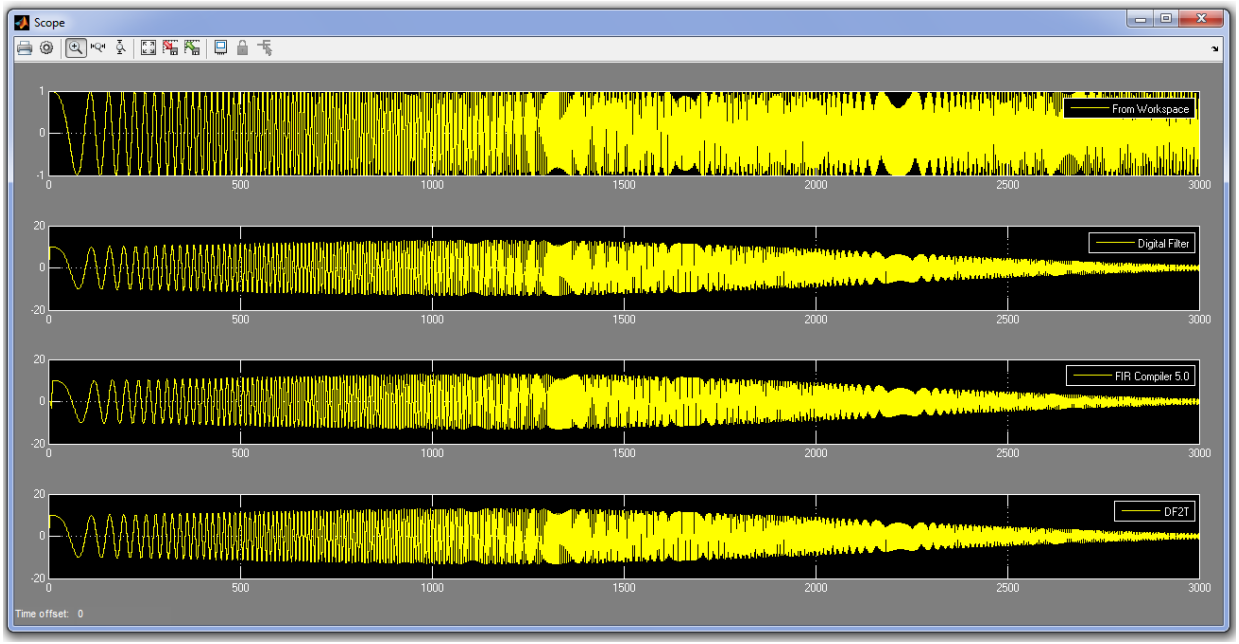


Рис. 6.3. Имитационное моделирование в системе M*atlab*/Simulink КИХ-фильтра на 4 отвода, созданного с помощью блоков Digital Filter, FIR Compiler 5.0 и DF2T

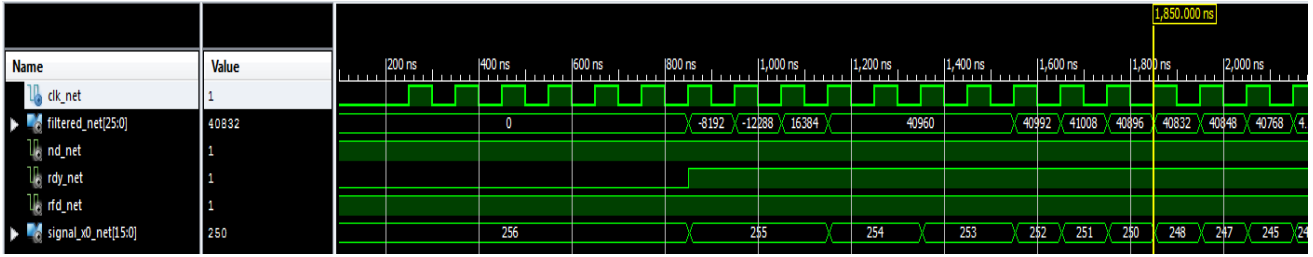


Рис. 6.4. Входной сигнал умножается на масштабный множитель 256, коэффициенты фильтра на 16. Период синхросигнала 100 нс

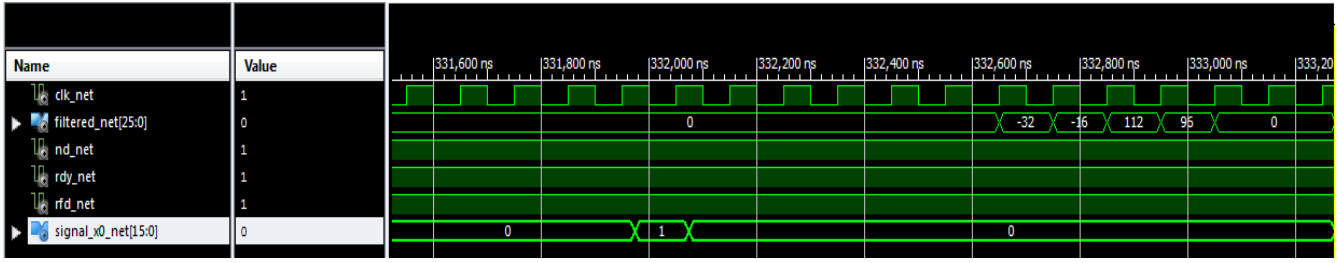


Рис. 6.5. Импульсная характеристика фильтра. Коэффициенты фильтра умножаются на 16

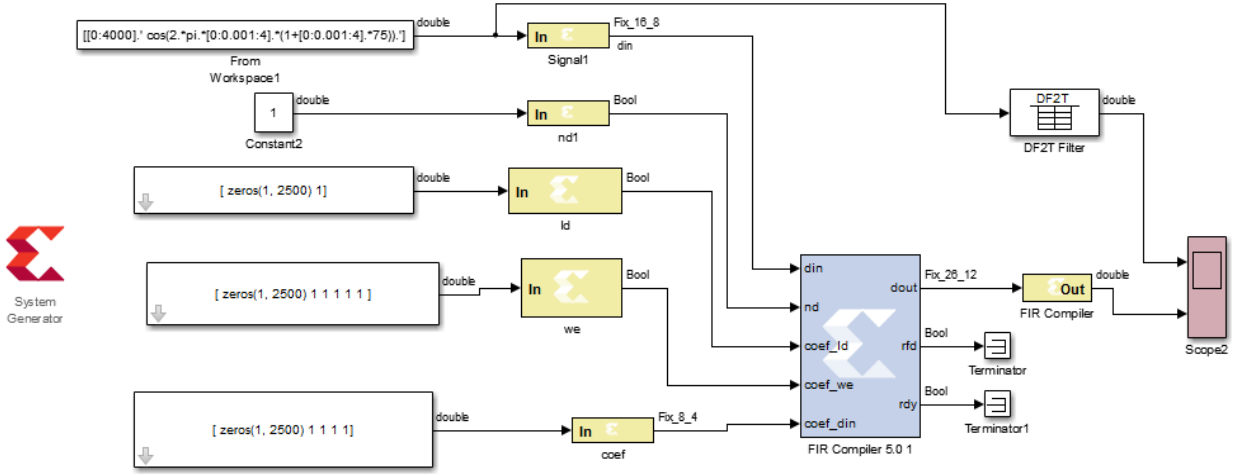


Рис. 6.6. Модель КИХ-фильтра на 4 отвода с перегружаемыми коэффициентами с использованием блока FIR Compiler v5.0

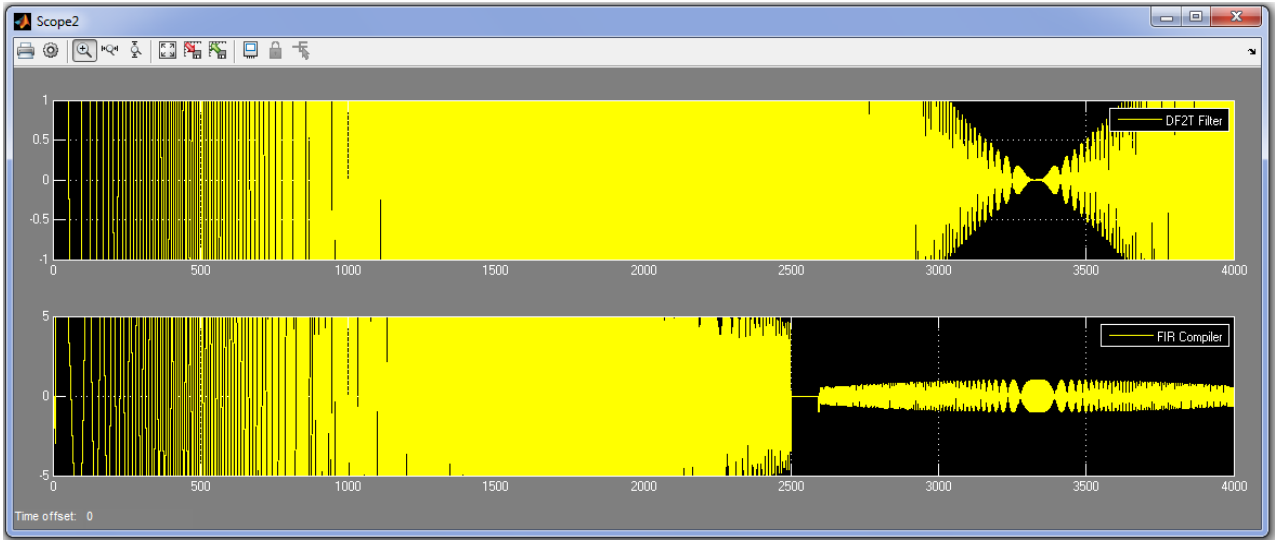


Рис. 6.7. При достижении 2500 отсчета происходит загрузка вектора значений коэффициентов [1 1 1 1] вместо [–2 –1 7 6]

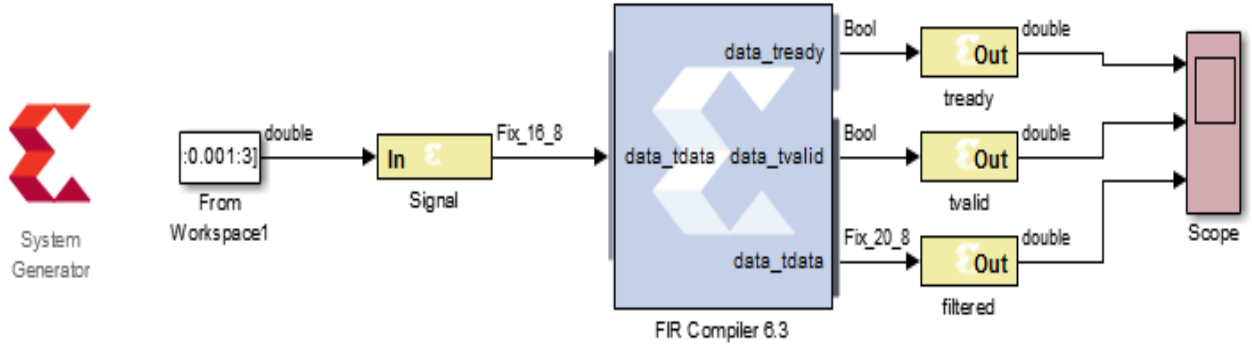
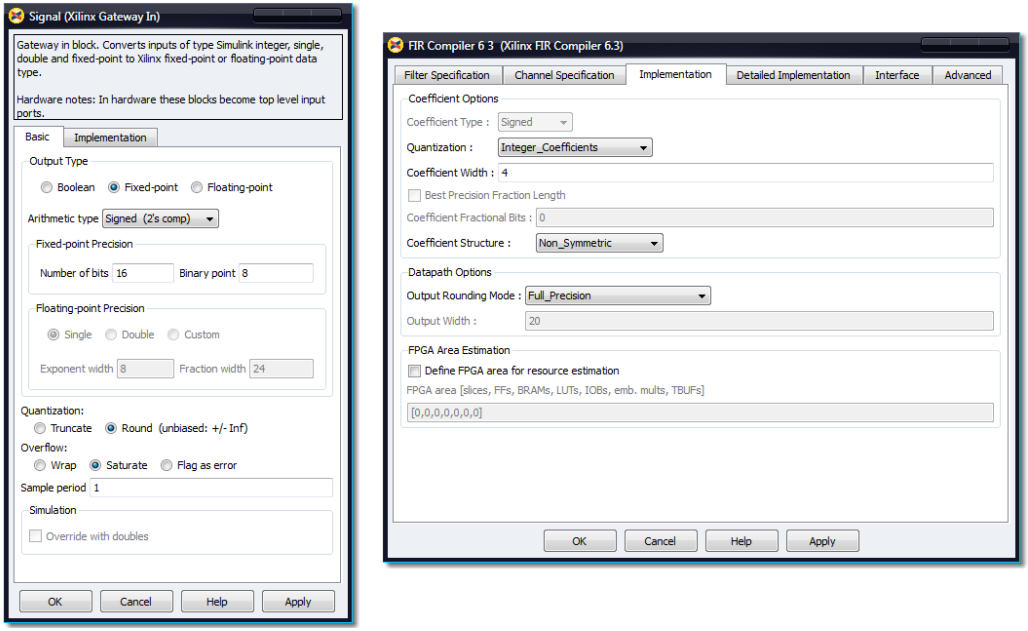


Рис. 6.8. Систолический КИХ-фильтр на 4 отвода с использованием блока FIR Compiler v6.3



а) б)

Рис. 6.9. Настройки блоков: а) входной сигнал представляется в формате FIX_16_8; б) закладка “реализация”, коэффициенты фильтра целые, со знаком, представлены в формате FIX_4_0

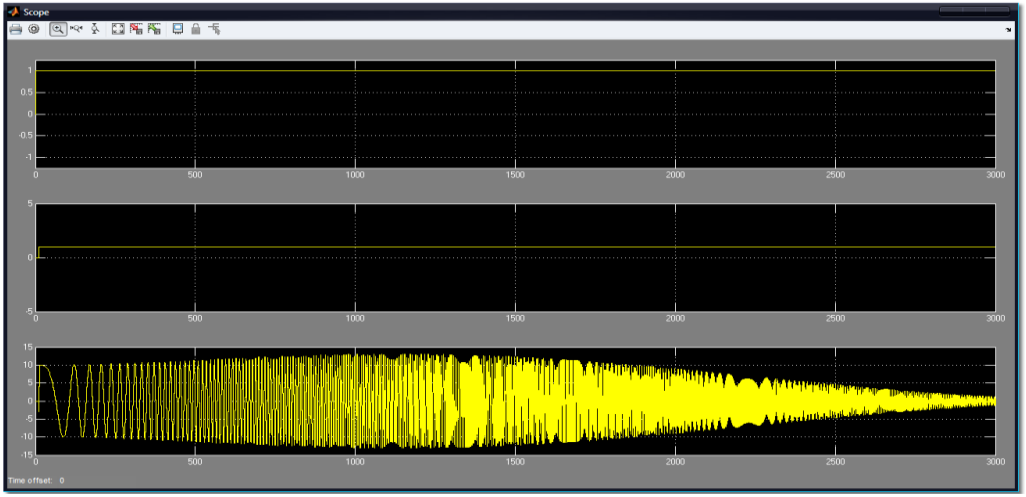


Рис. 6.10. Имитационное моделирование фильтра на 4 отвода, созданного с помощью блока FIR Compiler 6.3

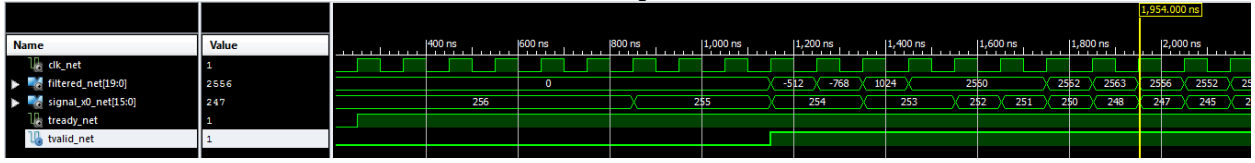


Рис. 6.11. Функциональное моделирование с использованием моделирующей программы на языке VHDL, сгенерированной в автоматическом режиме. Входной сигнал умножается на 256, а коэффициенты фильтра не масштабируются

6.2. Проектирование последовательных КИХ-фильтров со структурой MAC-блоков в системе Xilinx System Generator САПР ISE Design Suite

Рассмотрим проектирование КИХ-фильтра (рис. 6.12) с использованием MAC-блока с большим числом отводов для подавления высокочастотных шумов в аудиосистеме (CD-качество). Такие фильтры получили название MAC-фильтр. Данный пример основан на функциональном блоке n-tap Dual Port Memory MAC FIR Filter из библиотеки Xilinx Reference Blockset /DSP.

Для хранения коэффициентов фильтра и входных отсчетов будем использовать блочную память ПЛИС Xilinx (двухпортовое ОЗУ). Параметры спецификации фильтра следующие: единицы измерения – Гц; частота взятия отчетов $F_S = 44100$ Гц (44.1 кГц); порядок фильтра – автоматический выбор (Minimum Order); граница полосы пропускания – $f_{Pass} = 6000$ Гц (6 кГц); граница полосы задерживания (подавления) – $f_{Stop} = 7725$ Гц (7.725 кГц); неравномерность АЧХ в полосе пропускания – A_{Pass} ($R_p = 1$ дБл); минимальное затухание в полосе задерживания – A_{Stop} ($R_S = 48$ дБл). Осуществим синтез фильтров с равномерными пульсациями АЧХ в полосах пропускания и задерживания методом Ремеза (Equiripple). По заданной технической спецификации синтезируется фильтр с порядком $n = 42$ (импульсная характеристика содержит $n+1$ ненулевых отсчетов) или 43 коэффициента.

На рис. 6.13 показаны настройки функционального блока источника случайного сигнала. В поле Sample Time необходимо указать период дискретизации сигнала $1/F_S = 1/44\ 100$.

Библиотека DSP Xilinx blockset содержит встроенную графическую среду для синтеза и анализа фильтров FDATool, представленную в виде одноименного блока помеченного маркером X (рис. 6.14а). Использование специальной функции xlfda_numerator(‘FDATool’) позволяет импортировать рассчитанные коэффициенты фильтра (числитель передаточной функции) непосредственно в блок MAC Based FIR. Коэффициенты можно также посмотреть в системе M*atlab* с помощью команды xlfda_numerator('FDATool').

Блок MAC Based FIR является параметризованным (6 переменных). Двойным кликом по блоку можно задать значения параметров в определенных полях (рис. 6.14б). Входные данные представляются в формате FIX_10_8, а коэффициенты как FIX_12_12. Обозначения переменных блока можно посмотреть в маске (правая кнопка мыши, Mask, Edit Mask) (рис. 6.15).

Максимальное значение коэффициента КИХ-фильтра составляет 0.3022 (определяется командой в системе M*atlab*: max(xlfda_numerator('FDATool')), а минимальное -0.067. Поэтому коэффициенты фильтра целесообразно представить в формате Fix_12_12. Это обеспечивает приведение коэффициентов к диапазону $[-0.5, 0.4998]$. Для сравнения можно было бы использовать формат Fix_12_11 (перенесение десятичной точки влево на один разряд при сохранении длины слова), что привело бы к диапазону $[-1, 0.9995]$.

На рис. 6.16 показана структурная схема КИХ-фильтра на 43 отвода с использованием одного MAC-блока и блочной двухпортовой памяти (ОЗУ). Входной сигнал перед записью в блочную память подвергается передискретизации на величину в 43 раза выше, чем частота взятия отсчетов F_S. Память разбита на два банка, которые используют различные режимы работы. Первый сконфигурирован как ОЗУ и работает в режиме циклического буффера, второй – как ПЗУ.

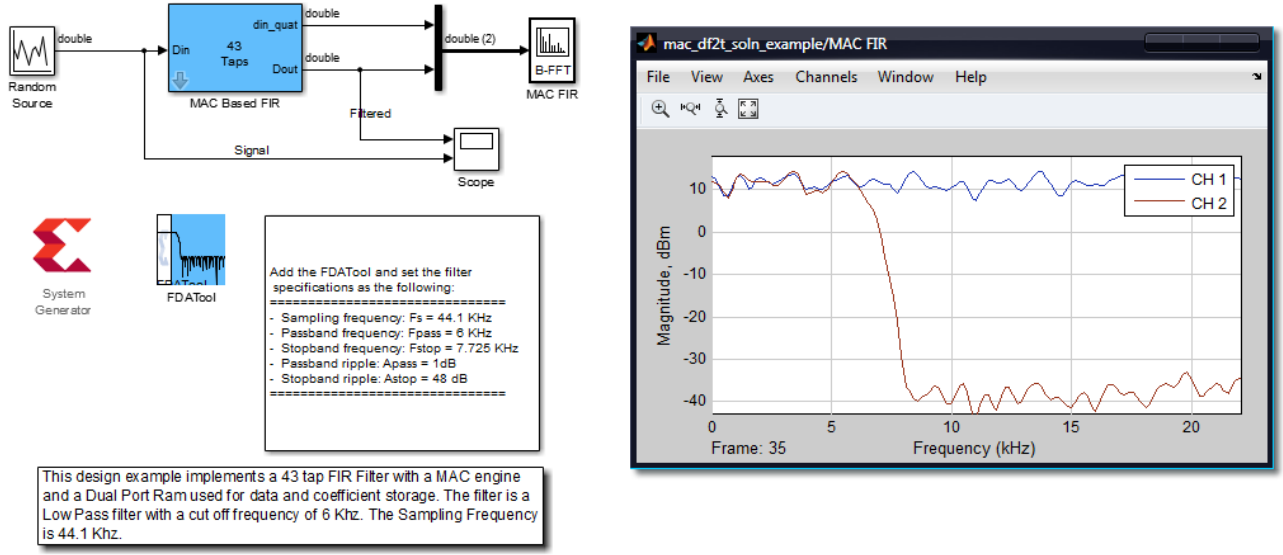


Рис. 6.12. Имитационная модель КИХ-фильтра на 43 отвода с возможностью вычисления коэффициентов по заданной спецификации с помощью графической среды FDATool и АЧХ. На вход фильтра подключен источник случайного сигнала

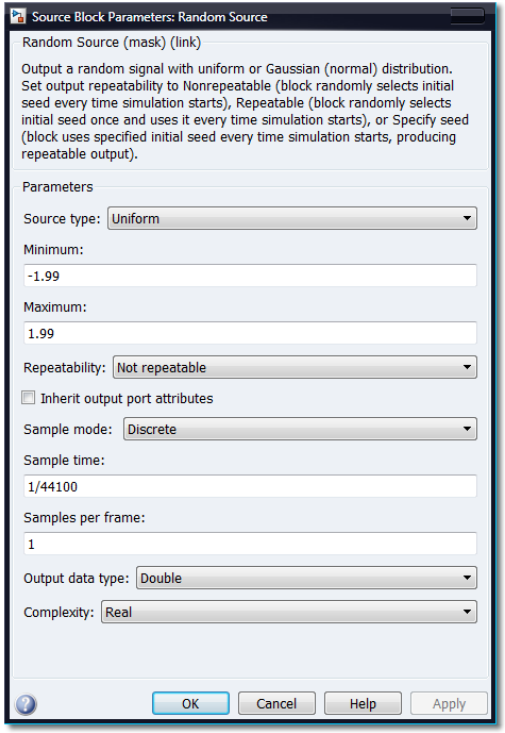
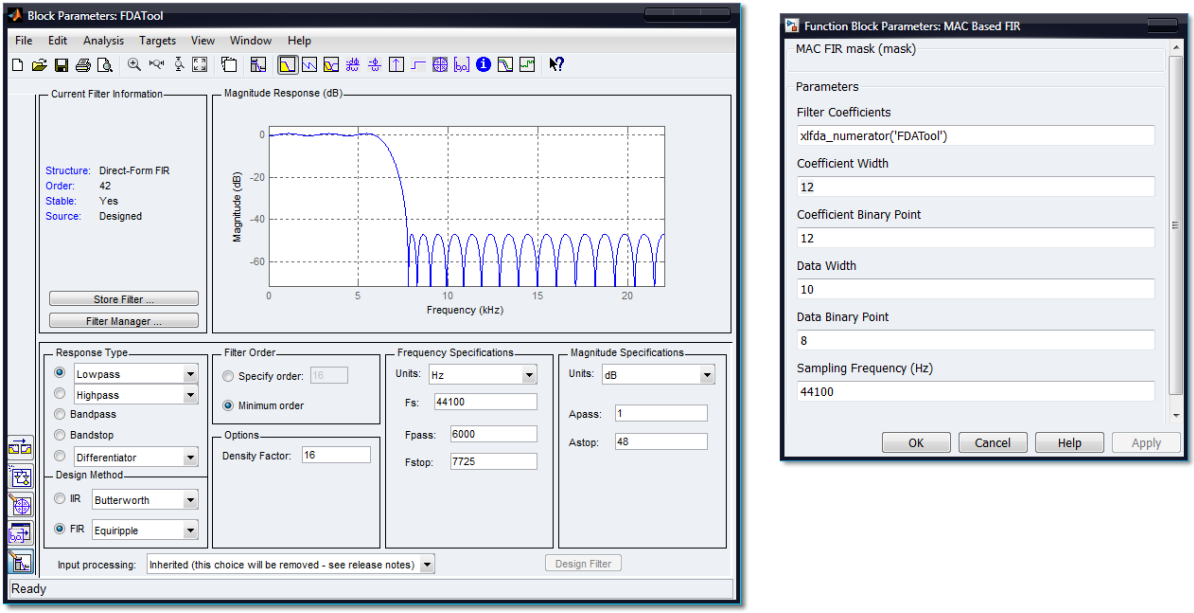


Рис. 6.13. Источник случайного сигнала с периодом дискретизации 1/44100



а) б)

Рис. 6.14. Блок FDATool (а) и маска с параметрами блока MAC Based FIR (б). Значения коэффициентов фильтра вычисляются с помощью команды xlfda_numerator(‘FDATool’)

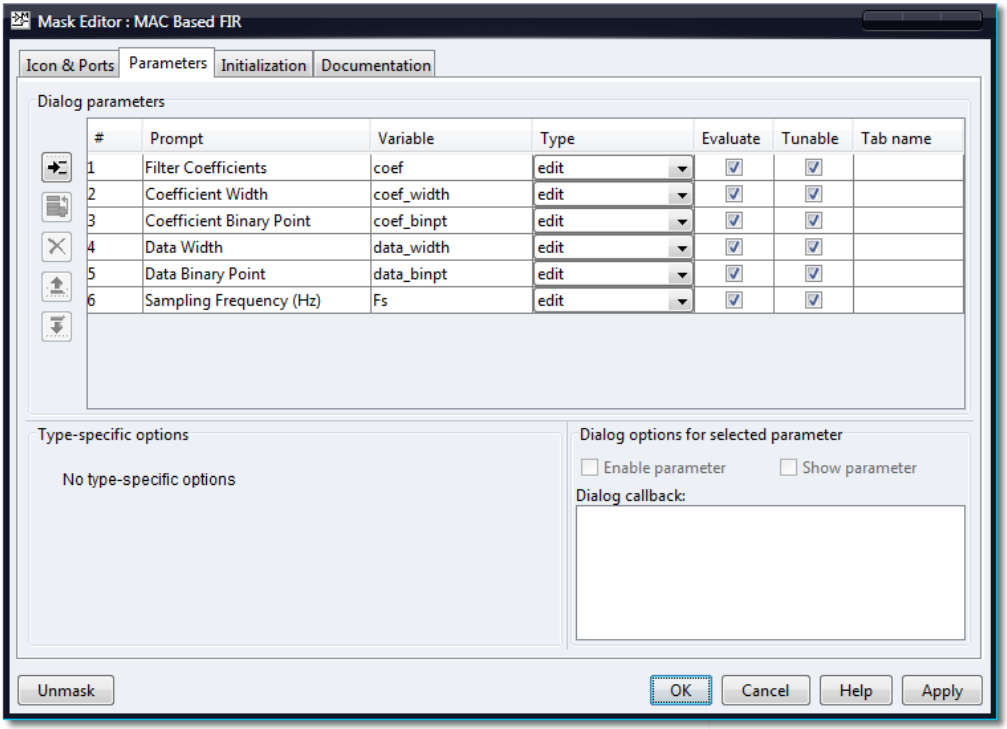


Рис. 6.15. Маска с параметрами функционального блока MAC Based FIR

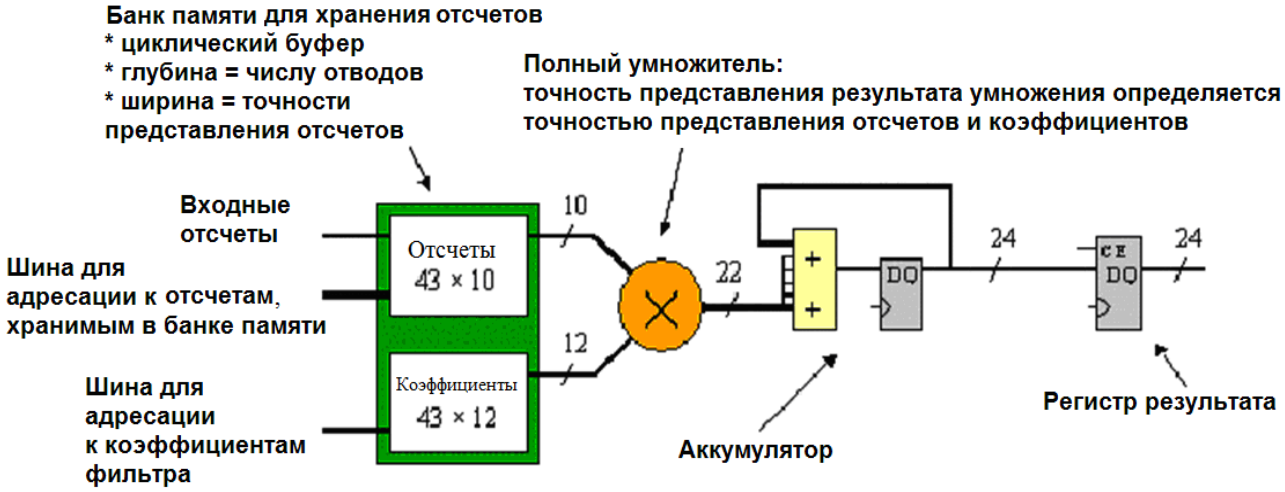


Рис. 6.16. Структурная схема КИХ-фильтра на 43 отвода с использованием одного MAC-блока и блочной памяти, разбитой на два банка, один из которых используется в виде циклического буфера для хранения и считывания отсчетов, а другой – для хранения коэффициентов фильтра

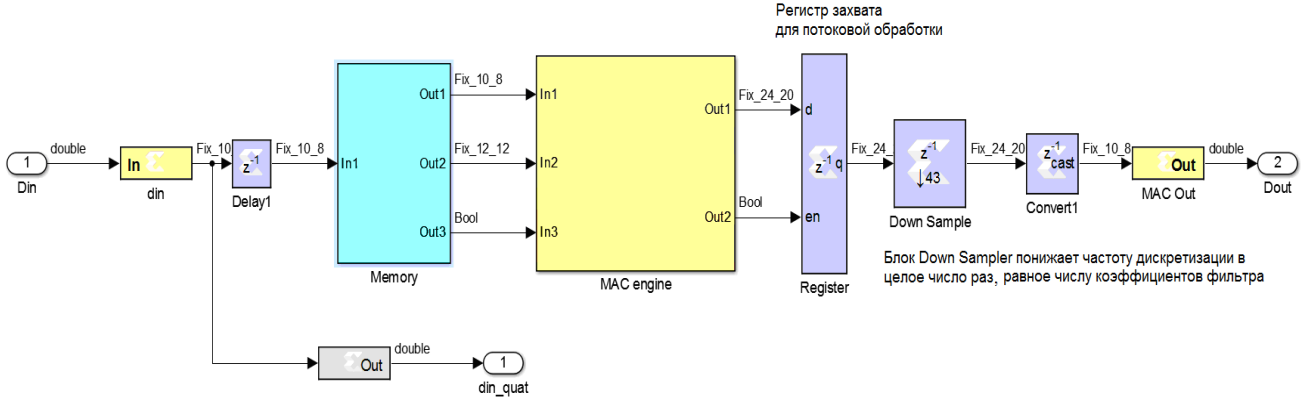


Рис. 6.17. Структурная схема MAC-фильтра с применением библиотек Xilinx System Generator

Емкость первого банка памяти, предназначенного для хранения отсчетов составляет 43 строки (адреса 0 – 42) на 10 столбцов (FIX_10_8) или 43 10-разрядных слова. Циклический буфер имитирует работу линии задержки фильтра (рис. 6.16). Второй банк предназначен для хранения коэффициентов фильтра емкостью 43 строки (адреса 43 – 85) на 12 столбцов (FIX_12_12) или 43 12-разрядных слова.

На рис. 6.17 показана структурная схема MAC-фильтра с применением библиотек System Generator. Схема предназначена для размещения в логические ресурсы ПЛИС. Основные функциональные блоки: Memory (двухпортовая память с управляющим автоматом), MAC engine (блок умножения с накоплением), Register (регистр захвата для операции конвейеризации), Down Sample (понижение частоты дискретизации в 43 раза), Convert 1 (представляет результат фильтрации с требуемой точностью или преобразует формат FIX_24_20, образующийся в процессе умножения и накопления в формат FIX_10_8). Блок din (Xilinx Gateway In, можно рассматривать как вход в ПЛИС) преобразует тип double в формат FIX_10_8. Блок MAC Out выполняет обратную функцию, преобразует формат FIX_10_8 в тип double. Его можно рассматривать как выход из ПЛИС.

Блок Dual Port RAM представляет память ОЗУ с двумя портами A и B. Входные отсчеты записываются и считываются из порта A (ОЗУ), коэффициенты считываются из порта B (рис. 6.18). Процессом записи и считывания управляет автомат (блок Address Control) (рис. 6.19).

Основные функциональные блоки автомата – это два суммирующих счетчика coef_counter (предварительно загружается число 43) и Data_Counter (предварительно загружается 0), адресующихся к портам A и B ОЗУ и компаратор (рис. 6.19).

На один из входов компаратора подключается константа 85 (команда 2*length(coef)-1). Компаратор управляет входом разрешения счета счетчика Data_Counter и формирует сигнал we (латентность 1), разрешающий запись в ОЗУ, и сигнал сброса аккумулятора (латентность 5) с последующим формированием сигнала разрешения работы регистра захвата.

ОЗУ (рис. 6.20) инициализируется вектором значений с применением следующей команды [zeros(1, length(coef)) (coef)] (рис. 6.21). Блок памяти имеет латентность – 1.

На рис. 6.20 показано, что сигнал, подлежащий фильтрации, подвергается передискретизации в 43 раза, перед тем, как быть записанным в ОЗУ. Передискретизация и последующая операция понижения частоты дескритизации (децимация) обеспечивают по внешним входам фильтра организовывать структуру фильтра типа Single-Rate FIR.

Для выравнивания числа столбцов в банках памяти используется блок pad (основан на блоке Xilinx Bus Concatenator), выполняющий роль конкатенации (склеивания) двух шин. Склеиваются две шины в форматах UFix_10_0 (10-разрядная шина, младшие разряды lo) и UFix_2_0 (2-разрядная шина, старшие разряды hi). Результатом склеивания является шина в формате UFix_12_0, которая преобразуется в тип Fix_12_12.

На рис. 6.22 показан умножитель и аккумулятор. Результат умножения представляется в формате Fix_22_20, а результат сложения – в формате Fix_24_20 с учетом переполнения и операции расширения знака числа. Разрядность выходной шины аккумулятора определяется следующей командой:

ceil(log2(max(1,sum(abs(coef*2^coef_binpt)))))+data_width+1.

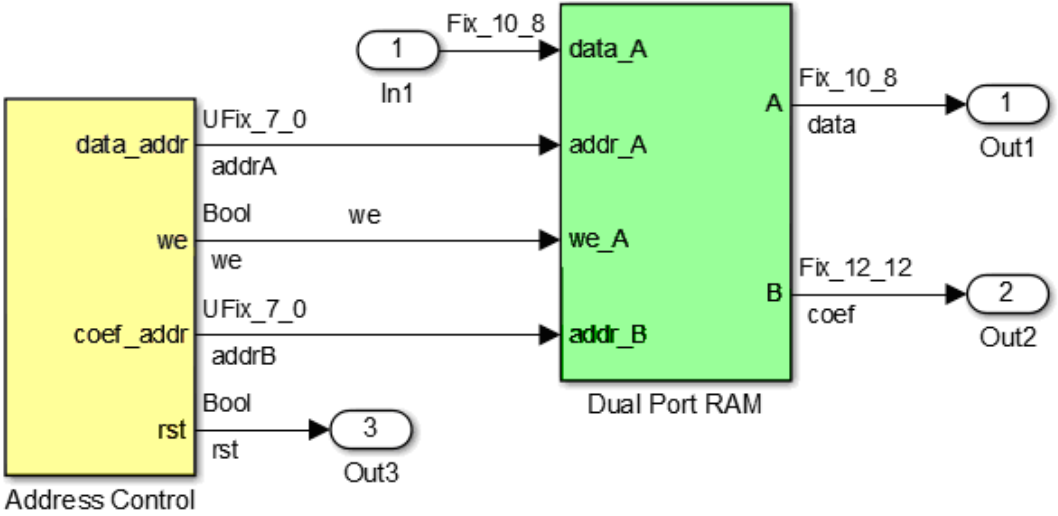


Рис. 6.18. Двухпортовая память с управляющим автоматом

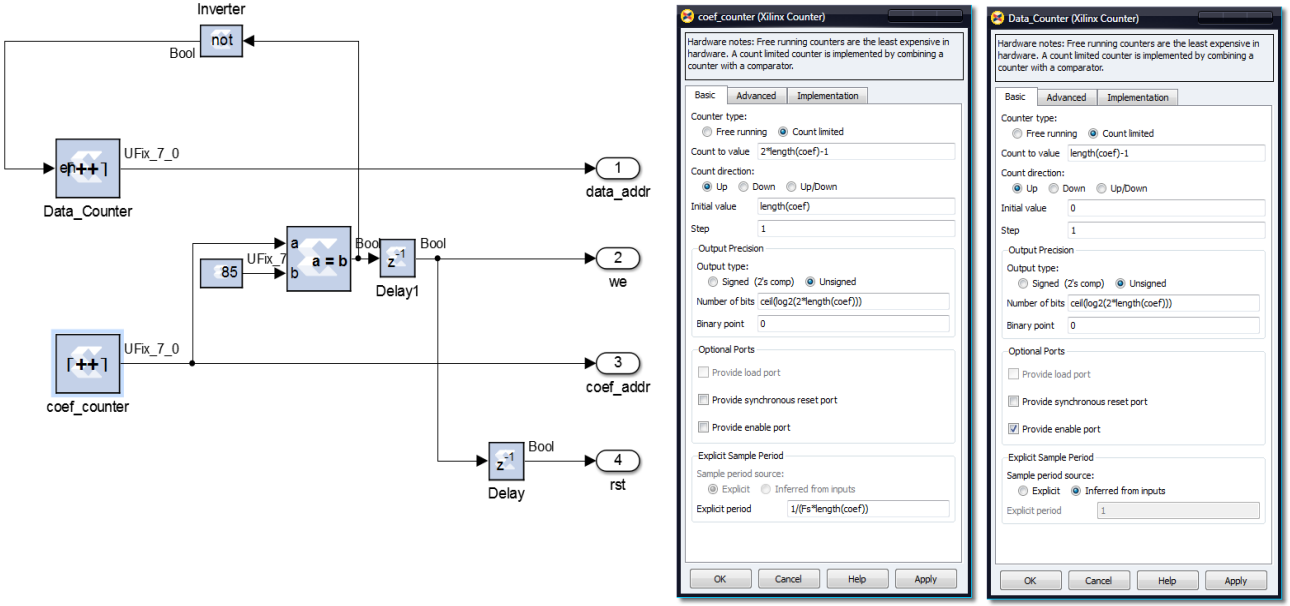


Рис. 6.19. Управляющий автомат двухпортовой памяти и настройки блоков счетчиков coef_counter и Data_Counter

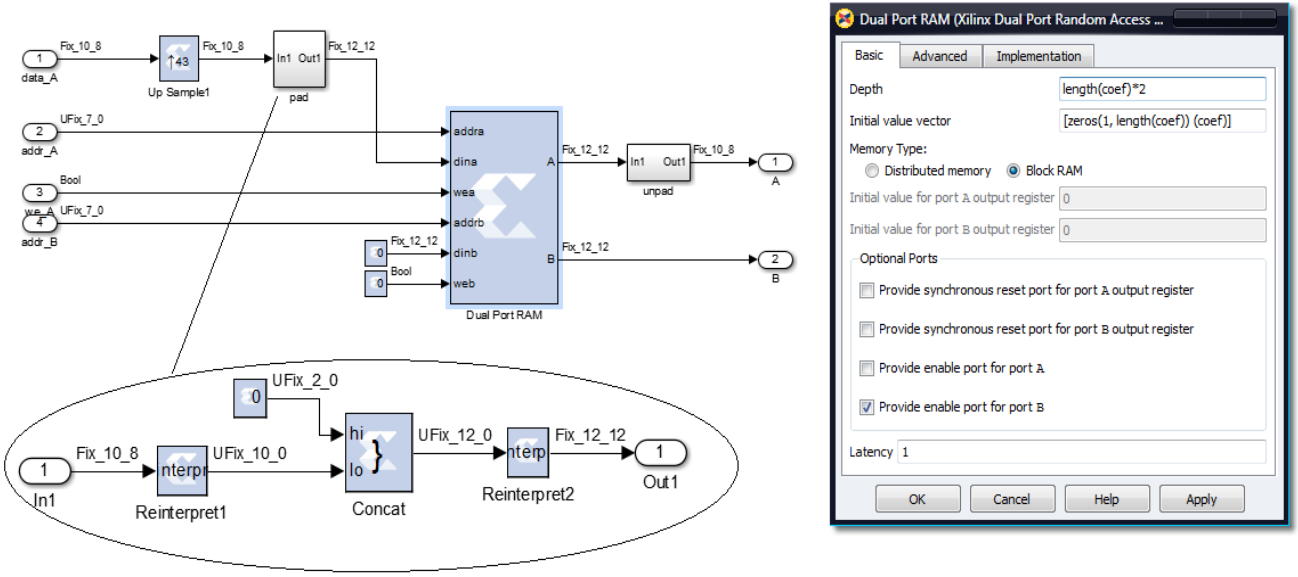


Рис. 6.20. Двухпортовая память на основе блочной памяти ПЛИС и настройки блока Xilinx Dual Random Access

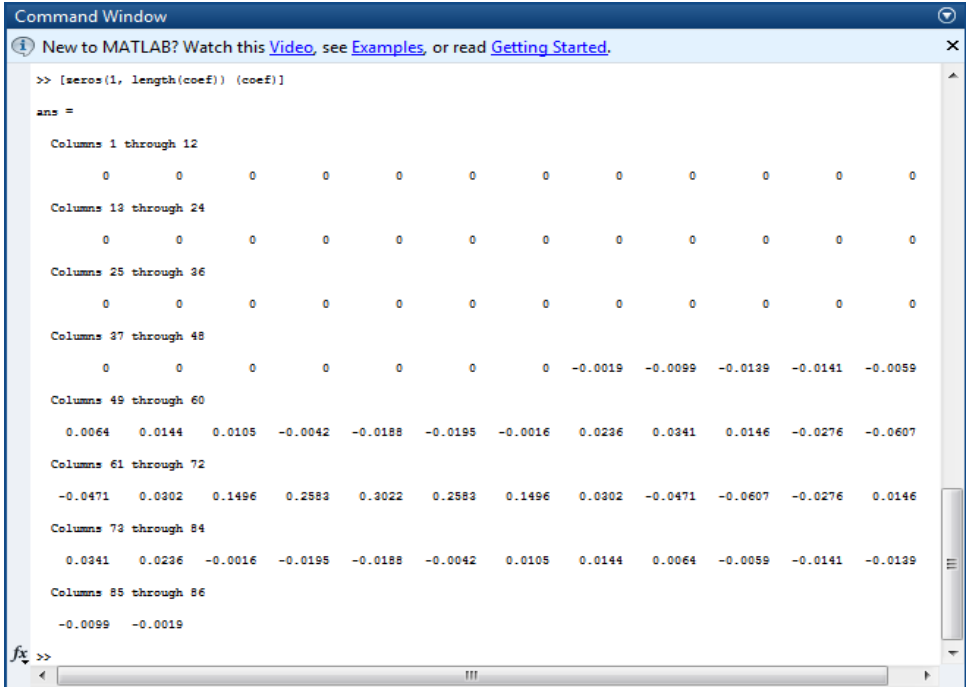


Рис. 6.21. Вектор инициализации блочной памяти ПЛИС
(коэффициенты КИХ-фильтра симметричны)

Умножитель имеет латентность 3 такта синхроимпульса для согласования с реальной латентностью равной 3-м, характерной для встроенных умножителей в ПЛИС Xilinx.

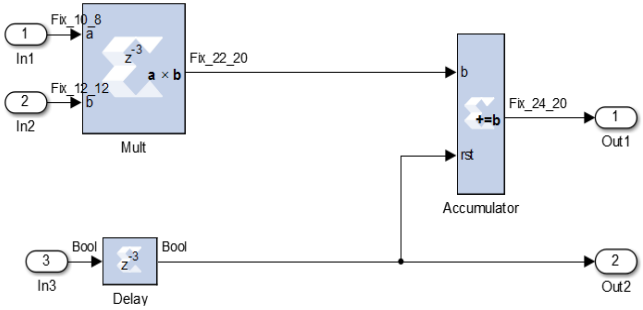


Рис. 6.22. Умножитель и аккумулятор

На рис. 6.23 показана настройка периода симуляции в Simulink. С учетом того, что используется передискретизация, период симуляции должен быть $1/(43F_S)$. Период синхросигнала задаем 10 нс. В меню Configuration Parameters задаем время симуляции 0.05 с. На рис. 6.24 показано удаление высокочастотных составляющих из случайного сигнала с помощью КИХ-фильтра на 43 отвода.

Имитационная модель и моделирование отклика КИХ-фильтра на единичный импульс в Simulink показаны на рис. 6.25. Период синхросигнала увеличен до 100 нс. Функциональное моделирование показывает, что входной сигнал (единичный импульс по амплитуде с произвольной шириной, не путать с дельта-функцией, позволяющей просмотреть коэффициенты фильтра) и выходной сигнал умножаются на 256 (рис. 6.26). Профильтрованные значения обновляются через 43 такта синхроимпульса. Фрагменты значений выходного сигнала и коэффициентов фильтра в форматах с плавающей double и фиксированной запятой FIX приведены в табл. 6.1.

Рис. 6.23. Настройка периода симуляции в Simulink

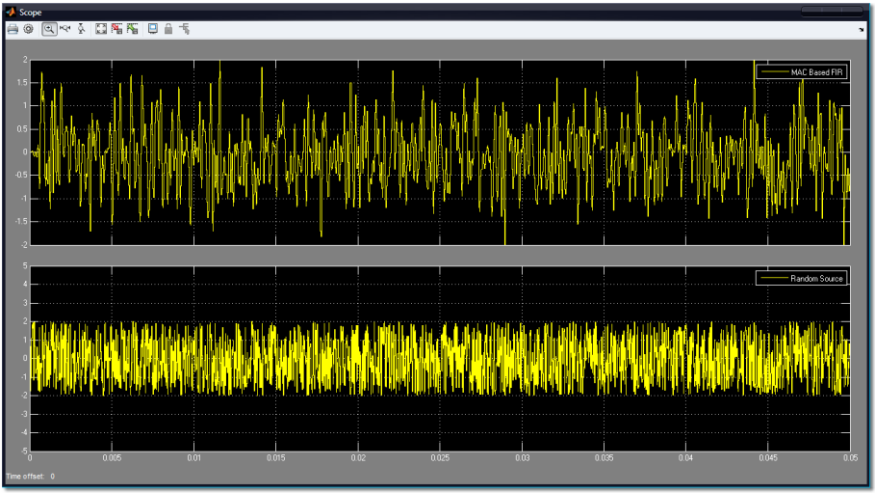
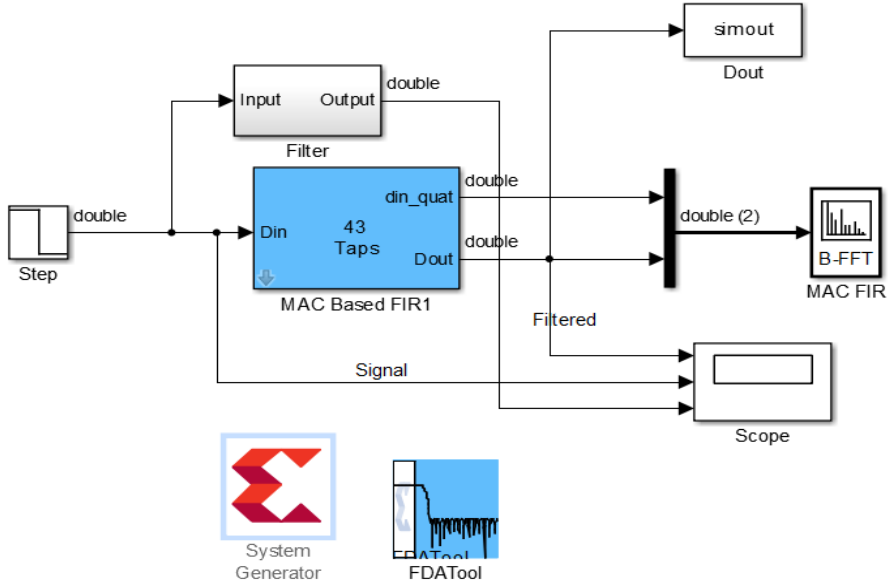
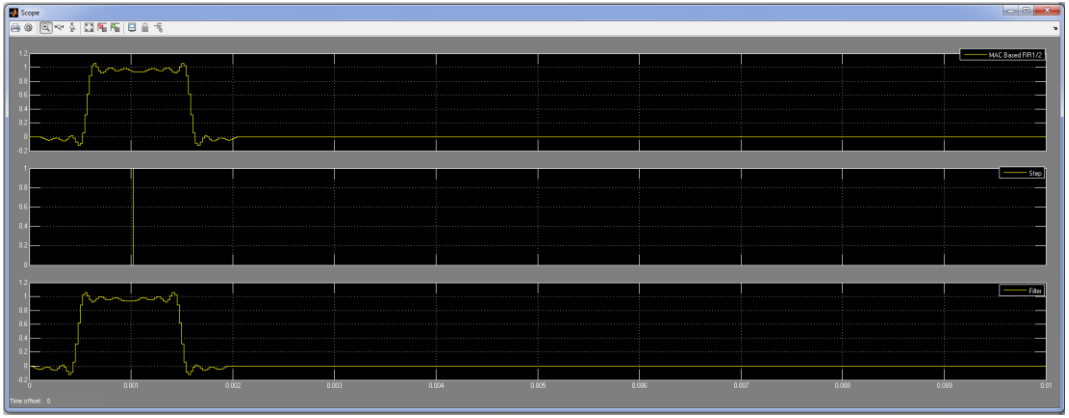


Рис. 6.24. Имитационное моделирование в системе M*atlab*/Simulink КИХ-фильтра на 34 отвода с использованием одного MAC-блока и блочной памяти



а)

Рис. 6.25. а) Имитационная модель КИХ-фильтра (на вход подается единичный импульс); б) моделирование отклика КИХ-фильтра



б)

Рис. 6.25. а) имитационная модель КИХ-фильтра (на вход подается единичный импульс);
б) моделирование отклика КИХ-фильтра

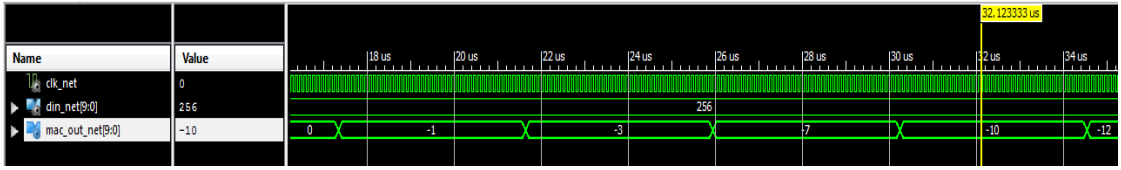


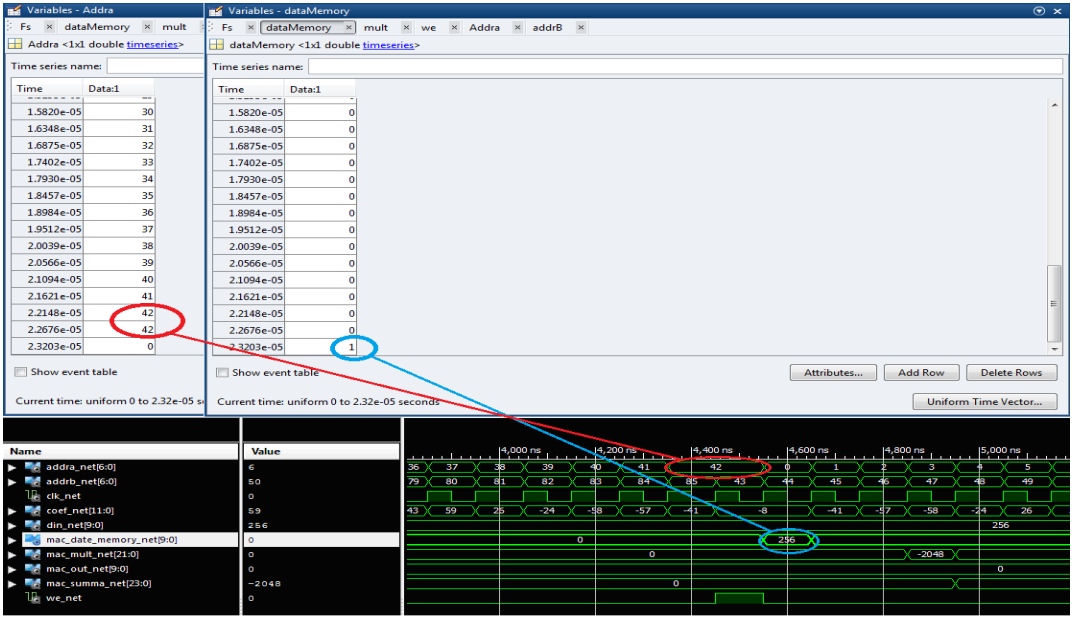
Рис. 6.26. Функциональное моделирование с использованием моделирующей программы на языке VHDL, сгенерированной в автоматическом режиме. Входной сигнал (единичный импульс) и выходной сигнал умножаются на 256

Таблица 6.1

Фрагменты значений выходного сигнала и коэффициентов фильтра в форматах double и FIX при прохождении по структуре единичного импульса

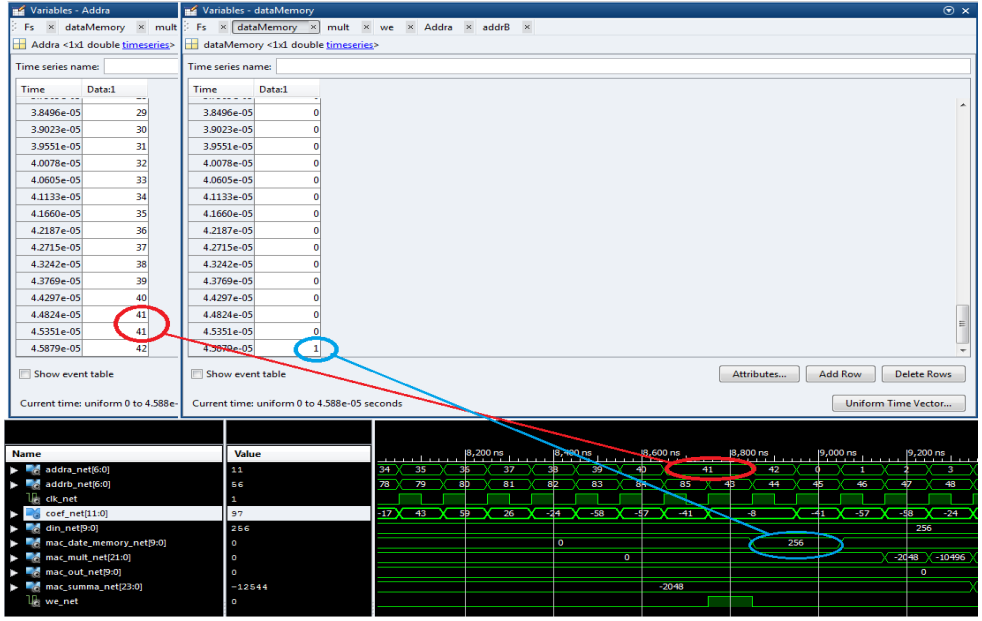
Значения отклика в формате double	Значения отклика в формате FIX_10_8 (значения отклика * на масштабный множитель 256)	Коэффициенты фильтра в формате double	Коэффициенты фильтра в формате FIX_12_12 (коэф. фильтра * на масштабный множитель 4096)
–0,00195;	–1	–0.0019	–8
–0,01196;	–3	–0.099	–41
–0,02588;	–7	–0.0139	–57
–0,0400;	–10	–0.0141	–58

В проекте используется оригинальная идея организации работы циклического буфера. Сигнал we приостанавливает на один такт синхроимпульса работу счетчика Data_Counter (рис. 6.27), тем самым происходит двойная адресация к строке 42 (два такта синхроимпульса). Это обеспечивает запись десятичного числа 1 в строку памяти с адресом 0. При работе в САПР ISE эта единица умножается на масштабный множитель 256. Далее это число будет умножено на коэффициент -8 (-0.0019 в формате double), что и даст результат -2048 (рис. 6.27а). Через последующие 43 такта десятичная единица будет записана в строку с адресом 42 (рис. 6.27б). Эта единица (256) вышеописанным способом заполнит первый банк ОЗУ, т.е. «пробежится» по всем коэффициентам фильтра, и процедура повторится снова в зависимости от ширины единичного импульса, которая задается параметром Step Time. В нашем случае это величина 0.001 (рис. 6.25, блок Step). Как только импульс упадет в ноль, им последовательно будет заполнен первый банк ОЗУ.



а)

Рис. 6.27. Результаты расчетов в пошаговом режиме в системе M*atlab* и временные диаграммы в ISE Design Suite, поясняющие принцип работы циклического буфера



б)

Рис. 6.27. Результаты расчетов в пошаговом режиме в системе M*atlab* и временные диаграммы в ISE Design Suite, поясняющие принцип работы циклического буфера (продолжение)

6.3. Проектирование последовательных КИХ-фильтров в системе Xilinx System Generator с применением библиотеки Reference BlockSet/DSP

Рассмотрим проектирование двух последовательных КИХ-фильтров на 4 отвода для реализации в базисе ПЛИС в системе Xilinx System Generator с использованием блока умножения и накопления (MAC-блока), линии задержки сигнала на основе адресуемого сдвигового регистра и двухпортовой блочной памяти, сконфигурированной для работы в различных режимах.

Проектирование КИХ-фильтра с использованием адресуемого сдвигового регистра и блочной памяти в режиме ПЗУ. В библиотеке Reference BlockSet/DSP системы Xilinx System Generator представлен параметризованный функциональный блок n-tap MAC FIR filter и пример на его основе, позволяющий спроектировать КИХ-фильтр на 16 отводов с использованием одного MAC-блока. Применение данного блока основано на том, что логический генератор ПЛИС Xilinx, основанный на статической памяти, может быть сконфигурирован как быстрый сдвиговый регистр с организацией 16×1. На базе такого адресуемого сдвигового регистра, основанного на примитиве SRL16E, можно построить линию задержки, а на основе блочной или распределенной памяти ПЛИС можно сконструировать ПЗУ для хранения коэффициентов фильтра.

Коэффициенты фильтра вычисляются с помощью функции fir1(15,.5) системы M*atlab* методом обратного преобразования Фурье с использованием окон, с частотой среза Wn, находящейся в диапазоне 0 < Wn < 1.0, где 1 соответствует половине частоты дискретизации Fs/2.

Предположим, что нам необходимо осуществить разработку КИХ-фильтра на 4 отвода в библиотеке Reference BlockSet/DSP: $y = C_0x_0 + C_1x_1 + C_2x_2 + C_3x_3$ с заданными коэффициентами $C_0 = -2$, $C_1 = -1$, $C_2 = 7$ и $C_3 = 6$. Рассмотрим случай, когда на вход фильтра поступает сигнал

-5, 3, 1, 0, 0, 0 и т.д. Правильные значения на выходе фильтра: 10, -1, -40, -10, 26, 6, 0 и т.д.

Воспользуемся готовым блоком n-tap MAC FIR filter (рис. 6.28). Проект разместим в базис ПЛИС серии Spartan-6 xa6slx4-3tqg144. Настроим маску с параметрами блока n-tap MAC FIR filter (рис. 6.29а). Коэффициенты фильтра и входной сигнал, подлежащий фильтрации, представляются в формате с фиксированной запятой с 4-битной точностью. Число разрядов дробной части – 0.

На рис. 6.29б показано задание частоты тактирования фильтра в САПР ISE (100 ns) и периода симуляции в Simulink (1/4 с). На рис. 6.29в и рис. 6.30 показана оценка ресурсов проекта в Simulink и САПР ISE. Для реализации КИХ-фильтра в базисе ПЛИС xa6slx4-3tqg144 требуется один ЦОС-блок DSP-48A. Максимальная частота проекта по коду языка VHDL, извлеченного в автоматическом режиме оценивается величиной 233 МГц. Рисунок 6.31 показывает имитационное моделирование в системе M*atlab*/Simulink КИХ-фильтра на 4 отвода.

Для более детального изучения структуры блока n-tap MAC FIR filter разработаем модель КИХ-фильтра на его основе для случая 4-tap MAC FIR filter. Рассмотрим управление латентностью линии задержки на базе адресуемого сдвигового регистра. Структурная схема адресуемого сдвигового регистра показана на рис. 6.32. Разработаем имитационную модель управления адресуемым сдвиговым регистром с помощью 2-разрядного суммирующего счетчика (рис. 6.33). Результаты имитационного моделирования показывают, что латентность блока ASR составляет 4 такта синхрочастоты (рис. 6.34). Латентность блока может быть задана вручную или вычислена исходя из максимальной разрядности адресного порта. Подключение задержки на один такт на вход ASR-блока приводит к увеличению латентности на 8 тактов синхрочастоты (рис. 6.35 и рис. 6.36). А подключение задержки на один такт на его вход и задержки на два такта на выходе составит для такой конструкции 10 тактов синхрочастоты (рис. 6.37 и рис. 6.38).

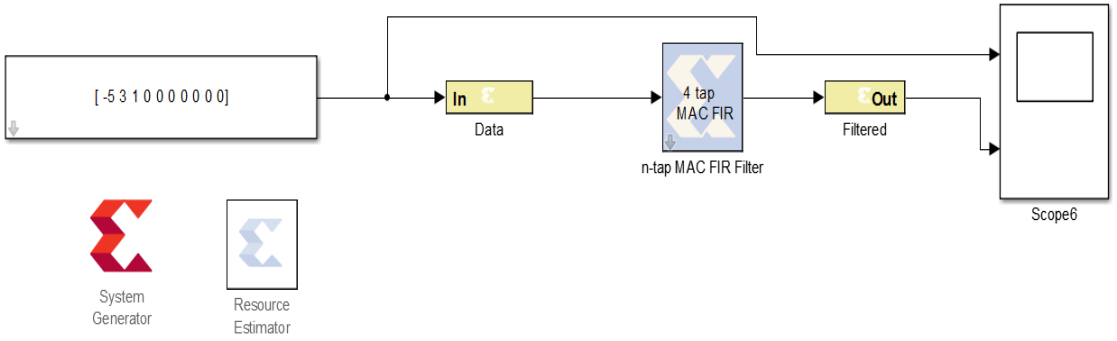
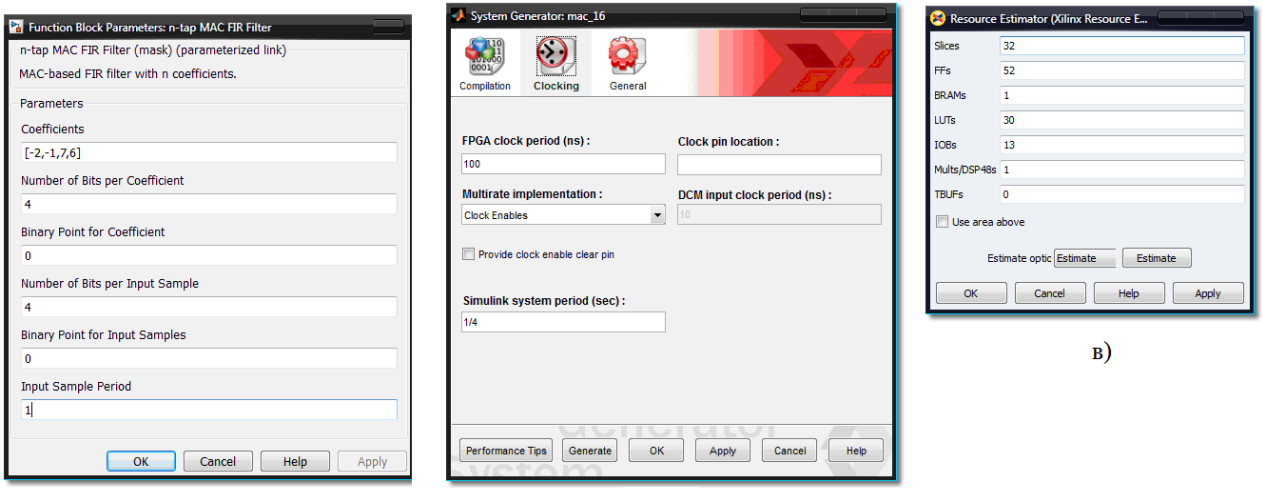


Рис. 6.28. Имитационная модель КИХ-фильтра на 4 отвода для реализации в базисе ПЛИС серии Spartan-6 xa6slx4-3tqg144 на основе n-tap MAC FIR filter



а) б) в)

Рис. 6.29. а) маска с параметрами блока n-tap MAC FIR filter; б) задание частоты тактирования фильтра в САПР ISE (100 ns) и периода симуляции в Simulink (1/4 с); в) оценка ресурсов проекта в Simulink

Device Utilization Summary			
Slice Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	52	4,800	1%
Number used as Flip Flops	52		
Number used as Latches	0		
Number used as Latch-thrus	0		
Number used as AND/OR logics	0		
Number of Slice LUTs	39	2,400	1%
Number used as logic	22	2,400	1%
Number using O6 output only	20		
Number using O5 output only	0		
Number using O5 and O6	2		
Number used as ROM	0		
Number used as Memory	8	1,200	1%
Number used as Dual Port RAM	0		
Number used as Single Port RAM	0		
Number used as Shift Register	8		
Number of DSP48A1s	1	8	12%

Рис. 6.30. Оценка ресурсов проекта в САПР ISE по VHDL-коду, извлеченному в автоматическом режиме с помощью маркера System Generator

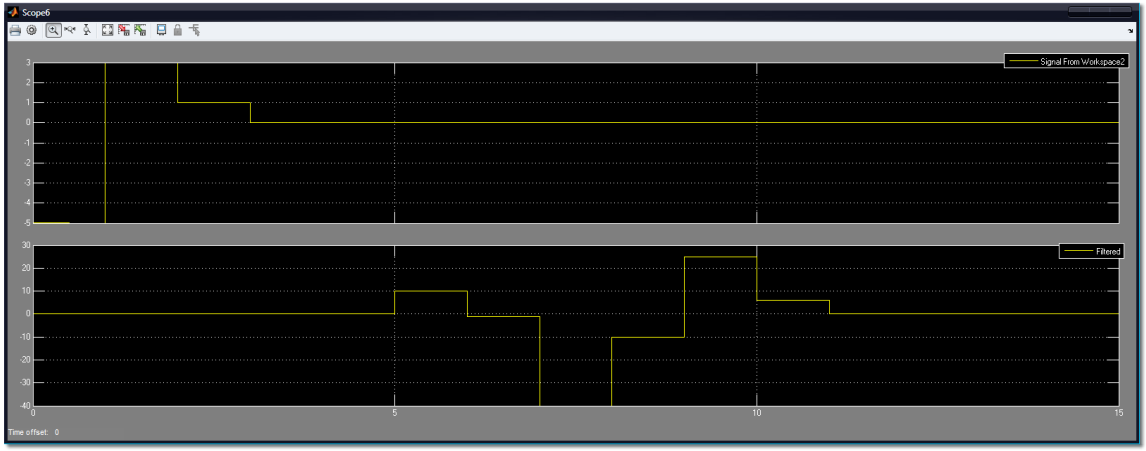


Рис. 6.31. Имитационное моделирование в системе M*atlab*/Simulink КИХ-фильтра на 4 отвода

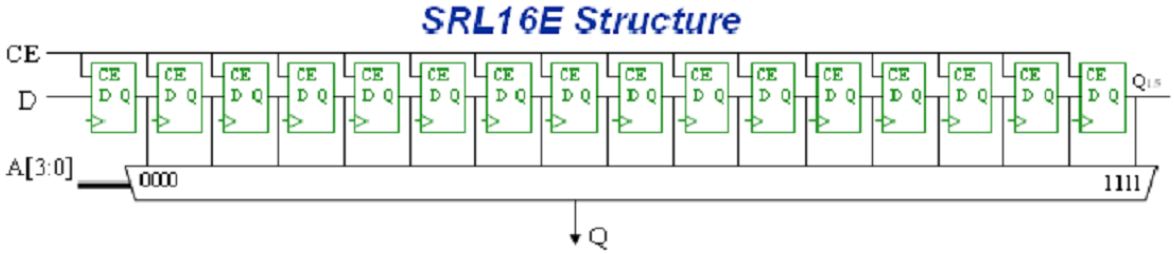


Рис. 6.32. Структурная схема адресуемого сдвигового регистра на примитиве SRL16E

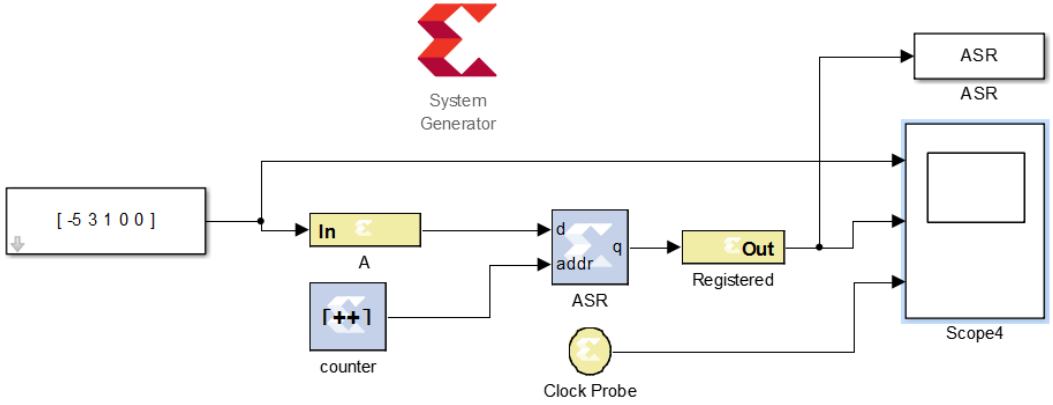


Рис. 6.33. Управление адресуемым сдвиговым регистром (функциональный блок ASR) с помощью счетчика

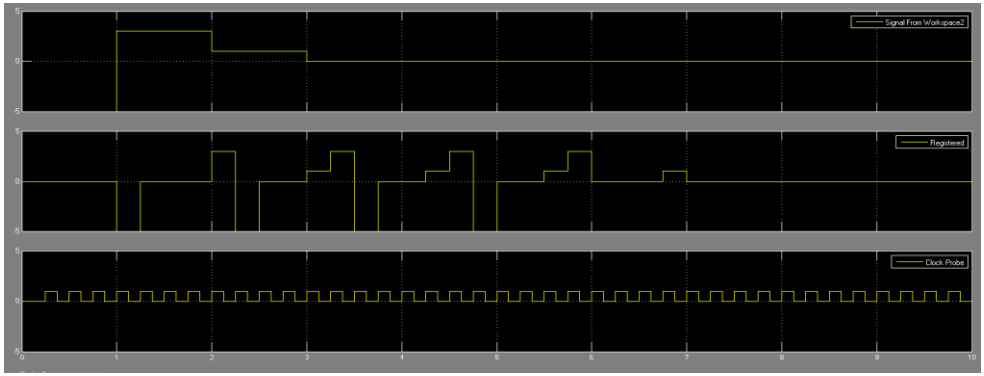


Рис. 6.34. Латентность ASR-блока 4 такта синхрочастоты

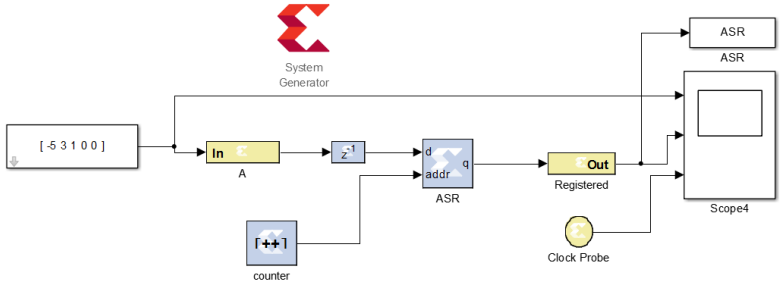


Рис. 6.35. Исследование латентности ASR-блока при подключении задержки на один такт на его вход

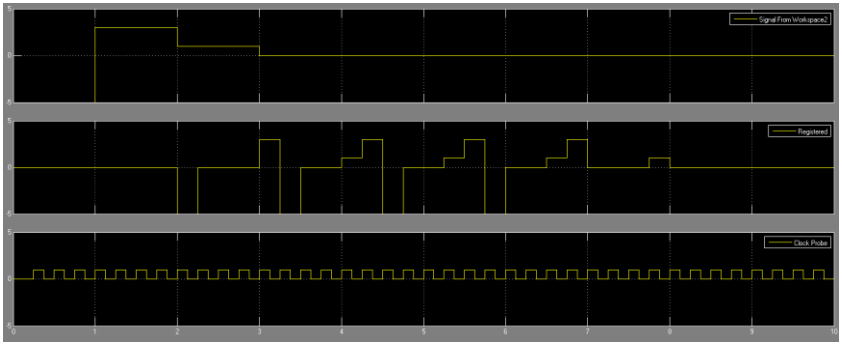


Рис. 6.36. Латентность ASR-блока при подключении задержки на его вход составляет 8 тактов синхрочастоты

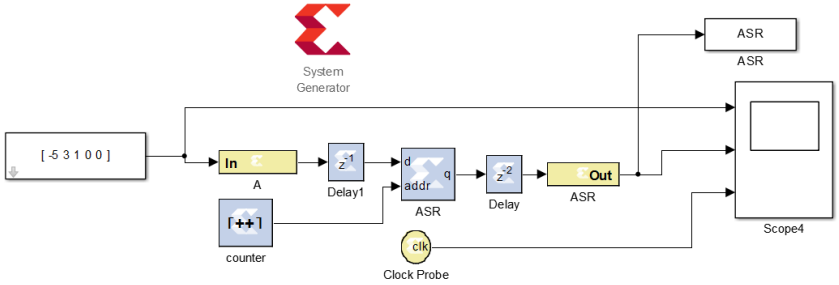


Рис. 6.37. Исследование латентности ASR-блока при подключении задержки на один такт на его вход и задержки на два такта на выход

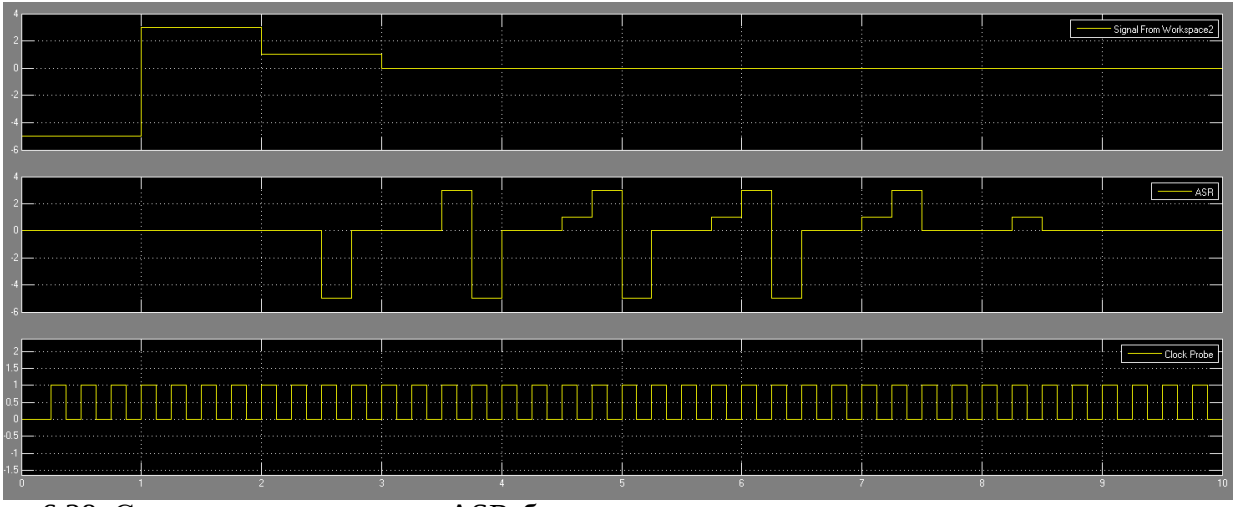


Рис. 6.38. Суммарная латентность ASR-блока при подключении задержки на один такт на его вход и задержки на два такта на выход составит 10 тактов синхрочастоты

Рассмотрим структуру блоков, входящих в состав функционального блока n-tap MAC FIR filter. Структурная схема управляющего автомата (функциональный блок Address Control) показана на рис. 6.39, а на рис. 6.40 представлена структурная схема функционального блока Memory. Выход суммирующего 2-разрядного счетчика подключен к адресным шинам блоков ASR и ROM (ПЗУ). Емкость ПЗУ составляет 4 строки по 4 бита, инициализируется вектором значений $[-2\ -1\ 7\ 6]$. На рис. 6.41 показаны настройки ASR-блока и ROM-блока.

Латентность сигналов A и B составит 10 и 2 такта синхрочастоты, что обеспечивает правильность формирования результата умножения. Сигнал res_mac в структурной схеме функционального блока Memory имеет латентность 5 тактов синхрочастоты. Обеспечивает правильность загрузки новых значений произведений сигналов A и B (сигнал aXb) в аккумулятор MAC-блока и последующих накоплений значений суммы произведений (рис. 6.42). Умножитель, входящий в состав Mult-блока, имеет латентность три такта синхрочастоты и реализуется на базе аппаратных умножителей ПЛИС, встроенных в DSP-блоки. Произведение сигналов A и B представляется с 8-битной, а сумма произведений с 9-битной точностью.

На рис. 6.43 показан КИХ-фильтр на 4 отвода с использованием MAC-блока на основе функционального блока n-tap MAC FIR filter. Фильтр состоит из функциональных блоков Memory, MAC engine, регистра и блока понижения частоты дискретизации. На рис. 6.44 показаны значения сигналов на выходах блоков. Сигнал A (Data_RAM) на выходе линии задержки имеет латентность 10 тактов синхрочастоты (рис. 6.44а). Коэффициенты фильтра, извлекаемые из ПЗУ, имеют латентность два такта синхрочастоты (сигнал B, он же Coef_ram) (рис. 6.44б). Сигнал res_mac имеет латентность – 5 (рис. 6.44в).

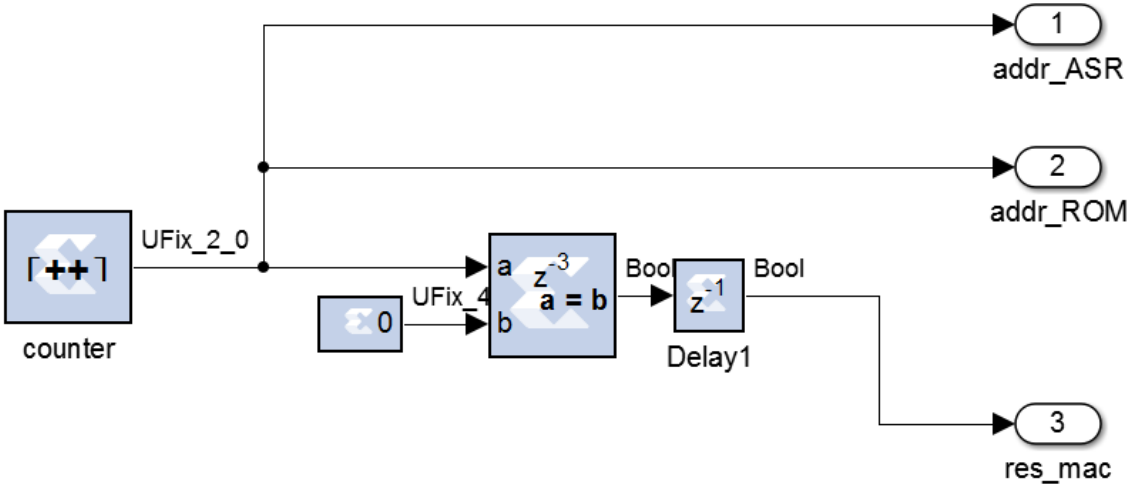


Рис. 6.39. Структурная схема управляющего автомата
(функциональный блок Address Control)

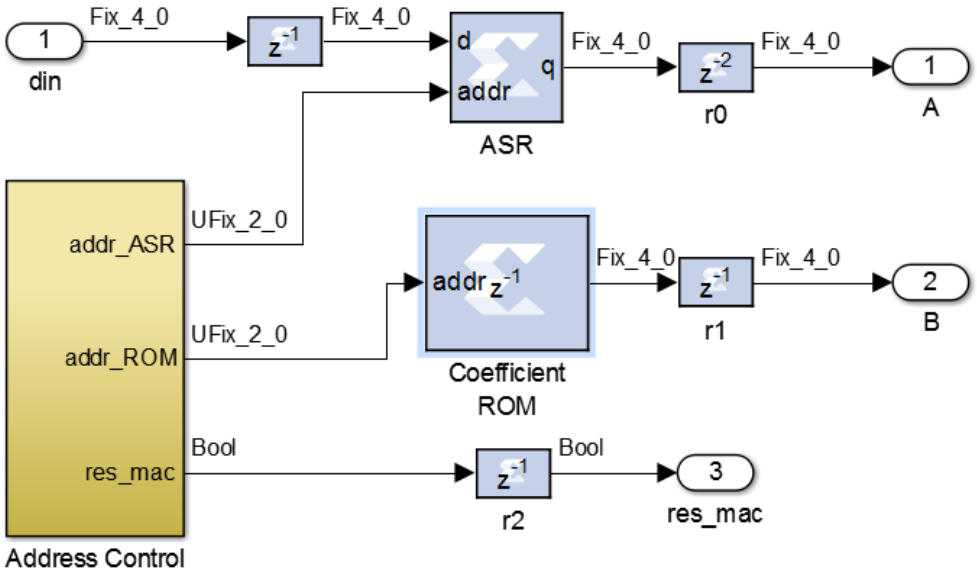


Рис. 6.40. Структурная схема функционального блока Memory

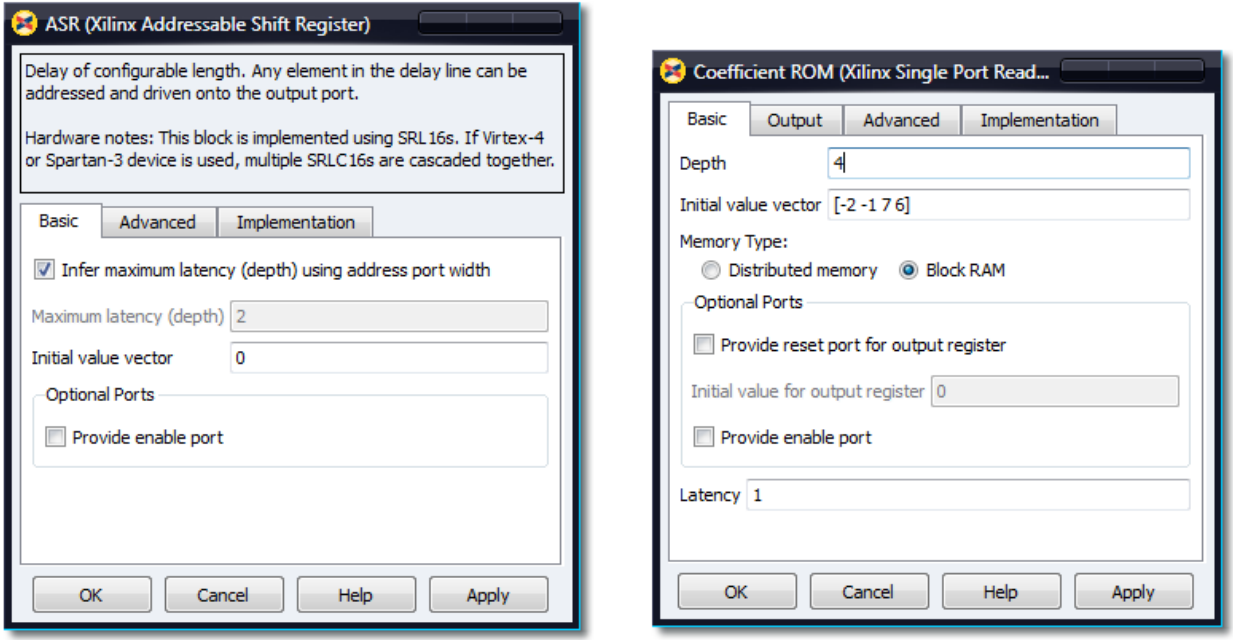


Рис. 6.41. Настройки ASR-блока и ROM-блока

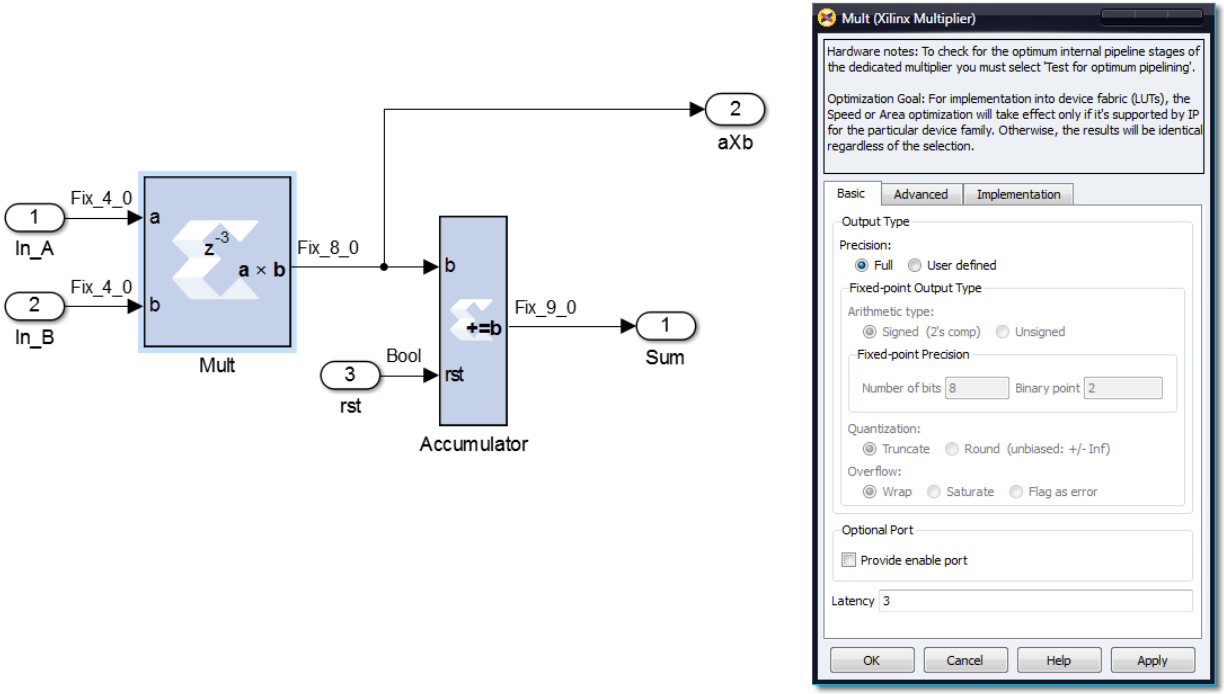


Рис. 6.42. Блок умножения с накоплением (функциональный блок MAC engine) и настройки умножителя (Mult-блок)

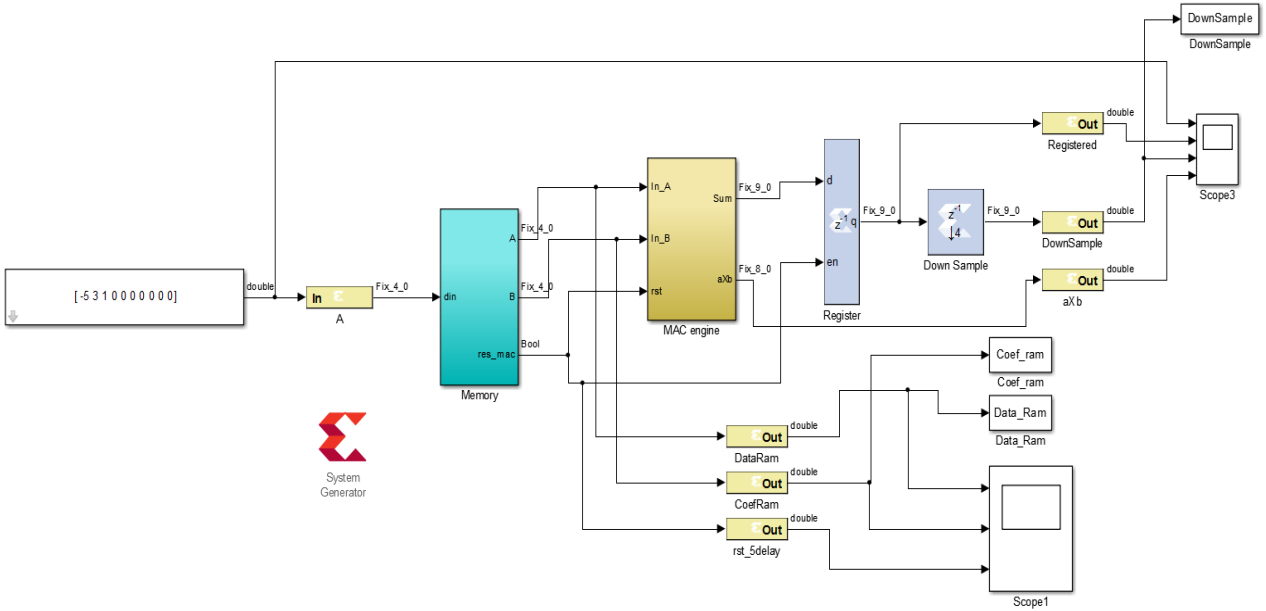


Рис. 6.43. Имитационная модель КИХ-фильтра на 4 отвода на основе функционального блока n-tap MAC FIR filter

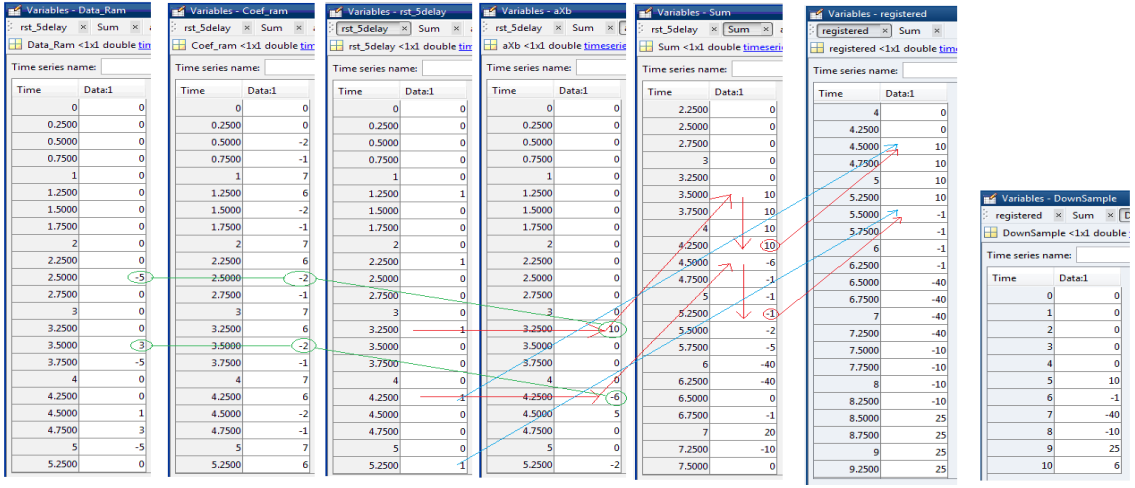


Рис. 6.44. Представление значений сигналов в главном окне системы M*atlab* в дискретные моменты времени: а) сигнал A; б) коэффициенты фильтра (сигнал B, он же Coef_ram); в) сигнал res_mac, обеспечивающий правильность загрузки новых значений произведения aXb в аккумулятор MAC-блока и последующих накоплений значений суммы произведений; г) значение произведения aXb на выходе умножителя; д) сигнал Sum, накопленный в аккумуляторе; е) сигнал registered на выходе регистра захвата; к) сигнал DownSample (сигнал после операции децимации, результат фильтрации)

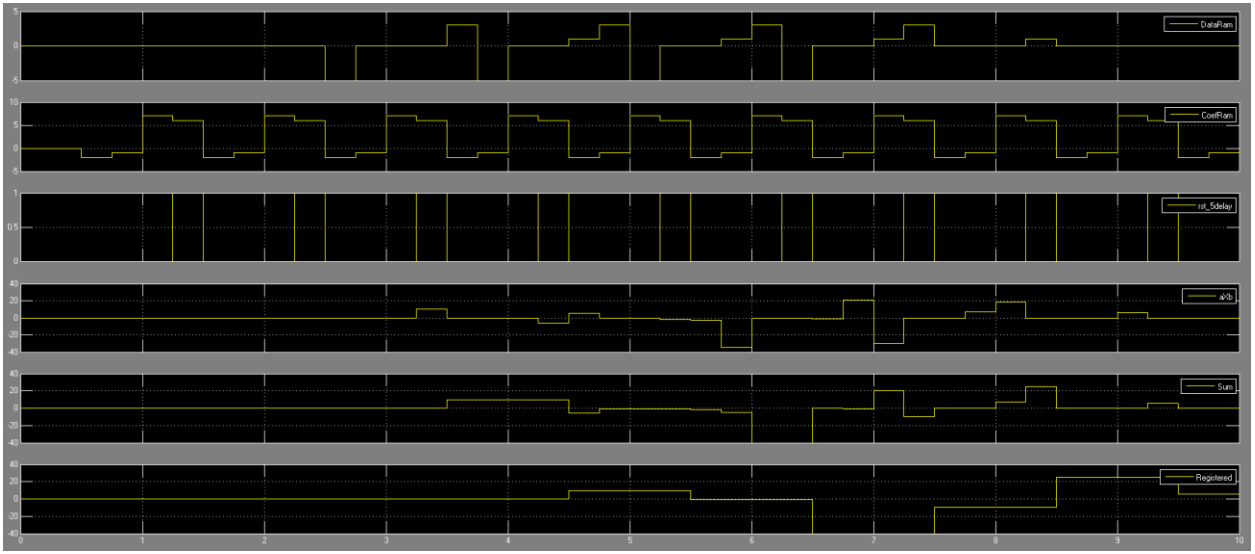


Рис. 6.45. Имитационное моделирование в системе M*atlab*/Simulink КИХ-фильтра на 4 отвода, созданного на основе функционального блока n-tap MAC FIR filter. Сигналы расположены согласно позициям а), б), в), г), д) и е), представленным на рис. 6.44

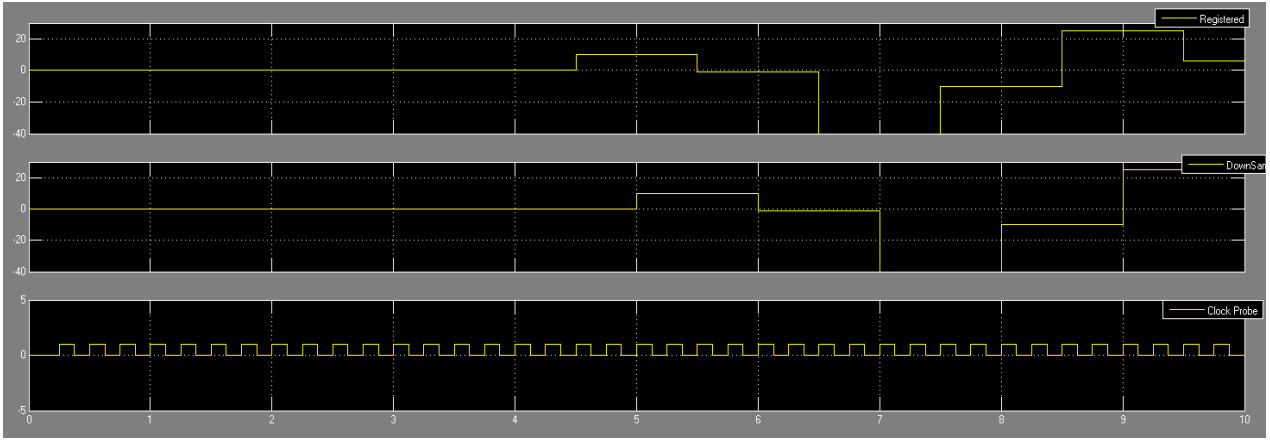


Рис. 6.46. Сигнал до и после операции децимации (позиции *е* и *к* на рис. 6.44)

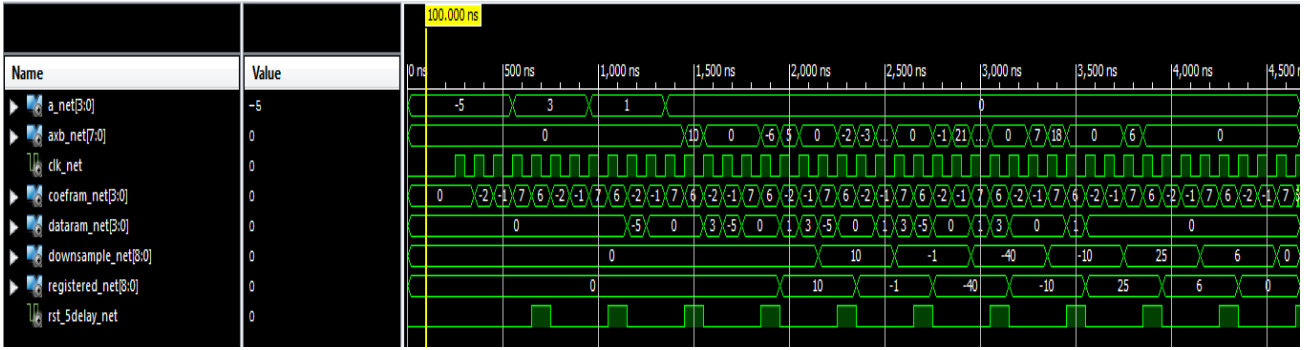


Рис. 6.47. Функциональное моделирование КИХ-фильтра на 4 отвода, созданного на основе функционального блока n-tap MAC FIR filter с использованием моделирующей программы на языке VHDL, сгенерированной в автоматическом режиме.
Обозначение сигналов согласно рис. 6.43

Сигнал aXb является текущим значением произведения сигналов A и B (рис. 6.44г), а сигнал Sum представляет сумму значений произведений сигналов A и B, накопленных в аккумуляторе (рис. 6.44д). Сигналы aXb и Sum формируются в блоке MAC engine.

На рис. 6.45 показаны сигналы, расположенные согласно позициям а), б), в), г), д) и е), соответствующие рис. 6.44, а на рис. 6.46 представлен сигнал до и после операции децимации (позиции *е* и *к* на рис. 6.44). Зелеными (формирование произведения), красными (формирование суммы произведений) и синими (значения суммы произведений в регистре захвата) стрелками на рис. 6.44 обозначены информационные потоки при вычислениях. Правильные значения на выходе фильтра: 10, -1, -40, -10, 26, 6 и все последующие нулевые значения показаны на рис. 6.44г.

Результаты функционального моделирования в САПР Xilinx ISE с использованием испытательного стенда, код которого получен в автоматическом режиме, показаны на рис. 6.47. Сравнивая результаты имитационного моделирования на рис. 6.45 и функционального, представленного на рис. 6.47, приходим к выводу, что фильтры работают корректно.

Проектирование КИХ-фильтра на основе блочной памяти в режиме ОЗУ. КИХ-фильтр также может быть реализован на основе функционального блока n-tap Dual Port Memory MAC FIR Filter, находящегося в библиотеке Reference BlockSet/DSP.

Имитационная модель КИХ-фильтра на 4 отвода для реализации в базисе ПЛИС серии Spartan-6 xa6slx4-3tqg144, адаптированная к нашей задаче, показана на рис. 6.48.

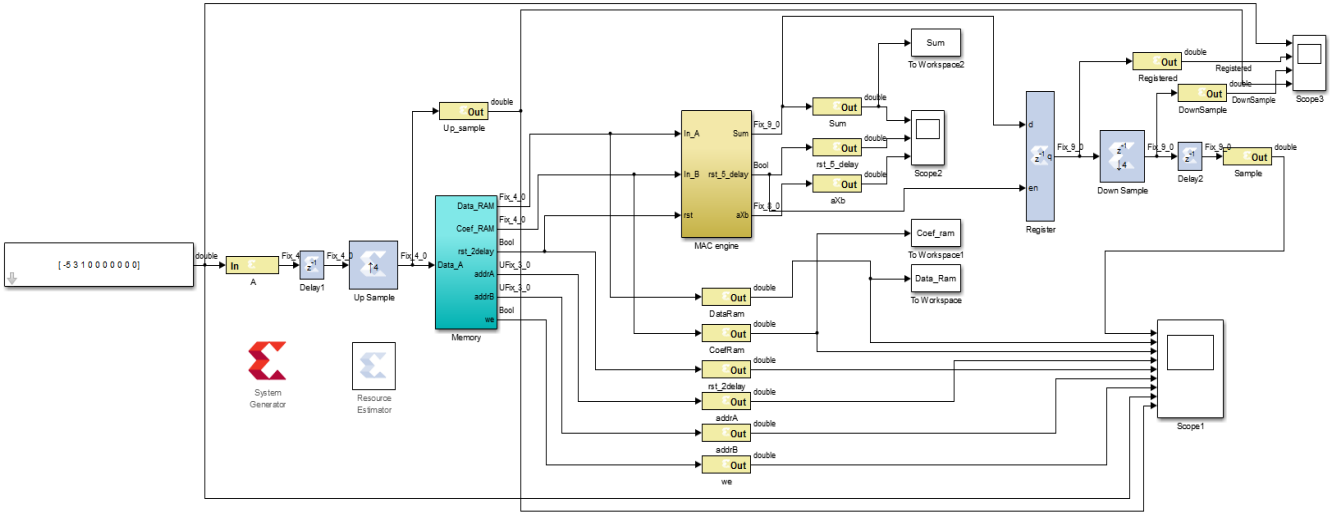


Рис. 6.48. Имитационная модель КИХ-фильтра на 4 отвода для реализации в базисе ПЛИС серии Spartan-6 xa6slx4-3tqg144 на основе n-tap Dual Port Memory MAC FIR Filter

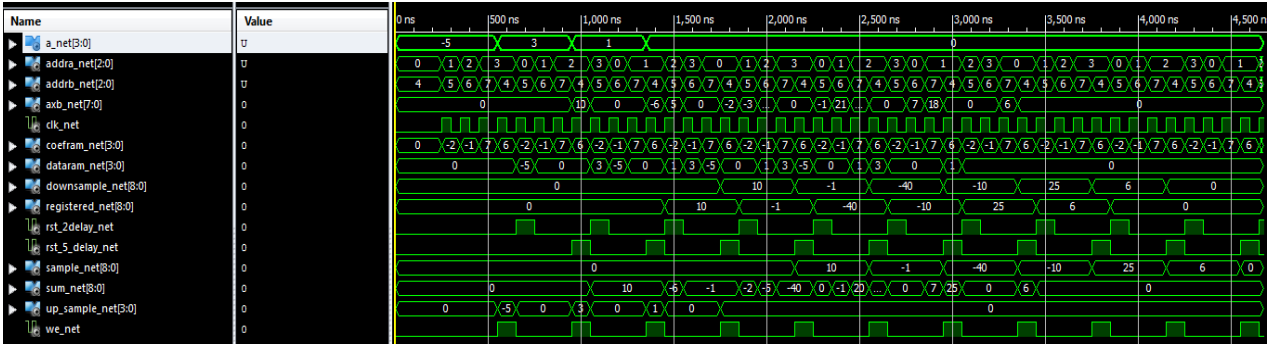


Рис. 6.49. Функциональное моделирование КИХ-фильтра на четыре отвода, созданного на основе функционального блока n-tap Dual Port Memory MAC FIR Filter с использованием моделирующей программы на языке VHDL, сгенерированной в автоматическом режиме

Таблица 6.2

Оценка ресурсов ПЛИС серии Spartan-6 xa6slx4-3tqg144 при реализации КИХ-фильтров на 4 отвода различными способами

Ресурсы ПЛИС	Генератор параметризированных ядер XLogiCORE IP функцией FIR Compiler v6.3, систолическая структура	XLogiCORE IP FIR Compiler v5.0, последовательная распределенная арифметика	XLogiCORE IP FIR Compiler v5.0, параллельная распределенная арифметика	Simulink, Xilinx System Generator, n-tap MAC FIR filter, 1 MAC-блок	Simulink, Xilinx System Generator, Dual Port Memory MAC FIR Filter, 1 MAC-блок
Рабочая частота, МГц	348	439	438	233	279
Число ЦОС-блоков DSP-48A	1	–	–	1	1
Триггеров	48	57	111	52	57
Секций с LUT	33	41	88	39	38

В составе КИХ-фильтра на основе n-tap Dual Port Memory MAC FIR Filter используются следующие функциональные блоки: MAC-блок; блоки интерполяции и децимации; двухпортовая память, разбитая логически на два банка памяти, один из которых работает как циклический буфер для считывания и записи входных отсчетов сигнала, подлежащего фильтрации, а второй для хранения коэффициентов фильтра (выполняет функцию ПЗУ). Вектор инициализации блочной памяти [0 0 0 0 –2, –1, 7, 6]. Емкость ОЗУ восемь 4-разрядных слов. Порт A настроен на режим чтение – затем запись. Порт B – чтение.

На рис. 6.49 показано функциональное моделирование. Максимальная частота проекта, созданного с использованием VHDL-кода, извлеченного в автоматическом режиме, оценивается величиной 279 МГц. Оценка ресурсов ПЛИС серии Spartan-6 xa6slx4-3tqg144 при реализации КИХ-фильтров на четыре отвода различными способами показана в табл. 6.2.

В Xilinx System Generator рассмотрено проектирование последовательных КИХ-фильтров на 4 отвода в формате с фиксированной запятой с использованием параметризованных функциональных блоков n-tap MAC FIR filter и n-tap Dual Port Memory MAC FIR Filter. Основные используемые блоки для построения структур КИХ-фильтров: адресуемый сдвиговый регистр для организации линии задержки на базе LUT; управляющий автомат; ПЗУ или ОЗУ на основе блочной памяти; умножитель и аккумулятор; интерполяция и децимация. Для правильного функционирования КИХ-фильтров необходимо производить учет латентности блоков.

Реализация КИХ-фильтра на основе блочной памяти ПЛИС n-tap Dual Port Memory MAC FIR Filter дает повышенное быстродействие 279 МГц против 233 МГц на основе адресуемого сдвигового регистра при незначительном возрастании требуемых логических ресурсов ПЛИС (табл. 6.2).

Данный эффект объясняется увеличением задержек в трассировочных ресурсах ПЛИС при реализации сдвигового регистра на ячейках конфигурационной памяти. В обоих случаях требуется один ЦОС-блок DSP-48A.

Как показывает анализ табл. 6.2, последовательные КИХ-фильтры с использованием одного MAC-блока являются самыми медленными по отношению к систолическому КИХ-фильтру, являющемуся разновидностью параллельной структуры (функция FIR Compiler v6.3), и к фильтру на основе распределенной арифметики, позволяющей организовывать «безумножительные» схемы умножения (FIR Compiler v5.0,). Однако в случае роста числа отводов фильтра можно получить существенный выигрыш в экономии ресурсов ПЛИС.

6.4. Проектирование КИХ-фильтров в системе Xilinx System Generator с применением методологии Black Boxes

Использование методологии Black Boxes Xilinx System Generator при разработке имитационных моделей цифровых устройств позволяет импортировать VHDL, Verilog, EDIF-коды разработанные, например, в САПР ПЛИС Xilinx ISE Design Suite в систему M*atlab*/Simulink, что значительно повышает возможности объектно-ориентированного проектирования.

Импорт проектов созданных в САПР ПЛИС Xilinx ISE Design Suite с использованием единственных VHDL-файлов. В Xilinx System Generator рассмотрим проектирование КИХ-фильтра на четыре отвода $y = C_0 x_0 + C_1 x_1 + C_2 x_2 + C_3 x_3$ с заданными коэффициентами $C_0 = -2$, $C_1 = -1$, $C_2 = 7$ и $C_3 = 6$ с применением методологии Black Boxes. Рассмотрим случай, когда на вход фильтра

поступает сигнал –5, 3, 1, 0, 0 и 0 т.д. Правильные значения на выходе фильтра: 10, –1, –40, –10, 25, 6 и 0 т.д.

Пример 1 демонстрирует нижний уровень иерархии проекта КИХ-фильтра на четыре отвода, созданного с использованием единственного VHDL-файла. Верхним уровнем иерархии является схемный файл fir.sch, состоящий из символа с именем fir4 и портов (не показан). Испытательный стенд сформируем в ручном режиме.

Согласно рекомендациям Xilinx для эффективного использования DSP-блоков ПЛИС необходим сигнал синхронного сброса (активным является сигнал низкого уровня). Выражение для выходного сигнала фильтра также должно использовать тактовый сигнал.

В данном примере для преобразования типов используется пакет std_logic_arith библиотеки IEEE (возможно также использование пакета numeric_std). Переменные pro и acc представлены в 8-ми и 10-разрядном дополнительном двоичном коде (знаковый тип signed). Поскольку порты date и q_reg типа std_logic_vector, то необходимо осуществить преобразование типов с помощью следующих функций conv_signed(conv_integer(date),4) для сигнала date и std_logic_vector(acc) для сигнала q_reg. Функция conv_integer(date) преобразует сигнал date в целое десятичное число с учетом знака, а функция conv_signed преобразует в 4-разрядный дополнительный двоичный код. Для линии задержки организованной на переменной shift формируется тип shif_arr (массив из четырех 4-разрядных двоичных слов) заданный пользователем.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
package coeffs is
type coef_arr is array (0 to 3) of signed(3 downto 0);
```

```
constant coefs: coef_arr:= coef_arr'("1110", "1111", "0111", "0110");
end coeffs;
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
use work.coeffs.all;
entity fir4 is
port (clk, reset, ce: in std_logic;
date: in std_logic_vector(3 downto 0);
q_reg: out std_logic_vector (9 downto 0));
end fir4;
architecture beh of fir4 is
begin
process(clk,reset,ce,date)
type shift_arr is array (3 downto 0) of signed (3 downto 0);
variable shift: shift_arr;
variable pro: signed(7 downto 0);
variable acc: signed (9 downto 0);
begin
q_reg<= (others => '0');
if(clk'event and clk = '1') then
if reset='0' then
for i in 0 to 3 loop
shift(i):= (others => '0');
end loop;
elsif ce='1' then
shift(0):=conv_signed(conv_integer(date),4);
pro := shift(0) * coefs(0);
acc := conv_signed(pro, 10);
for i in 2 downto 0 loop
pro := shift(i+1) * coefs(i+1);
acc := acc + conv_signed(pro, 10);
shift(i+1):= shift(i);
end loop;
end if;
end if;
```

```
q_reg<=std_logic_vector(acc);
end process;
end beh;
```

Пример 1. VHDL-код КИХ-фильтра на четыре отвода

Убедившись с помощью испытательного стенда в том, что код языка VHDL КИХ-фильтра на четыре отвода работает корректно, перейдем к разработке имитационной модели. На рис. 6.50 показана имитационная модель КИХ-фильтра на четыре отвода для реализации в базисе ПЛИС серии Spartan-6 xa6slx4-3tqg144 на основе функционального блока Black Box. Настройки маркера System Generator следующие: период синхросигнала 100 нс, период симуляции в Simulink 1 с.

Для того чтобы воспользоваться данной методологией, необходимо из библиотеки Xilinx System Generator добавить в разрабатываемую модель фильтра функциональный блок Black Box и указать на VHDL-файл с именем fir4 (пример 1), а также подключить симулятор ISE (рис. 6.51). В автоматическом режиме будет создан m-файл (m-функция КИХ-фильтра) системы M*atlab* с именем fir_config.

С учетом того, что современные ПЛИС построены на основе методологии синхронного проектирования, необходимо в коде языка VHDL для последовательностных устройств обеспечить связку синхросигнала clk и сигнала разрешения его подачи ce со следующей конструкцией:

```
elsif (clk'event and clk='1') then
if  (ce = '1') then
….
```

При этом на функциональном блоке Black Boxes сигналы clk и ce не отображаются. m-файл fir_config системы Matlab является редактируемым. Например, по умолчанию результат фильтрации на шине q_reg представляется в формате с фиксированной запятой для беззнаковых чисел как 'UFix_10_0'. Поскольку коэффициенты фильтра и входные

отсчеты могут быть как со знаком, так и без, то необходимо отредактировать формат представления профильтрованного сигнала Sout с 'UFix_10_0' на 'Fix_10_0'. В дальнейшем это придется делать для всех рассматриваемых примеров ниже.

Анализируя m-файл видим, что сформировался фильтр типа Single-Rate FIR (входная частота дискретизации fclk равна выходной частоте дискретизации, т.е. без изменения частоты дискретизации, такие фильтры получили название как “односкоростные фильтры”), функция

setup_as_single_rate(block,clkname,cename).

Частота дискретизации определяется как fclk/N для несимметричного и как fclk/N+1 для симметричного фильтра, где fclk–частота тактирования ядра фильтра; N–разрядность входной шины данных (точность представления входных значений подлежащих фильтрации).

```
function fir4_config(this_block)
  this_block.setTopLevelLanguage('VHDL');
  this_block.setEntityName('fir4');
  % System Generator has to assume that your entity  has a combinational
feed through;
  %   if it  doesn't, then comment out the following line:
  this_block.tagAsCombinational;
  this_block.addSimulinkInport('reset');
  this_block.addSimulinkInport('date');
  this_block.addSimulinkOutport('q_reg');
  q_reg_port = this_block.port('q_reg');
  q_reg_port.setType('Fix_10_0');
  % ----------------------------
  if (this_block.inputTypesKnown)
    % do input type checking, dynamic output type and generic setup in
this code block.
    if (this_block.port('reset').width ~= 1);
      this_block.setError('Input data type for port "reset" must have
width=1.');
    end
```

```
    this_block.port('reset').useHDLVector(false);
    if (this_block.port('date').width ~= 4);
      this_block.setError('Input data type for port "date" must have
width=4.');
    end
  end  % if(inputTypesKnown)
  % -----------------------------
  % -----------------------------
   if (this_block.inputRatesKnown)
     setup_as_single_rate(this_block,'clk','ce')
   end  % if(inputRatesKnown)
  % -----------------------------
    % (!) Set the inout port rate to be the same as the first input
    %     rate. Change the following code if this is untrue.
    uniqueInputRates = unique(this_block.getInputRates);
  % Add addtional source files as needed.
  %  |-------------
  %  | Add files in the order in which they should be compiled.
  %  | If two files "a.vhd" and "b.vhd" contain the entities
  %  | entity_a and entity_b, and entity_a contains a
  %  | component of type entity_b, the correct sequence of
  %  | addFile() calls would be:
  %  |    this_block.addFile('b.vhd');
  %  |    this_block.addFile('a.vhd');
  %  |-------------
  %    this_block.addFile('');
  %    this_block.addFile('');
  this_block.addFile('fir4.vhd');
return;
% ------------------------------------------------------------
function setup_as_single_rate(block,clkname,cename)
  inputRates = block.inputRates;
  uniqueInputRates = unique(inputRates);
  if (length(uniqueInputRates)==1 & uniqueInputRates(1)==Inf)
    block.addError('The inputs to this block cannot all be constant.');
    return;
  end
```

```
 if (uniqueInputRates(end) == Inf)
    hasConstantInput = true;
    uniqueInputRates = uniqueInputRates(1:end-1);
 end
 if (length(uniqueInputRates) ~= 1)
   block.addError('The inputs to this block must run at a single rate.');
   return;
 end
 theInputRate = uniqueInputRates(1);
 for i = 1:block.numSimulinkOutports
    block.outport(i).setRate(theInputRate);
 end
 block.addClkCEPair(clkname,cename,theInputRate);
 return;
```

Пример 2. m-файл, генерируемый Xilinx System Generator (с комментариями)

На рис. 6.52 показаны результаты имитационного моделирования в системе Matlab/Simulink КИХ-фильтра на четыре отвода, а на рис. 6.53 – функциональное моделирование в САПР ПЛИС Xilinx ISE Design Suite v.14.4 для двух случаев. В первом случае (рис. 6.53а) проект создан на основе единственного VHDL-файла (пример 1) с ручным формированием испытательного стенда, а во втором (рис. 6.53б) – оптимизированный VHDL-код с автоматическим формированием испытательного стенда, созданный с помощью маркера Xilinx System Generator.

Сравнивая результаты имитационного и функционального (рис. 6.52 и рис. 6.53) моделирования видим, что КИХ-фильтр работает корректно. Анализ рис. 6.53 показывает, что по коду языка VHDL, приведенному в примере 1, сформировался параллельный КИХ-фильтр. После задержки в четыре такта синхроимпульса, результат вычисления уже доступен с приходом нового такта синхроимпульса.

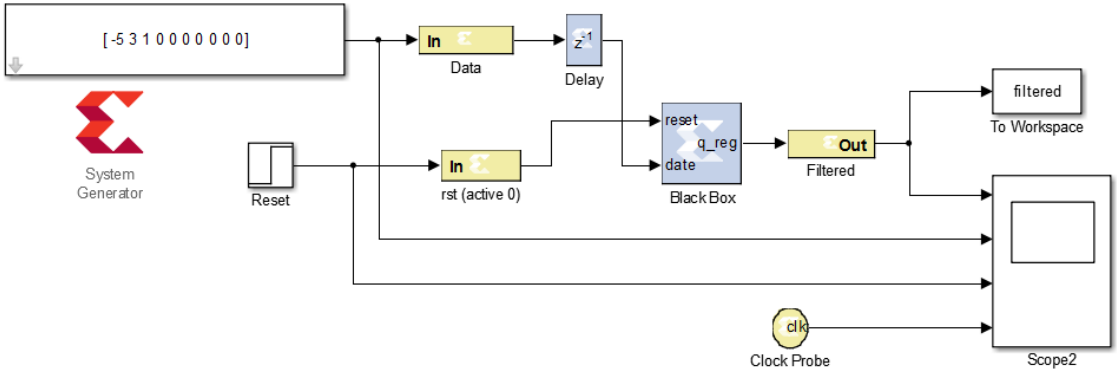


Рис. 6.50. Имитационная модель КИХ-фильтра на четыре отвода для реализации в базисе ПЛИС серии Spartan-6 xa6slx4-3tqg144 на основе функционального блока Black Box

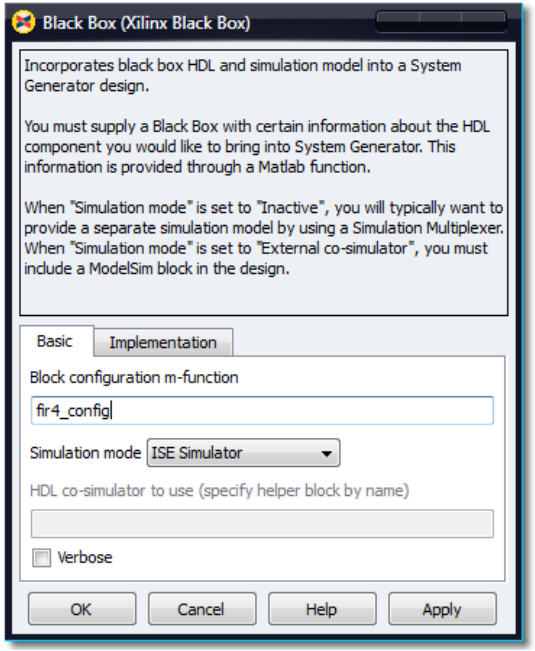


Рис. 6.51. Настройки функционального блока Black Box. Будет сформирован m-файл с именем fir4_config, задано функциональное моделирование в ISE

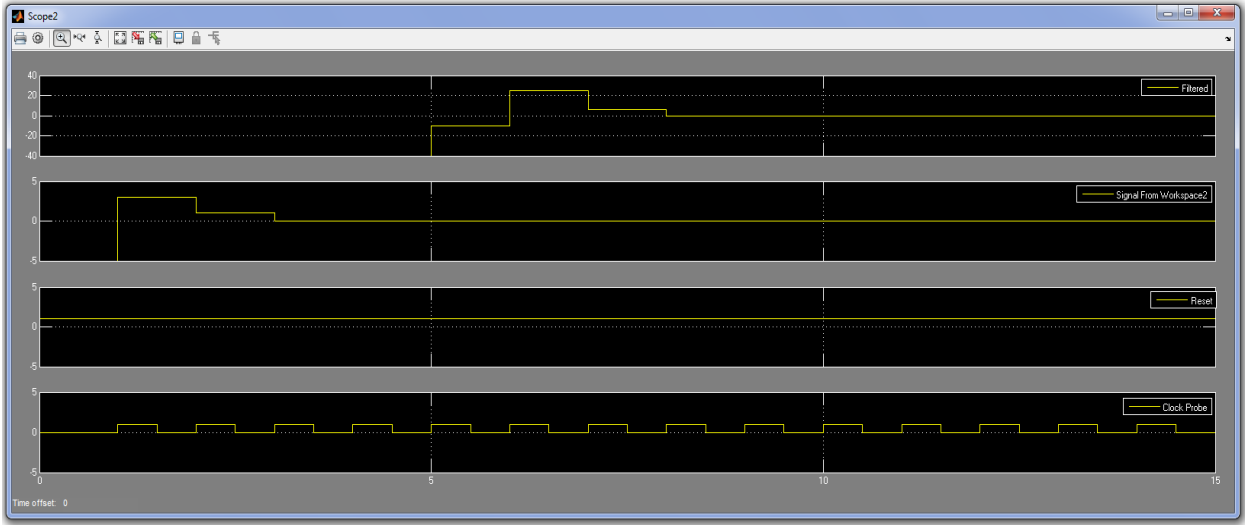
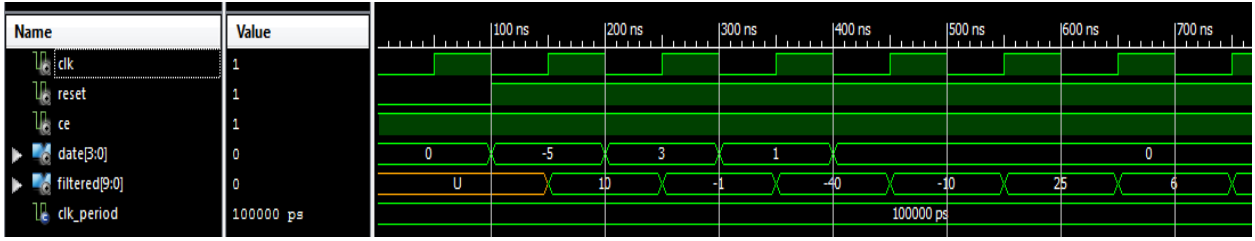
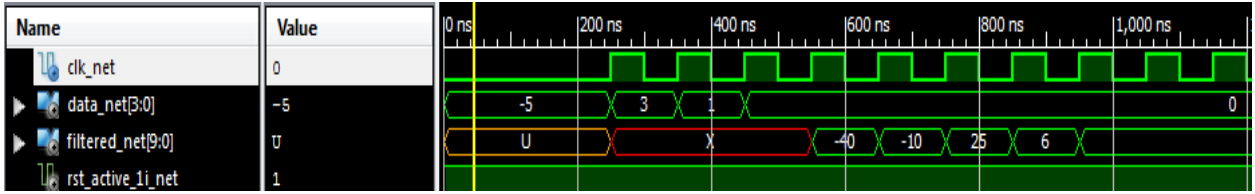


Рис. 6.52. Имитационное моделирование в системе Matlab/Simulink КИХ-фильтра на четыре отвода, созданного с использованием единственного VHDL-файла



а)



б)

Рис. 6.53. Функциональное моделирование КИХ-фильтра на четыре отвода в САПР ПЛИС Xilinx ISE Design Suite, созданного: а) на основе проекта, состоящего из единственного VHDL-файла (пример 1); б) на основе функционального блока Black Box

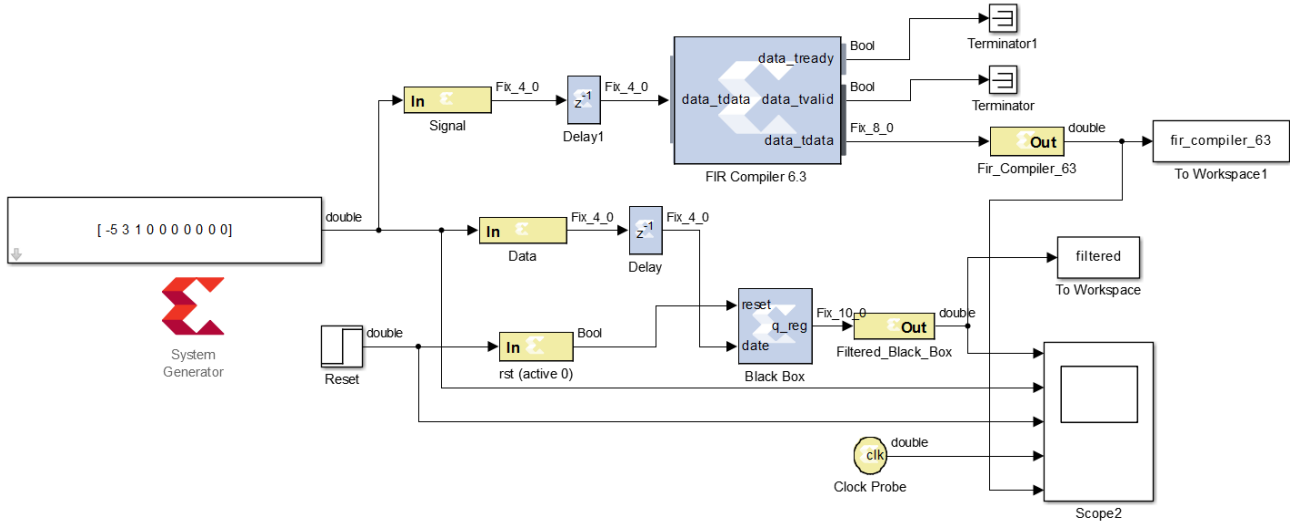
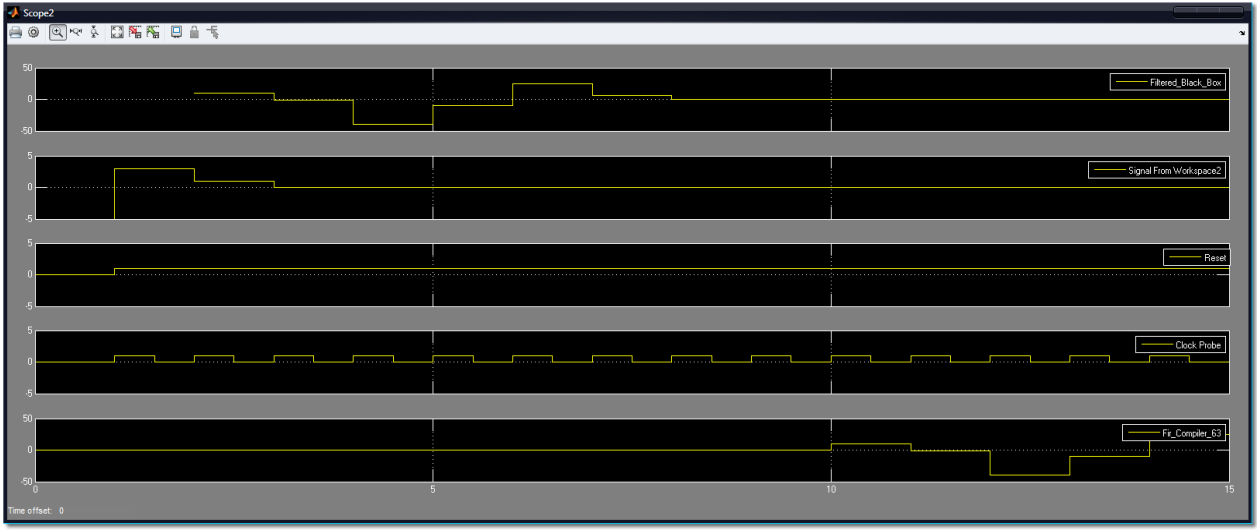
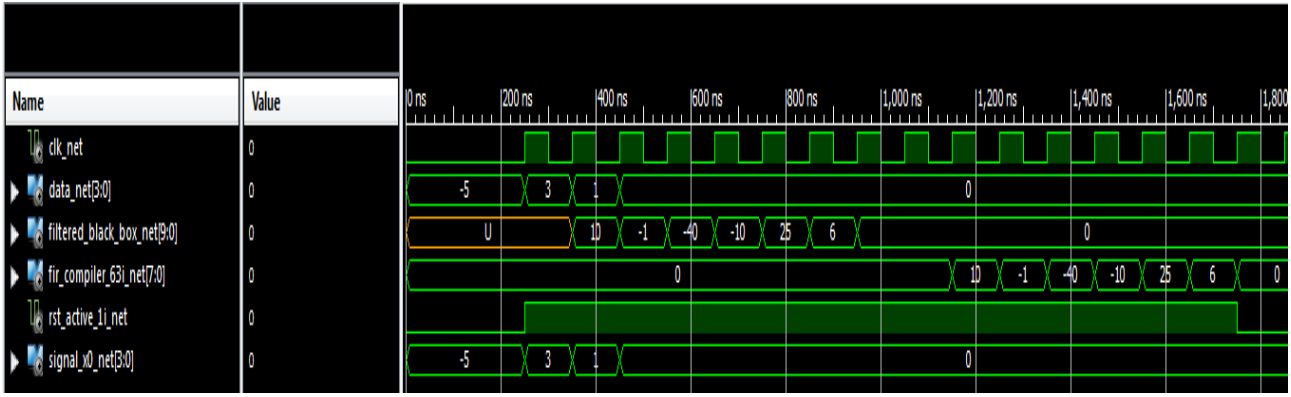


Рис. 6.54. Верификация двух имитационных моделей КИХ-фильтров, разработанных с использованием функционального блока FIR Compiler 6.3 и с применением VHDL-кода (функциональный блок Black Box)



а)

Рис. 6.55. Имитационное (а) и функциональное (б) моделирование двух моделей КИХ-фильтров, разработанных с использованием функционального блока FIR Compiler 6.3 Xilinx System Generator и с применением кода языка VHDL (функциональный блок Black Box)



б)

Рис. 6.55. Имитационное (а) и функциональное (б) моделирование двух моделей КИХ-фильтров, разработанных с использованием функционального блока FIR Compiler 6.3 Xilinx System Generator и с применением VHDL-кода (функциональный блок Black Box)

Таблица 6.3

Оценка ресурсов ПЛИС серии Spartan-6 xa6slx4-3tqg144 при реализации КИХ-фильтров на четыре отвода различными способами

Ресурсы ПЛИС	Параллельные структуры фильтров			Последовательная структура фильтра
	Имитационная модель в Simulink, Xilinx System Generator с применением Black Boxes (пример 1)	Проект в САПР ПЛИС Xilinx ISE Design Suite с применением функции FIR Compiler v6.3 и генератора XLogiCORE IP	Имитационная модель в Simulink, Xilinx System Generator с применением функционального блока FIR Compiler 6.3	Имитационная модель в Simulink, Xilinx System Generator, Dual Port Memory MAC FIR Filter, 1 MAC-блок
Максимальная частота, МГц (на этапе Post-Map)	114	334	305	279
Число ЦОС-блоков DSP-48A	3	1	4	1
Триггеров	8	48	91	57
Секций с LUT	6	33	56	38
Точность представления входного и профильтрованного сигнала, бит	FIX_4_0 и FIX_10_0	4 и 8	FIX_8_0 и FIX_20_8	FIX_4_0 и FIX_10_0

На рис. 6.54 показана верификация двух моделей КИХ-фильтров, разработанных с использованием функционального блока FIR Compiler 6.3 Xilinx System Generator, являющимся аналогом функции FIR Compiler v6.3 САПР Xilinx ISE получаемой с помощью генератора параметризированных ядер XLogiCORE IP и с применением VHDL-кода (функциональный блок Black Box).

В первом случае, функциональный блок FIR Compiler 6.3 настроен на реализацию систолического КИХ-фильтра, являющегося разновидностью параллельных структур.

На рис. 6.55 показаны результаты имитационного и функционального моделирования двух моделей КИХ-фильтров, разработанных с использованием функционального блока FIR Compiler 6.3 Xilinx System Generator и с применением кода языка VHDL. Для обоих моделей КИХ-фильтров испытательные стенды формируются в автоматическом режиме.

Анализ рис. 6.55 показывает, что КИХ-фильтр на основе систолической структуры характеризуется значительно большей латентностью появления профильтрованного сигнала (start-up latency) на выходе. Во втором случае, при синтезе кода по единственному VHDL-файлу формируется фильтр с наименьшей латентностью (рис. 6.55). Временные диаграммы работы характерны для параллельных структур КИХ-фильтров.

Оценка ресурсов ПЛИС серии Spartan-6 xa6slx4-3tqg144 при реализации КИХ-фильтров на четыре отвода различными способами представлена в таблице. Из табл. 6.3 видно, что наибольшим быстродействием обладают проекты, созданные с использованием либо генератора параметризированных ядер XLogiCORE IP или функциональных блоков Xilinx System Generator. Имитационная модель, созданная в Simulink с применением Xilinx System Generator и функционального блока Black Boxes, основанная на импорте VHDL-файла,

показала наихудшее быстродействие, но обеспечивает сбалансированное использование логических ресурсов (в сторону уменьшения) и ЦОС-блоков ПЛИС.

Импорт проектов созданных в САПР ПЛИС Xilinx ISE Design Suite с использованием структурного стиля языка VHDL. Проект будет состоять из двух VHDL-файлов. Низкого, файл fir4 и верхнего уровней иерархии fir. Файл верхнего уровня иерархии fir получим путем пересохранения файла fir.vhf, который извлекается из проекта (пример 1) в автоматическом режиме (HDL Functional Model), в файл fir.vhd (пример 3).

С помощью функционального блока Black Box создадим m-файл с именем fir_config (пример 4), указав на файл верхнего уровня иерархии fir.vhd (рис. 6.56). Рис. 6.50 и рис. 6.56 внешне схожи, за исключением имен портов, которые на рис. 6.56 соответствуют именам портов объекта fir (пример 3).

Пример 4 показывает, что необходимо отредактировать формат представления профильтрованного сигнала filtered с 'UFix_10_0' на 'Fix_10_0' и указать путь на файл низкого уровня иерархии fir4.vhd.

```
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric_std.ALL;
library UNISIM;
use UNISIM.Vcomponents.ALL;
entity fir is
  port ( ce      : in    std_logic;
         clk     : in    std_logic;
         date    : in    std_logic_vector (3 downto 0);
         reset   : in    std_logic;
         filtered : out   std_logic_vector (9 downto 0));
end fir;
architecture BEHAVIORAL of fir is
```

```
  component fir4
    port ( clk   : in    std_logic;
           reset : in    std_logic;
           ce    : in    std_logic;
           date  : in    std_logic_vector (3 downto 0);
           q_reg : out   std_logic_vector (9 downto 0));
  end component;
begin
  XLXI_1 : fir4
    port map (ce=>ce,
              clk=>clk,
              date(3 downto 0)=>date(3 downto 0),
              reset=>reset,
              q_reg(9 downto 0)=>filtered(9 downto 0));
end BEHAVIORAL;
```

Пример.3. Структурное описание КИХ-фильтра на 4 отвода (файл верхнего уровня иерархии)

```
function fir_config(this_block)
  this_block.setTopLevelLanguage('VHDL');
  this_block.setEntityName('fir');
  this_block.tagAsCombinational;
  this_block.addSimulinkInport('date');
  this_block.addSimulinkInport('reset');
  this_block.addSimulinkOutport('filtered');
  filtered_port = this_block.port('filtered');
  filtered_port.setType('Fix_10_0');
 …..
  this_block.addFile('fir4.vhd');
  this_block.addFile('fir.vhd');
return;
```

Пример 4. Фрагмент m-файла генерируемого Xilinx System Generator с учетом использования структурного стиля языка VHDL

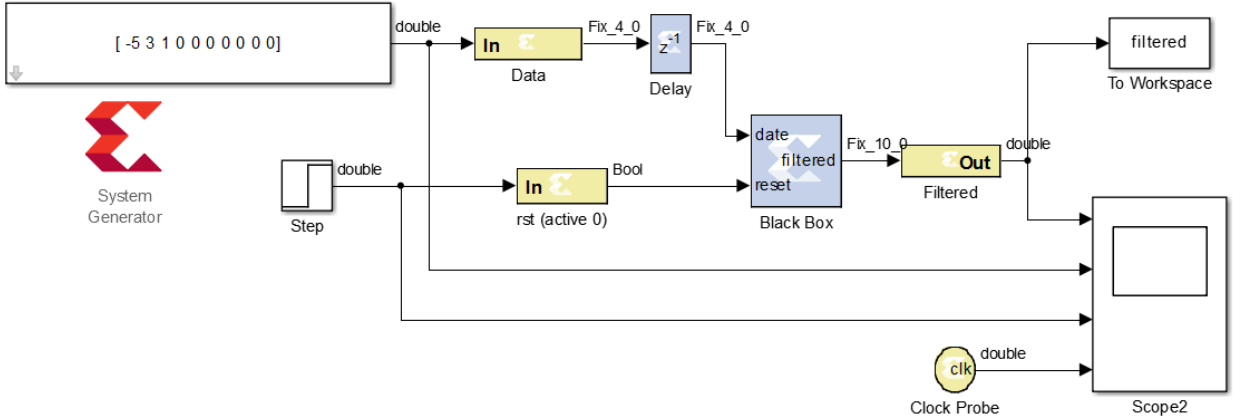


Рис. 6.56. Имитационная модель КИХ-фильтра на четыре отвода для реализации в базисе ПЛИС серии Spartan-6 xa6slx4-3tqg144 на основе функционального блока Black Box с использованием структурного стиля языка VHDL. m-файл с именем fir_config будет создан на основе файла верхнего уровня иерархии fir.vhd

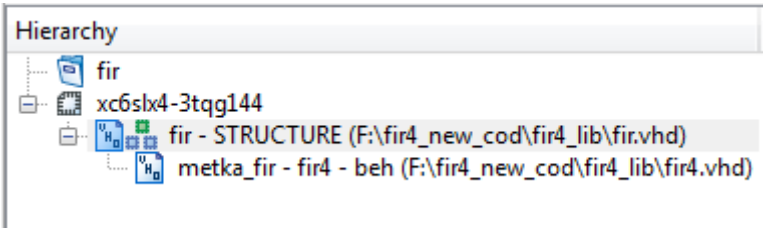
Импорт проектов, созданных в САПР ПЛИС Xilinx ISE Design Suite с использованием библиотечных элементов языка VHDL. В этом разделе также используется структурный стиль языка VHDL. Рассмотрим вариант разработки имитационной модели КИХ-фильтра с использованием библиотечных элементов (пример 5).

На рис. 6.57а показана иерархия проекта, состоящая из файла верхнего уровня иерархии fir.vhd и файла низкого уровня fir4.vhd, а на рис. 6.57б показано, что файл fir4.vhd размещен в дополнительно созданную библиотеку fir_lib, а файлы fir.vhd и test.vhd (пример 6) размещаются в библиотеку work. Пример 7 демонстрирует фрагмент изменений, который необходимо внести в m-файл fir4_config генерируемого Xilinx System Generator.

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
library fir_lib;
use fir_lib.all;
ENTITY fir IS PORT (
        date : IN std_logic_vector(3 DOWNTO 0);
        reset : IN std_logic;
        clk : IN std_logic;
        ce : IN std_logic;
        filtered : OUT std_logic_vector(9 DOWNTO 0)
);
END fir;
ARCHITECTURE STRUCTURE OF fir IS
COMPONENT fir4
        port ( clk   : in    std_logic;
         reset : in    std_logic;
         ce    : in    std_logic;
         date  : in    std_logic_vector (3 downto 0);
         q_reg : out   std_logic_vector (9 downto 0));
```

```
END COMPONENT;
BEGIN
metka_fir:      entity fir_lib.fir4
                port map (ce=>ce,
                clk=>clk,
                date(3 downto 0)=>date(3 downto 0),
                reset=>reset,
                q_reg(9 downto 0)=>filtered(9 downto 0));
END STRUCTURE;
```

Пример 5. Верхний уровень иерархии проекта с использованием библиотеки fir_lib



а)

б)

Рис. 6.57. а) Иерархия проекта КИХ-фильтра, состоящая из файла верхнего уровня иерархии fir.vhd и файла низкого уровня fir4.vhd; б) файл fir4.vhd размещен в библиотеку fir_lib, а файлы fir4.vhd и test.vhd размещаются в библиотеку work

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY test IS
```

```
END test;
ARCHITECTURE behavior OF test IS
   COMPONENT fir
   PORT(
        date : IN  std_logic_vector(3 downto 0);
        reset : IN  std_logic;
        clk : IN  std_logic;
        ce : IN  std_logic;
        filtered : OUT  std_logic_vector(9 downto 0)
       );
   END COMPONENT;
  --Inputs
  signal date : std_logic_vector(3 downto 0) := (others => '0');
  signal reset : std_logic := '0';
  signal clk : std_logic := '0';
  signal ce : std_logic := '1';
         --Outputs
  signal filtered : std_logic_vector(9 downto 0);
  -- Clock period definitions
  constant clk_period : time := 100 ns;
BEGIN
          -- Instantiate the Unit Under Test (UUT)
  uut: fir PORT MAP (
        date => date,
        reset => reset,
        clk => clk,
        ce => ce,
        filtered => filtered
       );
  -- Clock process definitions
  clk_process :process
  begin
                    clk <= '0';
                    wait for clk_period/2;
```

```
                clk <= '1';
                wait for clk_period/2;
  end process;
  -- Stimulus process
stim_proc: process
begin
-- hold reset state for 100 ns.
wait for 100 ns;
                reset <= '1';
  end process;
    tb : process
                begin
                        wait for 100 ns;
                        date <= "1011";
                        wait for 100 ns;
                        date <= "0011";
                        wait for 100 ns;
                        date <= "0001";
                        wait for 100 ns;
                        date <= "0000";
          wait;
  end process;
END;
```

Пример 6. Испытательный стенд на языке VHDL для моделирования прохождения сигнала –5, 3, 1, 0 по структуре КИХ-фильтра

```
this_block.addFile('fir.vhd');
 this_block.addFileToLibrary('fir4.vhd','fir_lib');
```

Пример 7. Фрагмент изменений, который необходимо внести в m-файла fir_config, генерируемого Xilinx System Generator

Импорт проектов, созданных в САПР ПЛИС Xilinx ISE Design Suite с использованием генератора параметризированных ядер XLogiCORE IP и функции FIR Compiler v6.3. На рис. 6.58а показан проект КИХ-фильтра на четыре отвода в САПР ПЛИС Xilinx ISE 14.4 с использованием генератора параметризированных ядер XLogiCORE IP FIR Compiler v6.3, а на рис. 6.58б – настройки функции FIR Compiler (закладка пять, поле «сигналы управления»).

Красным овалом на рис. 6.58 показана пара внешних портов clk и clken и пара портов aclk и aclken, принадлежащих символу fircore. Выбирается одноканальный, односкоростной (Single-Rate) систолический КИХ-фильтр прямой формы. Частота тактирования ядра фильтра установлена 250 МГц, а частота дискретизации – 50 МГц.

Далее разработаем испытательный стенд (пример 8). Характерной особенностью такого КИХ-фильтра на четыре отвода является латентность в одиннадцать тактов синхрочастоты (рис. 6.59).

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
LIBRARY UNISIM;
USE UNISIM.Vcomponents.ALL;
ENTITY fir4_sch_fir4_sch_sch_tb IS
END fir4_sch_fir4_sch_sch_tb;
ARCHITECTURE behavioral OF fir4_sch_fir4_sch_sch_tb IS
  COMPONENT fir4_sch
  PORT( s_tvalid       :       IN      STD_LOGIC;
    s_tready     :       OUT     STD_LOGIC;
    Data:        IN      STD_LOGIC_VECTOR (7 DOWNTO 0);
        clk      :       IN      STD_LOGIC;
        clken    :       IN      STD_LOGIC;
        m_tvalid :       OUT     STD_LOGIC;
 filtered:       OUT     STD_LOGIC_VECTOR (7 DOWNTO 0));
  END COMPONENT;
  SIGNAL s_tvalid      :       STD_LOGIC:= '1';
  SIGNAL s_tready      :       STD_LOGIC;
SIGNAL Data  :       STD_LOGIC_VECTOR (7 DOWNTO 0):=
(others => '0');
```

```
  SIGNAL clk  :        STD_LOGIC;
  SIGNAL clken        :        STD_LOGIC:= '1';
  SIGNAL m_tvalid     :        STD_LOGIC;
SIGNAL filtered : STD_LOGIC_VECTOR (7 DOWNTO 0);
BEGIN
  UUT: fir4_sch PORT MAP(
                s_tvalid => s_tvalid,
                s_tready => s_tready,
                Data => Data,
                clk => clk,
                clken => clken,
                m_tvalid => m_tvalid,
                filtered => filtered
  );
clk_process :process
                                begin
                                        clk <= '0';
                                        wait for 10 ns;
                                        clk <= '1';
                                        wait for 10 ns;
  end process;
        tb : process
                begin
                        wait for 20 ns;
                        Data <= "11111011";
                        wait for 20 ns;
                        Data <= "00000011";
                        wait for 20 ns;
                        Data <= "00000001";
                        wait for 20 ns;
                        Data <= "00000000";
        wait;
        END process;
END;
```

Пример 8. Испытательный стенд на языке VHDL для моделирования прохождения сигнала –5, 3, 1, 0 по структуре КИХ-фильтра, созданного с использованием генератора параметризированных ядер XLogiCORE IP и функции FIR Compiler v6.3

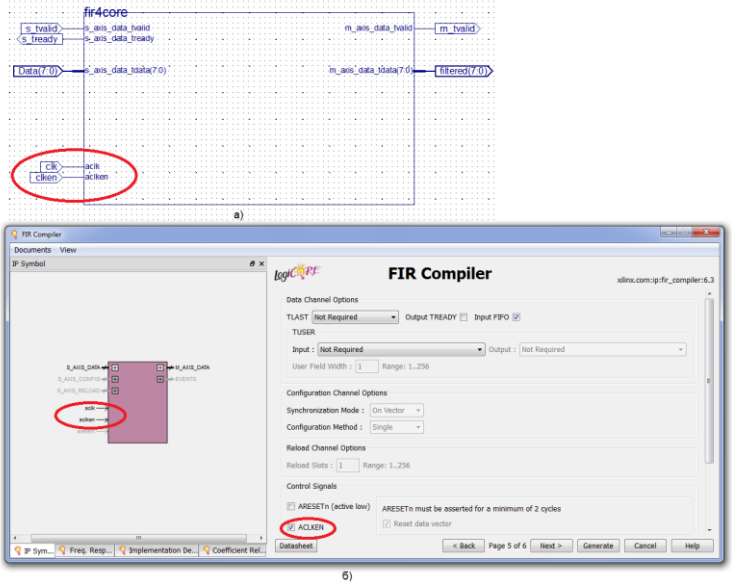


Рис. 6.58. а) Проект КИХ-фильтра на 4 отвода в САПР ПЛИС Xilinx ISE 14.2 с использованием генератора параметризированных ядер XLogiCORE IP FIR Compiler v6.3; б) настройки функции FIR Compiler, опционально подключается сигнал разрешения подачи синхросигнала aclken

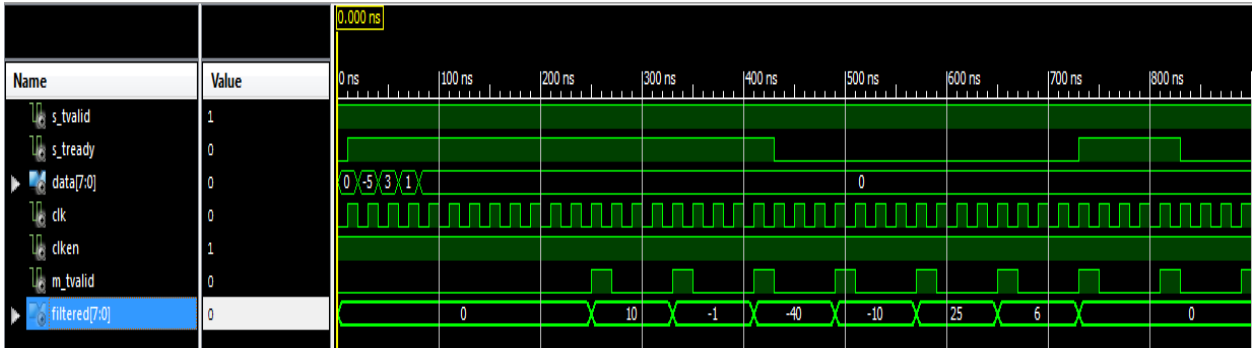


Рис. 6.59. Временные диаграммы работы КИХ-фильтра на четыре отвода, созданного с использованием генератора параметризированных ядер XLogiCORE IP и функции FIR Compiler v6.3. На вход КИХ-фильтра поступают входные отсчеты –5, 3, 1 и 0. Правильные значения на выходе фильтра: 10, –1, –40, –10, 26, 6

Убедившись в том, что КИХ-фильтр работает корректно пересохраним файл fir4_sch.vhf в файл fir4_sch.vhd (пример 9) и отредактируем его так, чтобы в нем использовалась пара сигналов с именами clk и ce (пример 10). Проанализируем полученный файл. Файл fir4_sch.vhd создан на основе компоненты fir4core, которая разработана с применением генератора параметризированных ядер XLogiCORE IP и функции FIR Compiler v6.3, также является разновидностью структурного стиля языка VHDL.

Генератор параметризированных ядер в процессе своей работы создает следующие файлы: fir4core.ngc (синтезируемая модель КИХ-фильтра представленная списком связей на низком логическом уровне в двоичном формате, создается с помощью специального приложения Xilinx Synthesis Technology (XST), входящего в состав ISE), fir4core.mif (коэффициенты фильтра в HEX-формате) и fir4core.vhd (спецификация фильтра).

Далее разработаем имитационную модель на основе функционального блока Black Box с использованием файла fir4_sch.vhd (рис. 6.60 и рис. 6.61). На рис. 6.60 также показано, что профильтрованный сигнал подвергается децимации. Блок децимации выполняет роль компрессора. В выходном сигнале сохраняются отсчеты с номерами кратными четырем. Такой прием позволяет сэкономить вычислительные ресурсы ПЛИС. Функциональный блок Black Box сгенерирует m-файл с именем fir4_sch_config (пример 11), который должен быть отредактирован. В него должны быть добавлены ссылки помимо файла fir4_sch.vhd файлы fir4core.ngc, fir4core.mif и fir4core.vhd.

```
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric_std.ALL;
library UNISIM;
use UNISIM.Vcomponents.ALL;
```

```
entity fir4_sch is
  port ( clk      : in    std_logic;
         clken    : in    std_logic;
         Data     : in    std_logic_vector (7 downto 0);
         s_tvalid : in    std_logic;
         filtered : out   std_logic_vector (7 downto 0);
         m_tvalid : out   std_logic;
         s_tready : out   std_logic);
end fir4_sch;
architecture BEHAVIORAL of fir4_sch is
  component fir4core
    port ( s_axis_data_tvalid : in    std_logic;
           s_axis_data_tready : out   std_logic;
           s_axis_data_tdata  : in    std_logic_vector (7 downto 0);
           aclk               : in    std_logic;
           aclken             : in    std_logic;
           m_axis_data_tvalid : out   std_logic;
           m_axis_data_tdata  : out   std_logic_vector (7 downto 0));
  end component;
begin
  XLXI_1 : fir4core
    port map (aclk=>clk,
              aclken=>clken,
              s_axis_data_tdata(7 downto 0)=>Data(7 downto 0),
              s_axis_data_tvalid=>s_tvalid,
              m_axis_data_tdata(7 downto 0)=>filtered(7 downto 0),
              m_axis_data_tvalid=>m_tvalid,
              s_axis_data_tready=>s_tready);
end BEHAVIORAL;
```

Пример.9. Файл fir4_sch.vhd, пересохраненный из файла fir4_sch.vhf (HDL Functional Model)

```
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric_std.ALL;
library UNISIM;
use UNISIM.Vcomponents.ALL;
```

```
entity fir4_sch is
  port ( clk      : in    std_logic;
         ce    : in    std_logic;
         Data     : in    std_logic_vector (7 downto 0);
         s_tvalid : in    std_logic;
         filtered : out   std_logic_vector (7 downto 0);
         m_tvalid : out   std_logic;
         s_tready : out   std_logic);
end fir4_sch;

architecture BEHAVIORAL of fir4_sch is
  component fir4core
    port ( s_axis_data_tvalid : in    std_logic;
           s_axis_data_tready : out   std_logic;
           s_axis_data_tdata  : in    std_logic_vector (7 downto 0);
           aclk               : in    std_logic;
           aclken             : in    std_logic;
           m_axis_data_tvalid : out   std_logic;
           m_axis_data_tdata  : out   std_logic_vector (7 downto 0));
  end component;

begin
  XLXI_1 : fir4core
    port map (aclk=>clk,
              aclken=>ce,
              s_axis_data_tdata(7 downto 0)=>Data(7 downto 0),
              s_axis_data_tvalid=>s_tvalid,
              m_axis_data_tdata(7 downto 0)=>filtered(7 downto 0),
              m_axis_data_tvalid=>m_tvalid,
              s_axis_data_tready=>s_tready);
end BEHAVIORAL;
```

Пример.10. Отредактированный файл fir4_sch.vhd, в котором используется пара сигналов с именами clk и ce

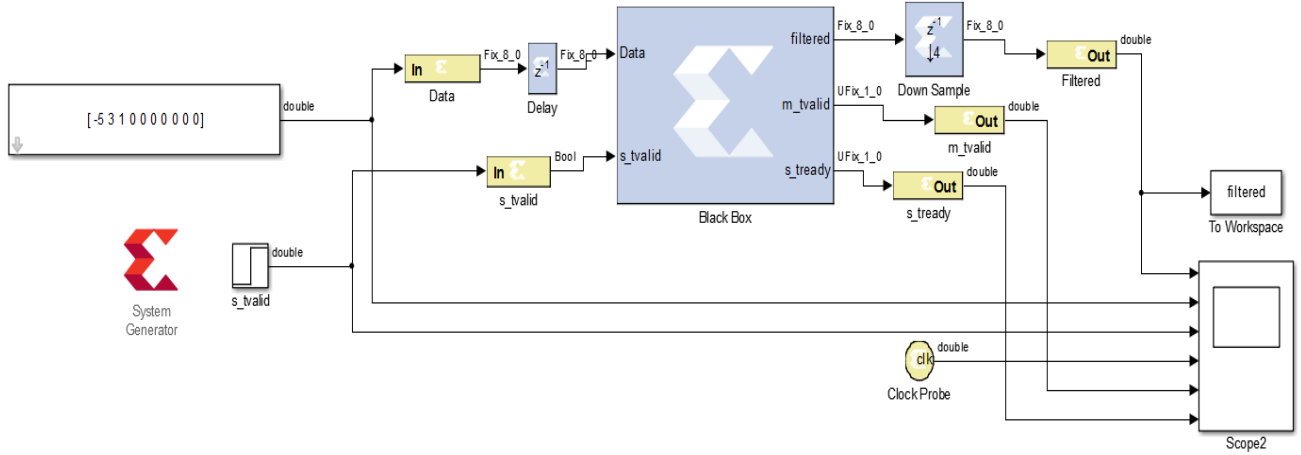


Рис. 6.60. Имитационная модель КИХ-фильтра на четыре отвода для реализации в базисе ПЛИС серии Spartan-6 xa6slx4-3tqg144 на основе функционального блока Black Box с использованием файла fir4_sch.vhd, созданного на основе компоненты fir4core

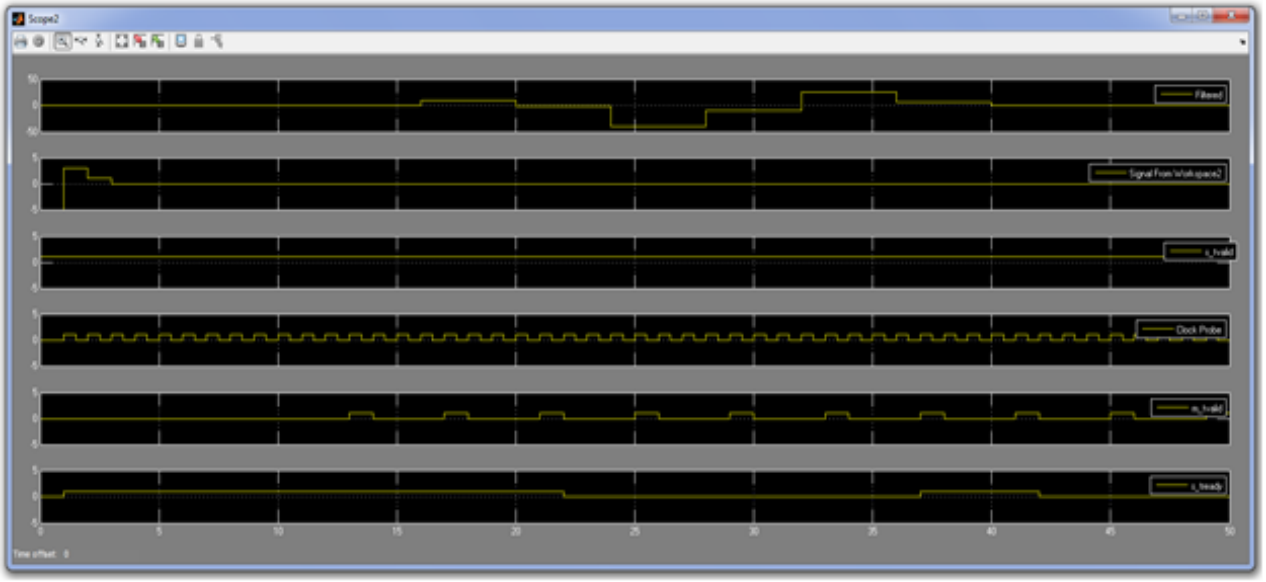


Рис. 6.61. Имитационное моделирование в системе Matlab/Simulink КИХ-фильтра на четыре отвода на основе функционального блока Black Box с использованием файла fir4_sch.vhd, созданного на основе компоненты fir4core

```
function fir4_sch_config(this_block)
  this_block.setTopLevelLanguage('VHDL');
  this_block.setEntityName('fir4_sch');
  this_block.tagAsCombinational;
  this_block.addSimulinkInport('Data');
  this_block.addSimulinkInport('s_tvalid');
  this_block.addSimulinkOutport('filtered');
  this_block.addSimulinkOutport('m_tvalid');
  this_block.addSimulinkOutport('s_tready');
  filtered_port = this_block.port('filtered');
  filtered_port.setType('Fix_8_0');
  m_tvalid_port = this_block.port('m_tvalid');
  m_tvalid_port.setType('UFix_1_0');
  m_tvalid_port.useHDLVector(false);
  s_tready_port = this_block.port('s_tready');
  s_tready_port.setType('UFix_1_0');
  s_tready_port.useHDLVector(false);
  if (this_block.inputTypesKnown)
    if (this_block.port('Data').width ~= 8);
      this_block.setError('Input data type for port "Data" must have
width=8.');
    end
    if (this_block.port('s_tvalid').width ~= 1);
      this_block.setError('Input data type for port "s_tvalid" must
have width=1.');
    end
    this_block.port('s_tvalid').useHDLVector(false);
  end
   if (this_block.inputRatesKnown)
     setup_as_single_rate(this_block,'clk','ce')
   end
    uniqueInputRates = unique(this_block.getInputRates);
  this_block.addFile('fir4core.vhd');
  this_block.addFile('fir4core.ngc');
  this_block.addFile('fir4core.mif');
  this_block.addFile('fir4_sch.vhd');
return;
function setup_as_single_rate(block,clkname,cename)
```

```
  inputRates = block.inputRates;
  uniqueInputRates = unique(inputRates);
  if (length(uniqueInputRates)==1 & uniqueInputRates(1)==Inf)
    block.addError('The inputs to this block cannot all be constant.');
    return;
  end
  if (uniqueInputRates(end) == Inf)
     hasConstantInput = true;
     uniqueInputRates = uniqueInputRates(1:end-1);
  end
  if (length(uniqueInputRates) ~= 1)
    block.addError('The inputs to this block must run at a single
rate.');
    return;
  end
  theInputRate = uniqueInputRates(1);
  for i = 1:block.numSimulinkOutports
     block.outport(i).setRate(theInputRate);
  end
  block.addClkCEPair(clkname,cename,theInputRate);
  return;
```

Пример 11. Отредактированный m-файл созданный на основе файла fir4_sch.vhd

Использование методологии Black Boxes Xilinx System Generator значительно повышает возможности объектно-ориентированного проектирования при разработке имитационных моделей цифровых устройств для реализации в базисе ПЛИС, так как позволяет импортировать коды высокоуровневых языков описания аппаратурных средств из САПР Xilinx ISE в систему M*atlab*/Simulink, полученных с использованием различных приемов.

ЗАКЛЮЧЕНИЕ

В учебном пособии на обширном иллюстративном материале показаны методы обработки цифровых сигналов в базисе ПЛИС с учетом их архитектурных особенностей с применением высокоуровневого языка описания аппаратных средств.

Показаны пример расчета спецификации КИХ-фильтра, эффекты квантования при работе в формате с фиксированной запятой, представлены методики вычисления коэффициентов в формате с фиксированной запятой. Продемонстрировано имитационное моделирование КИХ-фильтра в системе M*atlab*/Simulink с переходом на функциональные модели в САПР ПЛИС.

Демонстрируются различные варианты реализации параллельных и последовательных КИХ-фильтров в базисе ПЛИС с использованием умножителей и MAC-блоков на мегафункциях САПР Quartus II компании Altera.

Даются практические примеры проектирования КИХ-фильтров на последовательной и параллельной распределенной арифметике в САПР ПЛИС Altera Quartus II и Xilinx ISE Design Suite.

Показано, что систолический КИХ-фильтр является оптимальным решением для параллельных архитектур цифровых фильтров, позволяет существенно уменьшить число используемых ресурсов и повысить быстродействие системы.

Уделено внимание методологии объектно-ориентированного проектирования с использованием программных пакетов расширений Xilinx System Generator IDS и Altera DSP Builder в системе визуально-имитационного моделирования M*atlab*/Simulink.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Computer Arithmetic: Algorithms and Hardware Designs (Oxford U. Press, 2nd ed., 2010, ISBN 978-0-19-532848-6).
2. Кестнер У. Проектирование систем цифровой и смешанной обработки сигналов. – М.: Техносфера, 2010. 328 с.
3. Uwe Meyer-Baese. Digital Signal Processing with Field Programmable Gate Arrays. Fourth Edition. http://www.springer.com/series/4748.
4. Xilinx. DSP: Designing for Optimal Results High-Performance DSP Using Virtex-4 FPGAs. DSP Products Advanced Design Guide. Edition 1.0. March 2005.
5. Максфилд К. Проектирование на ПЛИС: курс молодого бойца [Текст] : пер. с англ. / К. Максфилд. – М.: Издательский дом Додэка XXI, 2007. 408 с.
6. Айфичер Э. Цифровая обработка сигналов: практический подход, 2-е изд. [Текст] : пер. с англ. / С. Эммануил Айфичер, У. Барри Джервис. – М.: Издательский дом "Вильямс", 2004. 992 с.
7. Смит Стивен. Цифровая обработка сигналов. Практическое руководство для инженеров и научных работников [Текст] : пер. с англ. / Стивен Смит. – М.: Издательский дом Додэка XXI, 2012. 720 с.
8. Сергиенко А. Б. Цифровая обработка сигналов [Текст]: учеб. пособие. – 3-е изд. – СПб.: БХВ-Петербург, 2011. 768 с.
9. Сато, Юкио. Цифровая обработка сигналов [Текст] / Юкио Сато : пер. с яп. – М.: Издательский дом Додэка XXI, 2010. 176 с.
10. Стешенко, В. Б. ПЛИС фирмы ALTERA: проектирование устройств обработки сигналов [Текст] / В. Б. Стешенко. – М.: Додэка, 2000. – 457 с.
11. Ефремов, Н. В. Введение в систему автоматизированного проектирования Quartus II [Текст] / Н. В. Ефремов. – М.: ГОУ ВПО МГУЛ, 2011. – 147 с.

12. Строгонов, А. В. Цифровая обработка сигналов в базисе программируемых логических интегральных схем [Текст]: Учебное пособие. / А. В. Строгонов. – 3-е изд., испр. и доп. – СПб.: Лань, 2018. – 312 с.

13. Строгонов, А. В. Проектирование цифровых фильтров в системе M*atlab*/Simulink и САПР ПЛИС Quartus [Текст] / А. В. Строгонов // Компоненты и технологии. – 2008. – № 6. – С. 122– 126.

14. Строгонов, А. В. Эффективность разработки конечных автоматов в базисе ПЛИС FPGA [Текст] / А. В. Строгонов, А. В. Быстрицкий // Компоненты и технологии. – 2013. – № 1.– С. 66–72.

15. Строгонов, А. В. КИХ-фильтр на распределенной арифметике: проектируем сами [Текст] / А. В. Строгонов, А. В. Быстрицкий // Компоненты и технологии. – 2013. – № 3. – С. 131–138.

16. Строгонов, А. В. КИХ-фильтры на параллельной распределенной арифметике: проектируем сами [Текст] / А. В. Строгонов, А. В. Быстрицкий // Компоненты и технологии. 2013. – № 5. – С. 84–88.

17. Строгонов, А. В. Проектирование параллельных КИХ-фильтров в базисе ПЛИС [Текст] / А. В. Строгонов, А. В. Быстрицкий // Компоненты и технологии. – 2013. – № 6.– С. 75–80.

18. Строгонов, А. В. Систолические КИХ-фильтры в базисе ПЛИС [Текст] / А. В. Строгонов, А. В. Быстрицкий // Компоненты и технологии. – 2013. – № 8. – С. 79–84.

19. Строгонов, А. В. Проектирование систолических КИХ-фильтров в базисе ПЛИС с помощью системы моделирования ModelSim-Altera [Текст] / А. В. Строгонов, А. В. Быстрицкий // Компоненты и технологии. – 2013. – № 9. – С. 58–62.

20. Строгонов, А. В. Проектирование умножителя методом правого сдвига и сложения с управляющим автоматом в базисе

ПЛИС [Текст] / А. В. Строгонов, А. В. Быстрицкий // Компоненты и технологии. – 2013. – № 12. – С. 52–56.

21. Строгонов, А. В. Проектирование умножителя целых чисел со знаком методом правого сдвига и сложения в базисе ПЛИС [Текст] / А. В. Строгонов, А. В. Быстрицкий // Компоненты и технологии. – 2014. – № 1. – С. 145–151.

22. Строгонов, А. В. Проектирование КИХ-фильтра на умножителе методом правого сдвига и сложения в базисе ПЛИС [Текст] / А. В. Строгонов, А. В. Быстрицкий // Компоненты и технологии. – 2014. – № 3. – С. 63–68.

23. Строгонов, А. В. Проектирование КИХ-фильтров с учетом архитектурных особенностей ПЛИС [Текст] / А. В. Строгонов, А. В. Быстрицкий // Компоненты и технологии. – 2014. – № 8. – С. 122–127.

24. Строгонов, А. В. Проектирование КИХ-фильтров в САПР ПЛИС Xilinx ISE Design Suite [Текст] / А. В. Строгонов, С. А. Цыбин, П. С. Городков // Компоненты и технологии. – 2014. – № 11. – С. 96–102.

25. Строгонов А. В. Проектирование КИХ-фильтров на распределенной арифметике в САПР ПЛИС Xilinx ISE Design Suite [Текст] / А. В. Строгонов, С. А. Цыбин, П. С. Городков // Компоненты и технологии. – 2015. – № 2. – С. 38–43.

26. Строгонов А. В. Разработка КИХ-фильтров в системе Xilinx System Generator САПР ISE Design Suite. [Текст] / А. В. Строгонов, С. А. Цыбин, П. С. Городков // Компоненты и технологии. – 2015. – № 5. – С. 49–58.

27. Строгонов А. В. Проектирование последовательных КИХ-фильтров в системе Xilinx System Generator с применением библиотеки Reference BlockSet/DSP. [Текст] / А. В. Строгонов, С. А. Цыбин, П. С. Городков // Компоненты и технологии – 2015. – № 6. – С. 86–92.

28. Строгонов А. В. Проектирование КИХ-фильтров в системе Xilinx System Generator с применением методологии Black Boxes. [Текст] / А. В. Строгонов, С. А. Цыбин,

П. С. Городков // Компоненты и технологии. – 2015. – № 7. – С. 62–68

29. Строгонов А. В. Проектирование последовательных КИХ-фильтров в системе визуально-имитационного моделирования M*atlab*/Simulink с использованием Altera DSP Builder. [Текст] / А. В. Строгонов, С. А. Цыбин, П. С. Городков // Компоненты и технологии. – 2015. – № 11. – С. 89–94.

30. Строгонов А.В. Проектирование КИХ-фильтров в системе визуально-имитационного моделирования M*atlab*/Simulink с использованием Altera DSP Builder. [Текст] / А. В. Строгонов, С. А. Цыбин, П. С. Городков // Компоненты и технологии. – 2015. – № 12. – С. 25–32.

31. Строгонов А. В. Проектирование последовательных КИХ-фильтров в САПР ПЛИС Quartus II [Текст] / А. В. Строгонов, С. А. Цыбин, П. С. Городков // Компоненты и технологии. – 2016. – № 1. – С. 10–15.

32. Строгонов А. В. Особенности использования двухпортовой памяти при проектировании последовательных КИХ-фильтров в САПР ПЛИС Quartus II [Текст] / А. В. Строгонов, С. А. Цыбин, П. С. Городков // Компоненты и технологии. – 2016. – № 4. – С. 124–130.

33. Строгонов А. В. Проектирование цифровых устройств обработки сигналов в системе визуально-имитационного моделирования *Matlab*/Simulink с использованием Altera DSP Builder [Текст] / А. В. Строгонов, С. А. Цыбин, П. С. Городков // Компоненты и технологии. – 2018. – № 12. – С. 25–30.

34. Строгонов А. В. Проектирование КИХ-фильтра на распределенной арифметике в системе визуально-имитационного моделирования *Matlab*/Simulink с использованием Altera DSP Builder [Текст] / А. В. Строгонов, П. С. Городков // Компоненты и технологии. – 2019. – № 2. – С. 106–112.

35. Строгонов А. В. Как устроены «настоящие» КИХ-фильтры на последовательной распределенной арифметике

[Текст] / А. В. Строгонов, П. С. Городков // Компоненты и технологии. – 2019. – № 4. – С. 103–110.

36. Строгонов А. В. Представление коэффициентов КИХ-фильтра в формате с фиксированной запятой [Текст] / А. В. Строгонов // Компоненты и технологии. – 2020. – № 5. – С. 86–90.

37. Строгонов А. В. Применение Altera Dsp Builder системы Matlab / Simulink для разработки имитационной модели КИХ-фильтра на параллельной распределенной арифметике [Текст] / А. В. Строгонов // Электроника: Наука, Технология, Бизнес.– 2020. – № 3. – С. 176–188.

38. Строгонов А. В. Диверсификация проекта последовательного КИХ-фильтра в базисе [Текст] / А. В. Строгонов, И. В. Семейкин // Электроника: Наука, Технология, Бизнес.– 2022. – № 6. – С. 108–123.

ОГЛАВЛЕНИЕ

Андрей Владимирович СТРОГОНОВ

РЕАЛИЗАЦИЯ АЛГОРИТМОВ ЦИФРОВОЙ ОБРАБОТКИ СИГНАЛОВ В БАЗИСЕ ПРОГРАММИРУЕМЫХ ЛОГИЧЕСКИХ ИНТЕГРАЛЬНЫХ СХЕМ

УЧЕБНОЕ ПОСОБИЕ

Издание шестое, исправленное и дополненное