ФГБОУ ВО «Воронежский государственный
технический университет»

А.В. Строгонов

РЕАЛИЗАЦИЯ АЛГОРИТМОВ ЦИФРОВОЙ ОБРАБОТКИ СИГНАЛОВ В БАЗИСЕ ПРОГРАММИРУЕМЫХ ЛОГИЧЕСКИХ ИНТЕГРАЛЬНЫХ СХЕМ

Утверждено Редакционно-издательским советом
университета в качестве учебного пособия

Воронеж 2016

УДК 621.3.049.77

Строгонов А.В. Реализация алгоритмов цифровой обработки сигналов в базисе программируемых логических интегральных схем: учеб. пособие [Электронный ресурс]. – Электрон. текстовые и граф. данные (9,5 Мб). – Воронеж: ФГБОУ ВО «Воронежский государственный технический университет», 2016. – 1 электрон. опт. диск (DVD-ROM): цв. – Систем. требования: ПК 500 и выше; 256 Мб ОЗУ; Windows XP; SVGA с разрешением 1024x768; Adobe Acrobat ; CD -ROM дисковод; мышь. – Загл. с экрана.

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, мегаядер и мегафункций САПР ПЛИС Altera Quartus II, генератора параметризированных ядер САПР ПЛИС Xilinx ISE Design Suite.

Издание соответствует требованиям Федерального государственного образовательного стандарта высшего образования по направлению подготовки 11.04.04 «Электроника и наноэлектроника» (направленность «Приборы и устройства в микро- и наноэлектронике»), дисциплинам «Цифровая обработка сигналов», «Архитектуры микропроцессорных вычислительных систем», «САПР БИС программируемой логики», «САПР системного уровня проектирования БИС».

Табл. 17. Ил. 244. Библиогр.: 35 назв.

Научный редактор д-р физ.-мат. наук, проф. С.И. Рембеза

Рецензенты: кафедра физики полупроводников и микроэлектроники Воронежского государственного университета (зав. кафедрой д-р физ.-мат. наук, проф. Е.Н. Бормонтов);
д-р техн. наук, доц. М.А. Ромащенко

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.

В первой главе рассматриваются основы двоичной арифметики, представление чисел со знаком, арифметические операции. Основное внимание уделено проектированию умножителей чисел со знаком, представленных в дополнительном коде. Приводятся сведения по программным умножителям в базисе ПЛИС. Дается практический пример по использованию учебного лабораторного стенда LESO2.1 (Лаборатории электронных средств обучения, ЛЭСО ГОУ

ВПО «СибГУТИ») для отладки проекта умножителя целых положительных чисел представленных в прямом коде размерностью 4х4 методом правого сдвига и сложения в базисе ПЛИС серии Cyclone.

Во второй главе рассматривается моделирование КИХ-фильтра в системе Matlab/Simulink (пакет Signal Processing, среда FDATool). Обсуждаются вопросы проектирования с учетом эффектов квантования возникающих при переходе от имитационной модели КИХ-фильтра разработанной в системе Matlab/Simulink к функциональной реализованной в САПР ПЛИС Quartus II компании Altera. Демонстрируются различные варианты реализации параллельных и последовательных КИХ-фильтров с использованием перемножителей на мегафункциях САПР Quartus II компании Altera.

В главе 3 затрагиваются вопросы проектирования высокопроизводительных КИХ-фильтров на последовательной и параллельной распределенных арифметиках учитывающих архитектурные особенности ПЛИС с использованием генераторов параметризированных ядер и мегафункций. В четвертой главе рассматривается проектирование систолических КИХ-фильтров в базисе ПЛИС с использованием системы цифрового моделирования ModelSim-Altera.

В пятой и шестой главах уделено внимание методологии объектно-ориентированного проектирования с использованием программных пакетов расширений Xilinx System Generator IDS 14.4 и Altera DSP Builder в системе визуально-имитационного моделирования Matlab/Simulink. Программные пакеты обеспечивают высокоуровневое представление систем цифровой обработки сигналов, абстрагированное от конкретной аппаратной платформы, с возможностью автоматической компиляции в базис ПЛИС.

1. АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ

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:

$$\begin{array}{lll} -3 & 101 & (-2^2 + 2^0) \\ 3 & 011 & (2^1 + 2^0) \end{array},$$

в четырехразрядном представлении:

$$\begin{array}{ll} 1101 & (-2^3 + 2^2 + 2^0) \\ 0011 & (2^1 + 2^0) \end{array}$$

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.

На рис. 1.1 представлено 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.

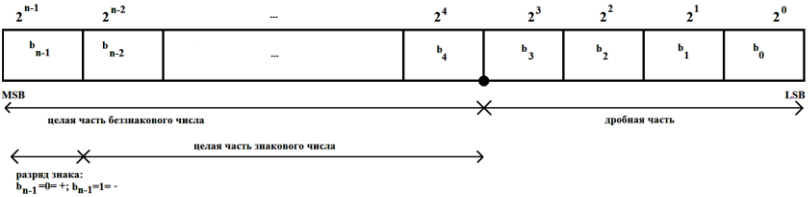


Рис. 1.1. Формат машинного слова

В файле помощи Fixed-Point Blockset системы Matlab 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: $S=Fx2^E$, где F - наклон дробной части, нормализованная величина $1 \le F < 2$; E - показатель степени E = -m. При проектировании устройств цифровой обработки сигналов принимают B = 0 и F = 1:

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

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 беззнаковое (MSB = 0) вещественное число 3.3125:

$$3.3125 = 2^{-4}\begin{pmatrix} 0*2^7 + 0*2^6 + 1*2^5 + 1*2^4 + 0*2^3 + \\ +1*2^2 + 0*2^1 + 1*2^0 \end{pmatrix}.$$

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

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

1.3. Сумматоры/вычитатели

Рассмотрим схему сумматора, основанного на поразрядном процессе. Обозначим два складываемых числа через $A = a_{n-1}a_{n-2}\ldots a_1a_0$ и $B = b_{n-1}b_{n-2}\ldots b_1b_0$. При сложении двоичных чисел значения цифр в каждом двоичном разряде должны быть сложены между собой с переносом из предыдущего разряда. Если результат при этом превышает 1, то возникает перенос в следующий разряд.

В логической схеме информацию о переносе в разряд i можно представить в виде бита переноса c_i, равного 1, если перенос из предыдущего разряда есть, и 0 – в противном случае. Операция, которую нужно выполнить для каждого разряда i, будет заключаться в сложении трех битов a_i, b_i и c_i, получении значения бита суммы s_i и переноса в следующий разряд c_{i+1}. Фактически c_{i+1} и s_i представляют старший и младший разряды в двухразрядной сумме битов a_i, b_i и c_i.

По функциям, заданным в табл. 1.2. Можно построить логические выражения канонические суммы минтермов для суммы s_i и переноса c_{i+1}:

$$s_i = \bar{a}_i\bar{b}_i\bar{c}_i + \bar{a}_ib_i\bar{c}_i + a_i\bar{b}_i\bar{c}_i + a_ib_ic_i = a_i \oplus b_i \oplus c_i.$$

$$c_{i+1} = \bar{a}_ib_ic_i + a_i\bar{b}_ic_i + a_ib_i\bar{c}_i + a_ib_ic_i = a_ib_i + a_ic_i + b_ic_i$$

Таблица 1.2

Определение битов суммы s_i и переноса c_{i+1} по значениям a_i, b_i и c_i при сложении

a_i	b_i	c_i	Сумма a_i, b_i и c_i	c_{i+1}	s_i
0	0	0	00	0	0
0	0	1	01	0	1
0	1	0	01	0	1
0	1	1	10	1	0
1	0	0	01	0	1
1	0	1	10	1	0
1	1	0	10	1	0
1	1	1	11	1	1

На рис. 1.2 показана схема, реализующая выражения для суммы s_i и переноса c_{i+1}. Эта схема называется полным сумматором, т.к. она суммирует три бита в разряде, включая перенос. Покаскадно соединив n полных сумматоров можно построить схему для сложения n-разрядных чисел. Разряды двух складываемых чисел подаются на входы a_i и b_i, а результат появляется на выходах s_i. Последний перенос c_n является старшим разрядом (n+1) – разрядной суммы. Входная линия переноса в младший разряд c_0 является еще одним входом всей схемы. Он позволяет задать начальное значение переноса, что удобно для сложения с многократной точностью.

Сумматоры на схеме рис. 1.3 называют сумматорами с последовательным переносом из-за наличия в них последовательной зависимости от переносов. Каждый полный сумматор в цепочке распространения переносов вносит задержку в двух логических уровнях. Поэтому конечный перенос c_n, зависящий от самых правых разрядов a_0, b_0 и c_0, проходит через 2n логических уровней. Таким образом, сумматор с последовательным переносом работает

существенно медленнее параллельного сумматора, в котором всего 2 логических уровня.

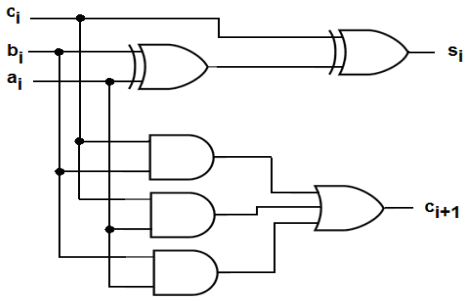


Рис. 1.2. Полный одноразрядный сумматор

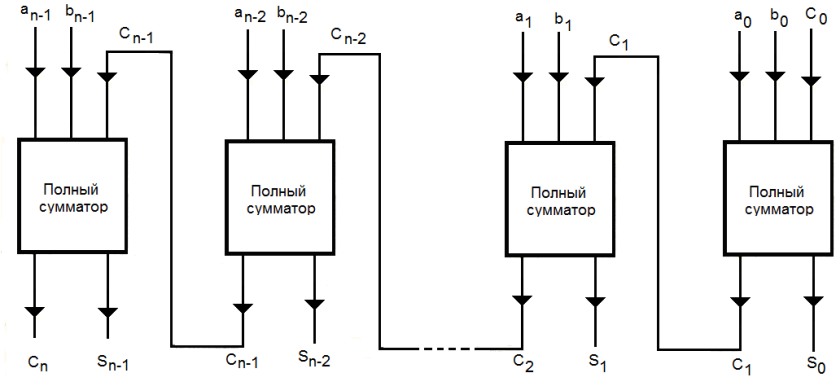


Рис. 1.3. Многоразрядный двоичный сумматор с последовательным переносом

Вычитатель с последовательным переносом можно построить по тому же принципу, что и сумматор. Обозначим через $A = a_{n-1}a_{n-2}\ldots a_1a_0$ уменьшаемое, а через $B = b_{n-1}b_{n-2}\ldots b_1b_0$ вычитаемое; т.е. вычитатель должен выполнять функцию A-B. Заем в соседнем разряде можно считать отрицательным переносом. Таким образом, разряд переноса c_i при вычитании может обозначать наличие заема

из предыдущего разряда. Операции, которые нужно выполнять в каждом разряде при вычитании, приведены в табл. 1.3. Бит разности обозначен через s_i.

Сравнивая табл. 1.3 для вычитания и табл. 1.2 для сложения, видим, что колонки для s_i идентичны. Следовательно, выражения для s_i в вычитателе и сумматоре совпадают. Каноническая сумма минтермов для вычитателя имеет вид:

$$c_{i+1} = \bar{a}_i b_i + \bar{a}_i c_i + b_i c_i .$$

Полученное выражение совпадает с выражением для сумматора, если $\bar{a}_i$ заменить на a_i. На рис. 1.4 показана схема полного вычитателя, построенного на основе выражений для разности s_i и переноса c_{i+1}.

Таблица 1.3

Определение битов разности s_i и переноса c_{i+1} по значениям a_i, b_i и c_i при вычитании

a_i	b_i	c_i	c_{i+1}	s_i
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Пользуясь сходством булевых выражений для сумматоров и вычитателей, можно построить комбинированную схему, которая сможет вычитать и складывать. Для этого необходимо предусмотреть управляющий вход, с помощью которого избирательно

инвертировать биты a_i в зависимости от требуемой операции. Для избирательной инверсии a_i можно применить вентиль Исключающее ИЛИ:

$$K \oplus a_i = K\bar{a}_i + \overline{K}a_i,$$

где K - управляющая линия. Если $K = 0$, то первый член в правой части равен нулю, а второй a_i. Если же $K = 1$, то второй член равен 0, а первый $\bar{a}_i$. Поэтому выражение $K \oplus a_i$ соответствует требуемой избирательной инверсии a_i. Схема на рис. 1.5 является n-разрядным сумматором/вычитателем с избирательной инверсией a_i в схеме вычисления переноса c_{i+1}.

Рассмотрим схему сумматора/вычитателя с использованием дополнительного кода. Дополнение можно получить, если прибавить 1 к результату обращения.

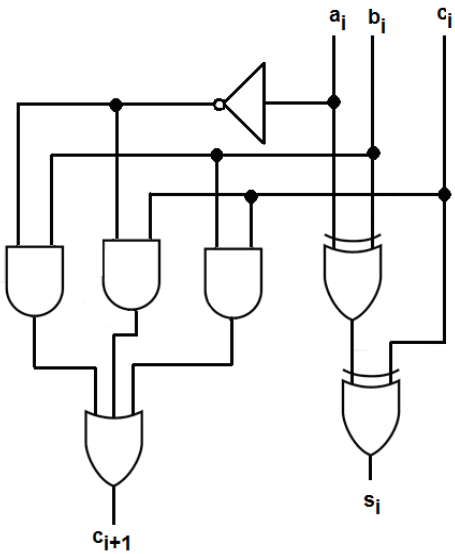


Рис. 1.4. Полный одноразрядный вычитатель

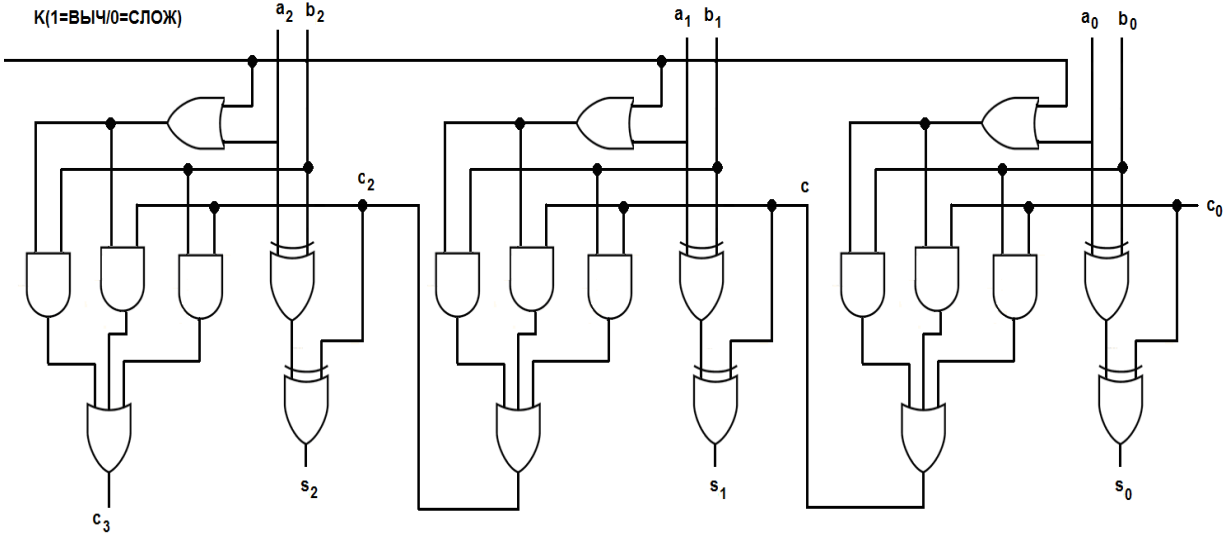


Рис. 1.5. Трехразрядный сумматор/вычитатель, составленный из полных сумматоров/вычитателей

Обращение логически эквивалентно инверсии каждого бита в числе (рис. 1.6). 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.

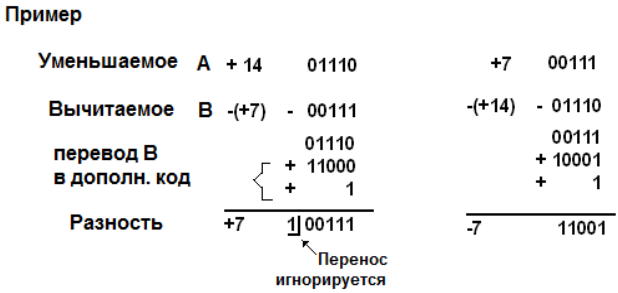


Рис. 1.6. Пример вычитания с использованием дополнительного кода (дополнение до двух). Осуществляется инвертирование вычитаемого и суммирование, и переноса 1 в младший значаший разряд с последующим сложением

Схема сумматора/вычитателя показана на рис. 1.7. Управляющий сигнал K подается на вентили Исключающее ИЛИ для всех разрядов b_i, а также на вход переноса c_0. При $K = 1$ формируется дополнение B.

Универсальная ИС полного сумматора 9304 представляет собой два полностью независимых полных одноразрядных сумматоров (рис. 1.8). Один из этих сумматоров имеет дополнительный набор входов противоположной полярности. ИС типа 9304 может использоваться для последовательного сложения и для сложения более чем двух переменных. На рис. 1.9 показаны таблицы истинности первого и второго сумматора с условными обозначениями для активно низких и высоких операндов.

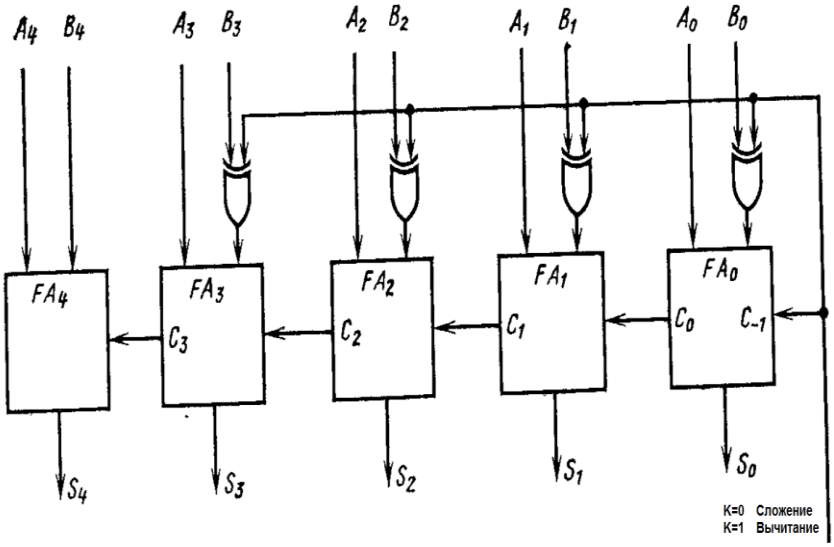


Рис. 1.7. Сумматор/вычитатель, в котором при вычитании второй операнд представляется в дополнительном коде

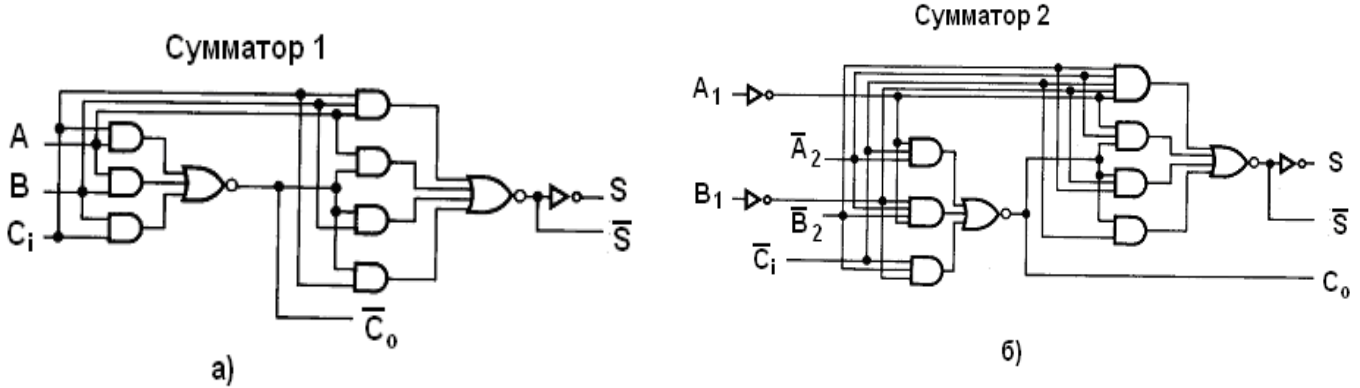
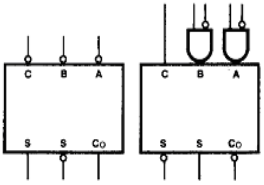


Рис. 1.8. Первая (а) и вторая (б) половина сумматора ИС типа 9304

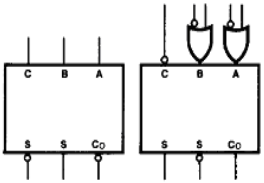
Сумматор 1

Входы			Выходы		
$\overline{C_I}$	B	A	$\overline{C_O}$	$\overline{S}$	S
L	L	L	H	H	L
L	L	H	H	L	H
L	H	L	H	L	H
L	H	H	L	H	L
H	L	L	H	L	H
H	L	H	L	H	L
H	H	L	L	H	L
H	H	H	L	L	H

Активные низкие операнды



Активные высокие операнды



Сумматор 2

Входы					Выходы		
$\overline{C_I}$	B_1	A_1	$\overline{B_2}$	$\overline{A_2}$	C_O	S	$\overline{S}$
L	L	L	L	L	H	H	L
L	L	L	L	H	H	L	H
L	L	L	H	L	H	L	H
L	L	L	H	H	L	H	L
L	L	H	L	L	H	H	L
L	L	H	L	H	H	H	L
L	L	H	H	L	H	L	H
L	L	H	H	H	H	L	H
L	H	L	L	L	H	H	L
L	H	L	L	H	H	L	H
L	H	L	H	L	H	H	L
L	H	L	H	H	H	L	H
L	H	H	L	L	H	H	L
L	H	H	L	H	H	H	L
L	H	H	H	L	H	H	L
L	H	H	H	H	H	H	L
H	L	L	L	L	H	L	H
H	L	L	L	H	L	H	L
H	L	L	H	L	L	H	L
H	L	L	H	H	L	L	H
H	L	H	L	L	H	L	H
H	L	H	L	H	H	L	H
H	L	H	H	L	L	H	L
H	L	H	H	H	L	L	H
H	H	L	L	L	H	L	H
H	H	L	L	H	L	H	L
H	H	L	H	L	L	H	L
H	H	L	H	H	L	L	H
H	H	H	L	L	H	L	H
H	H	H	L	H	H	L	H
H	H	H	H	L	H	L	H
H	H	H	H	H	H	L	H

Рис. 1.9. Таблицы истинности первого и второго сумматора с условными обозначениями для активно низких и высоких операндов ИС типа 9304

Пример:

+ 14	01110	+ 7	00111
-(+7)	- 00111	-(+14)	- 01110
	01110		00111
	+ 11000		+ 10001
	00110	-7	11000
	+ 1		
+7	00111		

- 6	11001
-(+8)	01000
	11001
	+ 10111
	10000
	+ 1
-14	10001

Рис. 1.10. Вычитание с использованием обратного кода, при котором вычитание производится инвертированием вычитаемого (обратное кодирование) и суммирование с циклическим переносом в младший значащий разряд

На рис. 1.10 показано сложение с использованием обратного кода, при котором вычитание производится инвертированием вычитаемого (обратное кодирование) и суммирование с циклическим переносом в младший значащий разряд. При сложении +7 и -14 в обратном коде циклический перенос не вырабатывается, а при переводе в дополнительный код потребовалось принудительное прибавление 1 к МЗР.

На рис. 1.11 показана схема последовательного сумматора/вычитателя с использованием обратного кода (как дополнение до единицы). Вычитание производится путем инвертирования, т.е. обратное кодирование вычитаемого и суммирование, используя циклический перенос. Для этого инвертируют вход В с использованием второй половины ИС типа 9304 (второй сумматор). Данная схема требует вторичного прохождения для циклического переноса. При этом триггер переноса при сложении взводится при активных высоких уровнях операндов, а при вычитании сбрасывается при активных низких операндах.

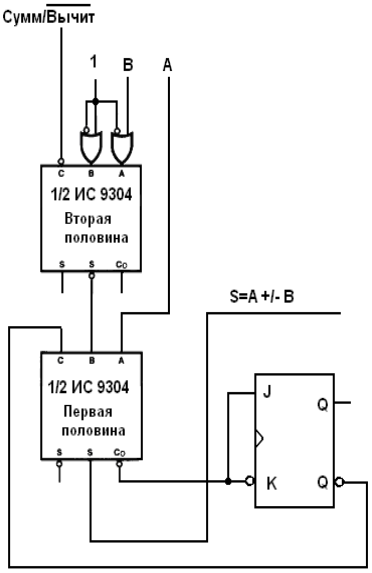


Рис. 1.11. Последовательное двоичное сложение/вычитание на ИС типа 9304 при использовании обратного кода (циклический перенос из знакового разряда к младшему значащему разряду)

1.4. Матричные умножители

Существует огромное число разновидностей матричных умножителей, превосходящих по скорости последовательностные умножители, основанные на методе сдвига и сложения. Известны и более сложные процедуры, например, с представлением суммирования в древовидном формате.

Рассмотрим один из хорошо известных умножителей чисел в дополнительном коде, так называемый умножитель Бо-Вули (Baugh-Wooley). Если A и B целые десятичные числа со знаком, то в дополнительном двоичном коде представляются в виде:

$$A = -a_{m-1}2^{m-1} + \sum_{i=0}^{m-2} a_i 2^i \quad \text{и} \quad X = -x_{n-1}2^{n-1} + \sum_{i=0}^{n-2} x_i 2^i ,$$

то

$$A * X = a_{m-1}x_{n-1}2^{m+n-2} + \\ + \sum_{i=0}^{m-2}\sum_{j=0}^{n-2} a_i x_j 2^{i+j} - \sum_{i=0}^{m-2} x_{n-1}a_i 2^{i+n-1} - \sum_{j=0}^{n-2} a_{m-1}x_j 2^{j+m-1} .$$

На рис. 1.12, а показан пример умножения чисел 5x5, представленных дополнительным кодом по формуле, а на рис. 1.12, б - пример умножения чисел 5x5, представленных дополнительным кодом по схеме Бо-Вули. На рис. 1.13 показана схема преобразований, позволяющая перевести частичные произведения со знаком в беззнаковые величины. На рис. 1.14 показан матричный умножитель Бо-Вули размерностью 5x5 чисел, представленных в дополнительном коде. Наличие полных сумматоров (FA) в матричной структуре такого умножителя является главным достоинством для реализации в базисе заказных БИС, а недостатком - пониженное быстродействие за счет увеличения высоты столбца с 5 до 7. Ниже показы примеры умножения для различных случаев по схеме Бо-Вули (рис. 1.15).

а

				×	a_4	a_3	a_2	a_1	a_0
					x_4	x_3	x_2	x_1	x_0
					$-a_4x_0$	a_3x_0	a_2x_0	a_1x_0	a_0x_0
		+		$-a_4x_1$	a_3x_1	a_2x_1	a_1x_1	a_0x_1	
			$-a_4x_2$	a_3x_2	a_2x_2	a_1x_2	a_0x_2		
		$-a_4x_3$	a_3x_3	a_2x_3	a_1x_3	a_0x_3			
	a_4x_4	$-a_3x_4$	$-a_2x_4$	$-a_1x_4$	$-a_0x_4$				
p_9	p_8	p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

б

				×	a_4	a_3	a_2	a_1	a_0
					x_4	x_3	x_2	x_1	x_0
					$a_4\overline{x_0}$	a_3x_0	a_2x_0	a_1x_0	a_0x_0
		+		$a_4\overline{x_1}$	a_3x_1	a_2x_1	a_1x_1	a_0x_1	
			$a_4\overline{x_2}$	a_3x_2	a_2x_2	a_1x_2	a_0x_2		
		$a_4\overline{x_3}$	a_3x_3	a_1x_4	a_1x_3	a_0x_3			
	a_4x_4	$\overline{a_3}x_4$	$\overline{a_2}x_4$	$\overline{a_2}x_3$	$\overline{a_0}x_4$				
	$\overline{a_4}$				a_4				
1	$\overline{x_4}$				x_4				
p_9	p_8	p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0
-2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Рис. 1.12. Пример умножения чисел 5x5, представленных дополнительным кодом (а) и по схеме Бо-Вули (б)

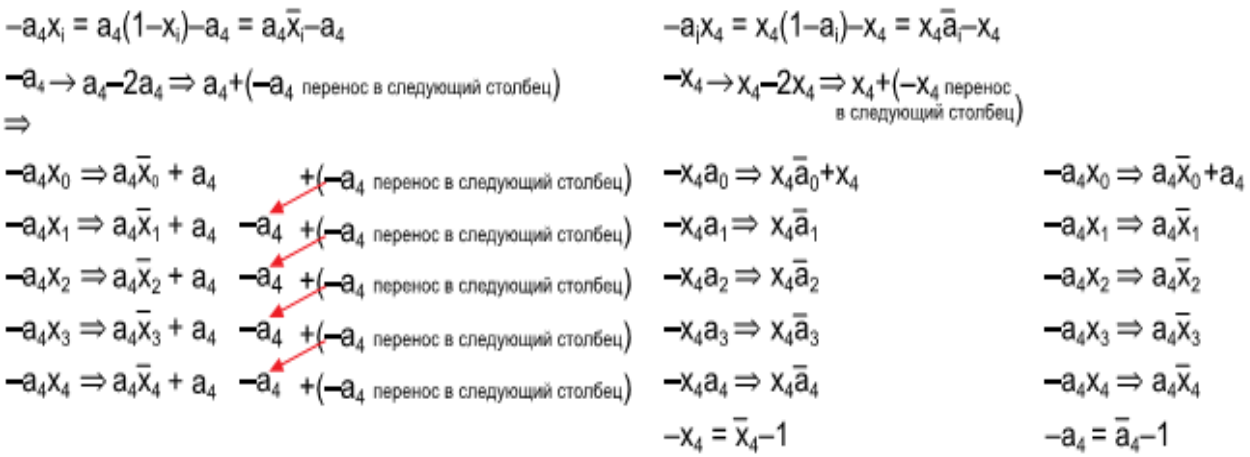


Рис. 1.13. Схема преобразований Бо-Вули

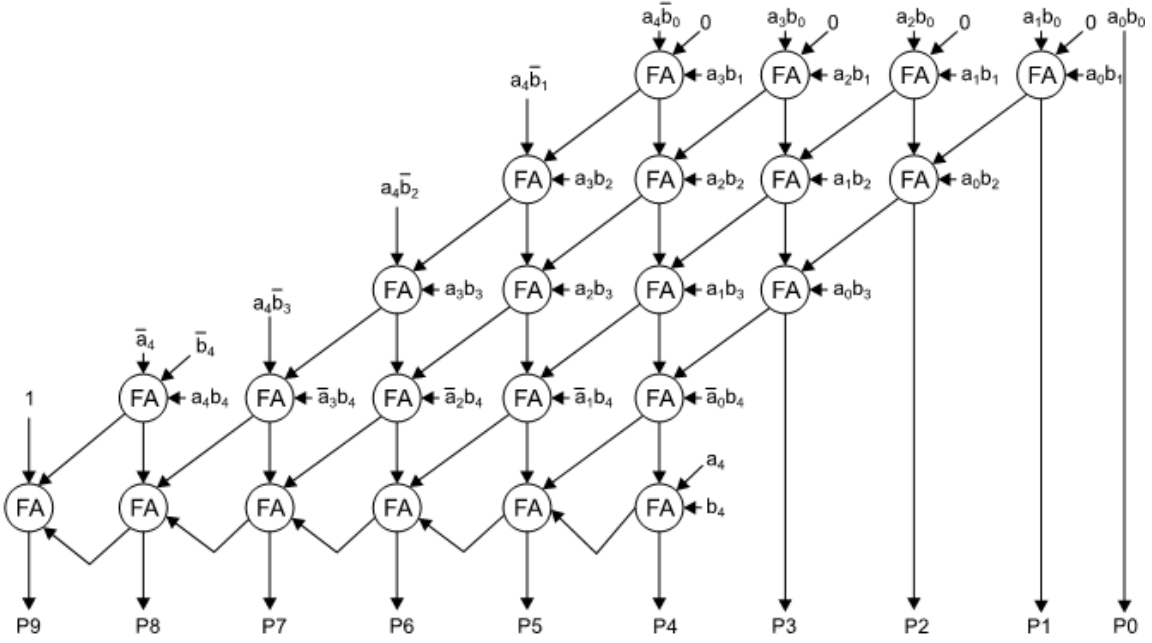
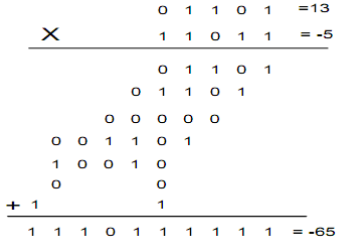
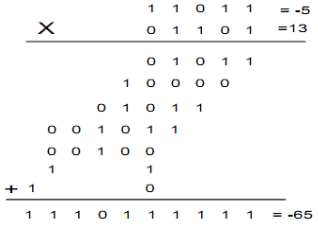


Рис. 1.14. Матричный умножитель Бо-Вули размерностью 5x5 чисел, представленных в дополнительном коде

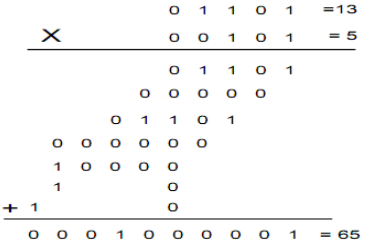
Случай 1.
Множимое - положительное число
Множитель - отрицательное число



Случай 2.
Множимое - отрицательное число
Множитель - положительное число



Случай 3.
Множимое и множитель - положительные числа



Случай 4.
Множимое и множитель - отрицательные числа

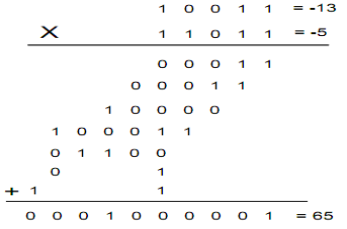


Рис. 1.15. Примеры умножения для различных случаев по схеме Бо-Вули

1.4.1. Умножение методом правого сдвига и сложения

Для проектирования КИХ-фильтров в базисе процессоров цифровой обработки сигналов (ЦОС-процессор) используется общепринятая методика умножения с накоплением с применением так называемых MAC-блоков из-за отсутствия встроенных комбинационных умножителей.

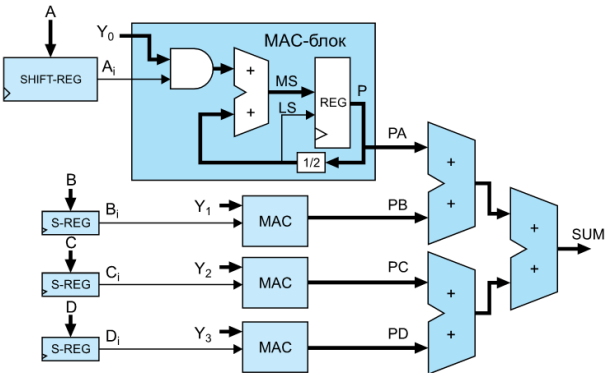
Использование данного метода для умножения чисел в базисе сигнальных процессоров является чрезвычайно популярным у разработчиков РЭА. На базе данного метода реализуются схемы быстрого умножения (например, кодирование по Буту, которое позволяет уменьшать число частичных произведений вдвое, умножение по основанию 4, модифицированное кодирование по Буту).

Например, при реализации КИХ-фильтра на четыре отвода в базисе ЦОС-процессоров требуется четыре блока умножения с накоплением (рис. 1.16, а) и многоразрядное дерево сумматоров. Применение последовательной распределенной арифметики позволяет сократить число используемых ресурсов за счет использования составных частей MAC-блока. Это четыре блока логики генерации частичных произведений, получаемых применением булевой операции логическое И к множимому (многоразрядные константы, являющиеся коэффициентами фильтра) и битовому значению множителя с выходов линии задержки и единственный масштабирующий аккумулятор. При этом дерево многоразрядных сумматоров не сокращается (рис. 1.16, б). При проектировании фильтра в базисе ПЛИС (рис. 1.16, в) на распределенной последовательной арифметике логика генерации частичных произведений и их последующее суммирование с помощью дерева многоразрядных сумматоров реализуются

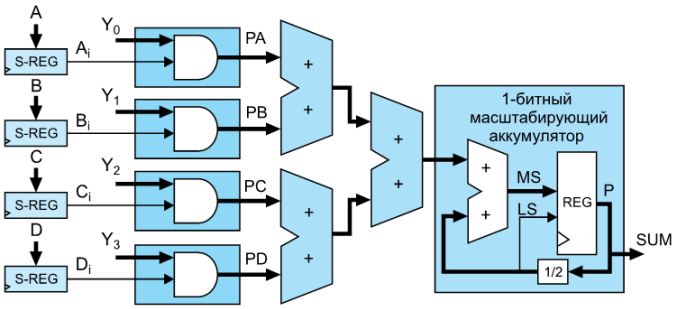
единственной таблицей перекодировки (LUT). Суммирование значений с выходов таблицы перекодировки осуществляется с использованием масштабирующего аккумулятора. Реализация КИХ-фильтров в базисе ПЛИС с использованием распределенной арифметики обеспечивает наивысшее быстродействие системы и наименьшее число используемых ресурсов проекта.

Принцип умножения методом правого сдвига и сложения показан на рис. 1.17. Идея схемы метода умножения методом правого сдвига и сложения показана на рис. 1.18. На рис. 1.19 предлагается структурная схема метода умножения с использованием управляющего автомата.

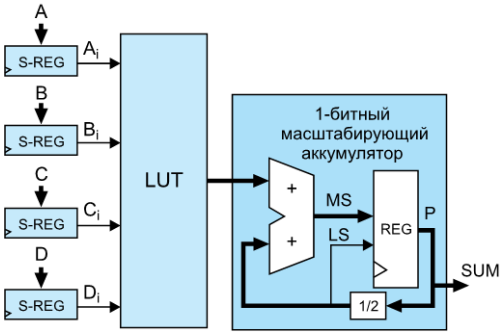
В качестве задания предлагается самостоятельно разработать функциональную схему умножителя двух 4-разрядных чисел без знака в САПР Altera Quartus II. Рассмотрим особенности MAC-блока с использованием управляющего автомата. Встраивание автомата в схему позволяет получить готовую функцию без использования дополнительных управляющих сигналов для умножения двух четырехразрядных чисел без знака. По входам MAC-блока потребуются всего лишь три сигнала: сигнал асинхронного сброса res, сигнал тактирования clk и сигнал разрешения загрузки числа X (множителя) в сдвиговый регистр load_PSC. Для корректной работы схемы необходимо обнулить все регистры умножителя (активный – высокий уровень сигнала res). Поскольку все регистры, в том числе и регистр для хранения состояний в управляющем автомате обнуляются лишь перед началом работы единожды, то для упрощения процесса разработки схемы можно воспользоваться асинхронным сбросом.



а)



б)



в)

Рис. 1.16. Миграция проекта КИХ-фильтра на четыре отвода: а) и б) реализация в базисе сигнальных процессоров; в) в базисе ПЛИС

	Cout	2 тетрада	1 тетрада	2 тетрада	2 тетрада	1 и 2 тетрады
a 10 x 11			1 0 1 0 1 0 1 1			
$p^{(0)}$ $+x_0a$		0 0 0 0 1 0 1 0	0 0 0 0	0 a	0 10	
$2p^{(1)}$ $p^{(1)}$ $+x_1a$	0	1 0 1 0 0 1 0 1 1 0 1 0	0 0 0 0	a a/2 a	10 5 10	80
$2p^{(2)}$ $p^{(2)}$ $+x_2a$	0	1 1 1 1 0 1 1 1 0 0 0 0	0 1 0 0 0	a/2+a (a/2+a)/2 0	15 7 0	120
$2p^{(3)}$ $p^{(3)}$ $+x_3a$	0	0 1 1 1 0 0 1 1 1 0 1 0	1 0 1 1 0 0	(a/2+a)/2 ((a/2+a)/2)/2 a	7 3 10	60
$2p^{(4)}$ $p^{(4)}$	0	1 1 0 1 0 1 1 0	1 1 0 1 1 1 0	((a/2+a)/2)/2+a (((a/2+a)/2)/2+a)/2	13 6	110

Рис. 1.17. Принцип умножения методом правого сдвига и сложения. Умножение десятичного числа 10 на десятичное число 11

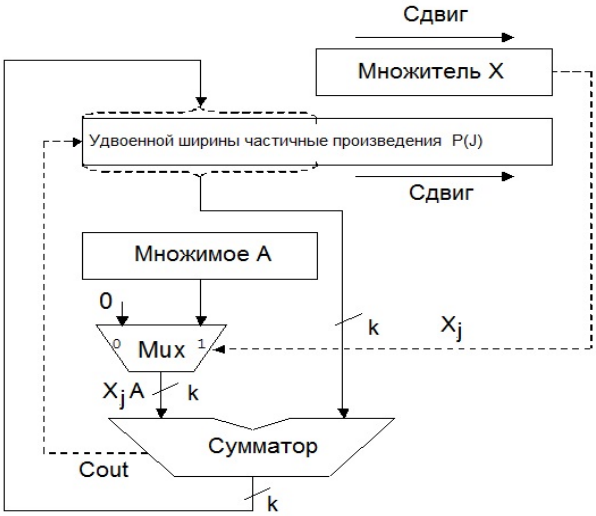


Рис. 1.18. Структурная схема умножителя методом правого сдвига и сложения

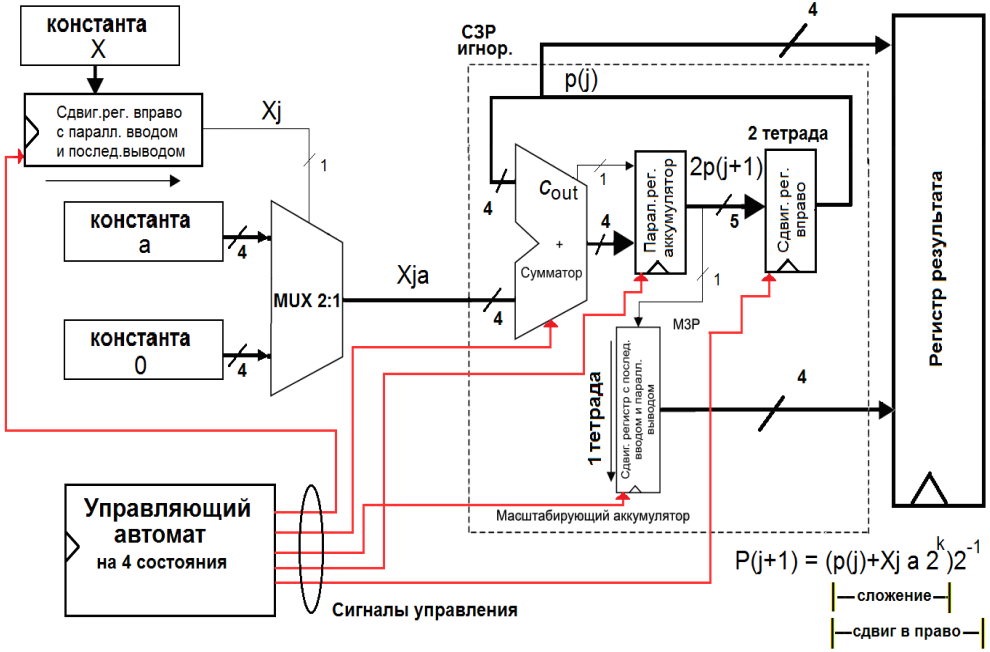


Рис. 1.19. Структурная схема метода умножения с использованием управляющего автомата

1.4.2. Умножение целых чисел со знаком методом правого сдвига и сложения

Рассмотрим пример последовательностного универсального умножителя целых чисел, представленных в дополнительном коде, методом правого сдвига и сложения (MAC-блок) в базисе ПЛИС.

Рассмотрим умножение чисел со знаком в “столбик” (рис. 1.20). 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 сумматора.

На рис. 1.20 показан принцип умножения чисел представленных дополнительным кодом, на примере умножения -5x-3. Представление процесса умножения в точечной нотации, в которой под каждой точкой подразумевается логическая 1 или логический 0, позволяет получить рекуррентную формулу (рис. 1.21).

Ниже показаны примеры умножения, рассматриваемые при разработке схемы универсального умножителя.

Случай 1. Множимое - отрицательное число Множитель - положительное число -10x11	**Случай 2.** Множимое и множитель - отрицательные числа -10x-11
Случай 3. Множимое и множитель - четные отрицательные числа -4x-4	**Случай 4.** Множимое - положительное число Множитель - отрицательное число 5x-3

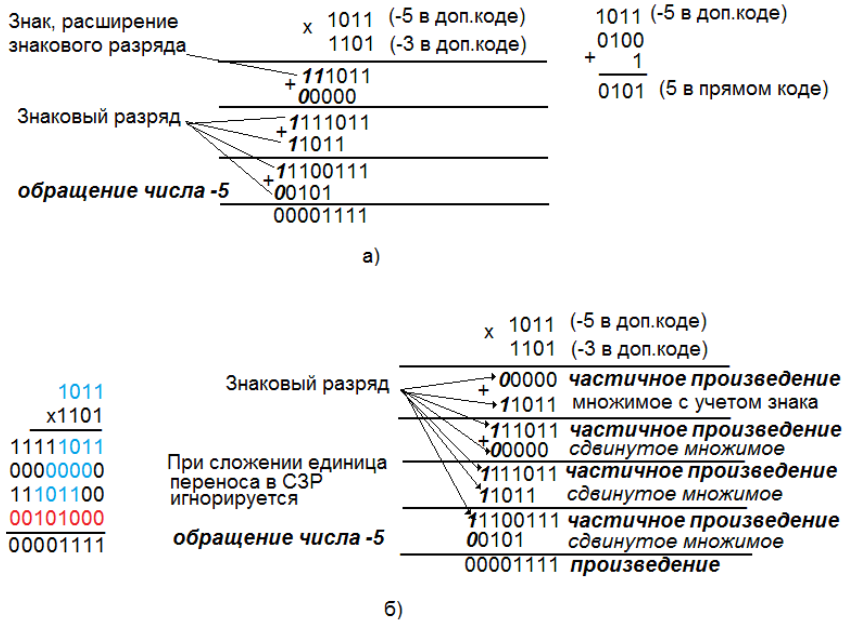


Рис. 1.20. Умножение в столбик (а); умножение методом сдвига множимого и последующего сложения с частичным произведением (умножение чисел -5x-3, представленных в дополнительном коде) (б)

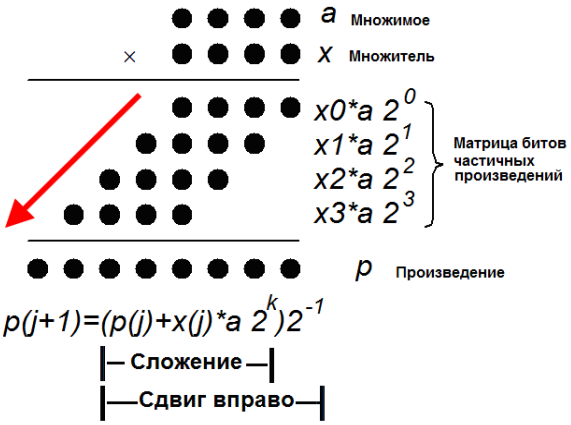


Рис. 1.21. Представление процесса умножения методом правого сдвига и сложения в точечной нотации

На рис. 1.22 показан принцип умножения методом правого сдвига и сложения для двух случаев. В первом случае осуществляется умножение числа со знаком -10 на число 11, а во втором - умножение -10 на -11. В первом случае множимое (-10) переводится в дополнительный код (дополнение до двух).

Множитель (11) – целое положительное число, расширенное знаковым разрядом 0, представлено в прямом коде. Для удвоенных частичных произведений 2p(1), 2p(2), 2p(3), 2p(4) и 2p(5) в поле MSB (название полей произвольное) необходимо добавить логическую 1. При формировании частичных произведений методом правого сдвига p(1), p(2), p(3), p(4) и p(5) логическая 1 из поля MSB попадает в старший разряд второй тетрады (название “тетрады” в данном случае не корректно, т.к. это поле уже 5-разрядное, но сохранено для приемлемости с принципом умножения без знаковых чисел).

При сложении p(3) с x3*a единица переноса в старший разряд, т.е. в поле MSB, игнорируется. Данная схема

вычислений справедлива только для случая, когда в младшем разряде множителя находится 1.

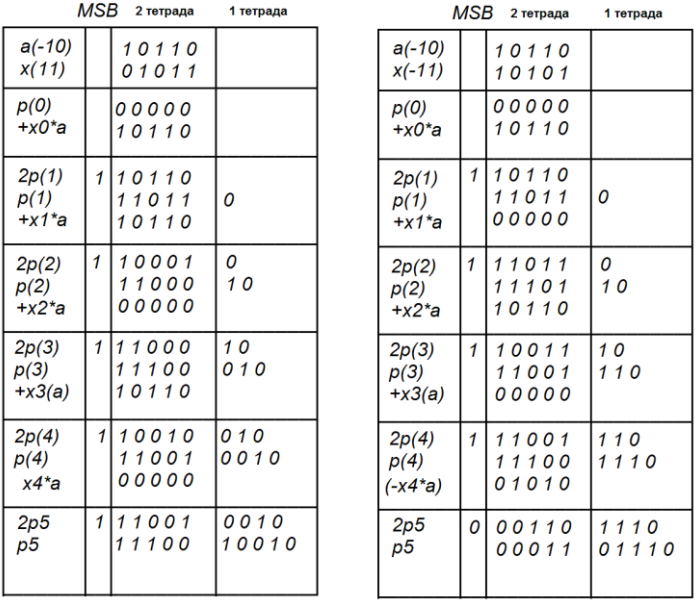


Рис. 1.22. Принцип умножения методом правого сдвига и сложения:
а) умножение -10х11; б) умножение -10х-11

При умножении двух отрицательных чисел, представленных дополнительным кодом (например, -10х-11), необходимо произвести два действия. Первое, необходимо учесть знак при представлении числа в дополнительном коде, что достигается обращением произведения старшего разряда множителя на множимое с последующим прибавлением 1 к младшему разряду. Дополнительный код произведения (–х4*а) при х4=1 есть число 10. Перевод в дополнительный код произведения (–х4*а) должен быть осуществлен до операции

сложения, т.е. до получения удвоенного частичного произведения 2p(5).

Второе, при формировании удвоенного частичного произведения 2p(5) необходимо произвести коррекцию, т.е. в поле MSB поставить логический ноль.

Рассмотрим умножение четных чисел со знаком (рис. 1.23). При умножении -4x-4 в поле MSB для удвоенных значений частичных произведений 2p(1) и 2p(2) должны стоять нули. А при умножении -8x-8 нуль в поле MSB должен быть еще 0 и для частичного произведения 2p(3). Далее, принцип умножения не отличается от умножения чисел, представленных дополнительным кодом (например, -10x-11).

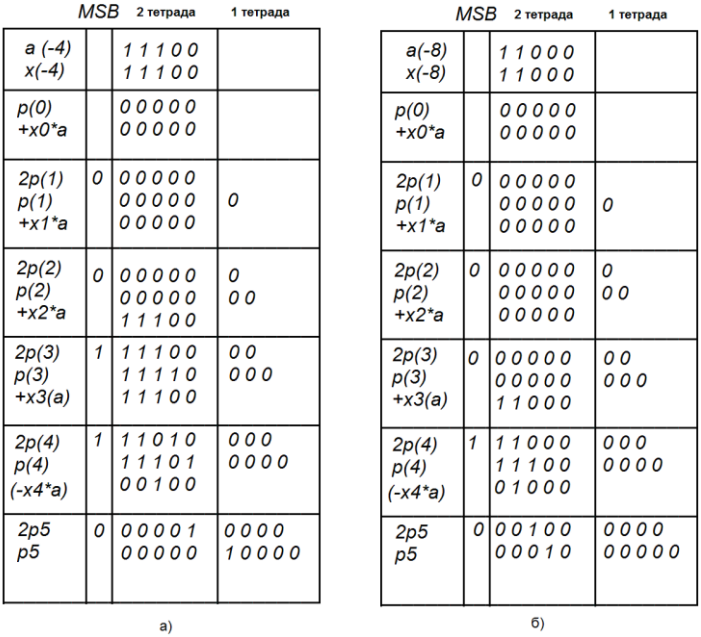


Рис. 1.23. Принцип умножения методом правого сдвига и сложения: а) умножение -4x-4; б) умножение -8x-8

На рис. 1.24 показан принцип умножения методом правого сдвига и сложения в случае, когда множимое – положительное, а множитель - отрицательное числа.

	MSB	2 тетрада	1 тетрада
a (5) x (-3)		0 0 1 0 1 1 1 1 0 1	
p(0) +x0*a		0 0 0 0 0 0 0 1 0 1	
2p(1) p(1) +x1*a	0	0 0 1 0 1 0 0 0 1 0 0 0 0 0 0	1
2p(2) p(2) +x2*a	0	0 0 0 1 0 0 0 0 0 1 0 0 1 0 1	1 0 1
2p(3) p(3) +x3(a)	0	0 0 1 1 0 0 0 0 1 1 0 0 1 0 1	0 1 0 0 1
2p(4) p(4) (-x4*a)	0	0 1 0 0 0 0 0 1 0 0 1 1 0 1 1	0 0 1 0 0 0 1
2p5 p5	1	1 1 1 1 1 1 1 1 1 1	0 0 0 1 1 0 0 0 1

	MSB	2 тетрада	1 тетрада
a (2) x (-2)		0 0 0 1 0 1 1 1 1 0	
p(0) +x0*a		0 0 0 0 0 0 0 0 0 0	
2p(1) p(1) +x1*a	0	0 0 0 0 0 0 0 0 0 0 0 0 0 1 0	0
2p(2) p(2) +x2*a	0	0 0 0 1 0 0 0 0 0 1 0 0 0 1 0	0 0 0
2p(3) p(3) +x3(a)	0	0 0 0 1 1 0 0 0 0 1 0 0 0 1 0	0 0 1 0 0
2p(4) p(4) (-x4*a)	0	0 0 0 1 1 0 0 0 0 1 1 1 1 1 0	1 0 0 1 1 0 0
2p5 p5	1	1 1 1 1 1 1 1 1 1 1	1 1 0 0 1 1 1 0 0

Рис. 1.24. Принцип умножения методом правого сдвига и сложения: а) умножение 5х-3; б) умножение 2х-2

На рис. 1.25 показан принцип умножения чисел без знака. Единица переноса при сложении в поле “2 тетрада” уже не игнорируется, а переносится в поле Cout (сигнал Cout является выходом переноса многоразрядного сумматора масштабирующего аккумулятора).

	Cout	2 тетрада	1 тетрада	1 и 2 тетрады
a x		1 1 1 1 1 1 1 1		
$p^{(0)}$ $+x_0a$		0 0 0 0 1 1 1 1		
$2p^{(1)}$ $p^{(1)}$ $+x_1a$	0	1 1 1 1 0 1 1 1 1 1 1 1	1 0 0 0	120
$2p^{(2)}$ $p^{(2)}$ $+x_2a$	1	0 1 1 0 1 0 1 1 1 1 1 1	0 1 0 0	180
$2p^{(3)}$ $p^{(3)}$ $+x_3a$	1	1 0 1 0 1 1 0 1 1 1 1 1	0 0 1 0	210
$2p^{(4)}$ $p^{(4)}$	1	1 1 0 0 1 1 1 0	0 0 0 1	225

Рис. 1.25. Принцип умножения методом правого сдвига и сложения. Умножение 15x15

1.5. Программные умножители в базисе ПЛИС

В ПЛИС для повышения их функциональных возможностей встраивают, например, для серии Cyclone III фирмы Altera аппаратные умножители, которые могут быть сконфигурированы в виде одного умножителя 18x18 либо в виде двух умножителей 9x9. Так, ПЛИС EP3CLS200 содержит 396 аппаратных умножителей 18x18, а на оставшихся ресурсах может быть реализован 891 программный умножитель 16x16. В итоге суммарное число умножителей составляет 1287 без какого-либо значительного использования логических ресурсов.

Для устройств цифровой обработки сигналов себя хорошо зарекомендовали софт-умножители (программные умножители), которые не требуют ресурсов аппаратных

умножителей, встроенных в базис ПЛИС. Повысить производительность устройств цифровой обработки сигналов позволяет также использование параллельного векторного умножителя и “безумножительных” схем умножения с использованием основ распределенной арифметики.

Рассмотрим параллельные программные умножители, способные вычислять произведение за один такт синхроимпульса, обеспечивая наивысшую производительность устройств цифровой обработки сигналов. Программные умножители БИС программируемой логики (БИС ПЛ) фирмы Actel серий Fusion, IGLOO и ProASIC3 реализуются на блочной памяти меньшей размерности, чем у ПЛИС фирмы Altera и их можно рассматривать как 8-входовые LUT или таблицы произведений. Таблица произведений множимого, записанная во фрагмент блочной памяти, и называется LUT. Табл. 1.4 показывает умножитель размерностью 3x3, реализованный с помощью 6-входовой LUT.

Таблица 1.4

Умножитель размерностью 3x3, реализованный с помощью 6-входовой LUT

Множитель \ Множимое	000	001	010	011	100	101	110	111
000	000000	000000	000000	000000	000000	000000	000000	000000
001	000000	000001	000010	000011	000100	000101	000110	000111
010	000000	000010	000100	000110	001000	001010	001100	001110
011	000000	000011	000110	001001	001100	001111	010010	010101
100	000000	000100	001000	001100	010000	010100	011000	011100
101	000000	000101	001010	001111	010100	011001	011110	100011
110	000000	000110	001100	010010	011000	011110	100100	101010
111	000000	000111	001110	010101	011100	100011	101010	110001

Например, у ПЛИС фирмы Actel используется ОЗУ емкостью 256 слов x 8 бит (256 8-разрядных слов), а у ПЛИС фирмы Altera может использоваться память M4K, которая может быть сконфигурирована, как 128 слов x 36 бит или 256

слов x 18 бит для серии Cyclone II. Такие умножители получили название RAM-LUT-умножители или LUT-based умножители.

На рис. 1.26 показан умножитель 4x4 на базе синхронного ОЗУ емкостью 256 8-разрядных слов. Множимое (младшие четыре разряда адресной шины) и множитель (старшие четыре разряда адресной шины), представленные 4-разрядным двоичным кодом, объединяются в 8-разрядную адресную шину, адресуя своим уникальным кодом содержимое конкретной строки ОЗУ (операнды), являющееся 8-ми разрядным произведением.

Недостатком такого умножителя является резкое возрастание требуемого объема блочной памяти в случае увеличения его разрядности. Для умножителя размерностью 8x8 требуется 65536 16-разрядных слов. Поэтому чтобы предотвратить рост требуемой памяти на практике используется умножитель на суммировании частичных произведений в соответствии со своим весом (partial product multipliers).

На рис. 1.27 показан пример умножения десятичного числа 24 на 43. Например, произведению 2 на 4 приписывается вес 100, что равносильно сдвигу на две позиции в десятичной системе. В этом случае необходим умножитель размерностью 8x8. Однако согласно принципу умножения с использованием частичных произведений требуются четыре умножителя размерностью 4x4 для формирования четырех частичных произведений и три сумматора: (4x3+((2x3)x10))+((4x4)+((2x4)x10)x10)=1032. На рис. 1.28 показана структурная схема такого умножителя. Так же для сдвига на одну и две десятичные позиции потребуются три сдвиговых регистра на четыре разряда влево и дополнительные блоки, выполняющие операции расширения знака со значением старшего разряда.

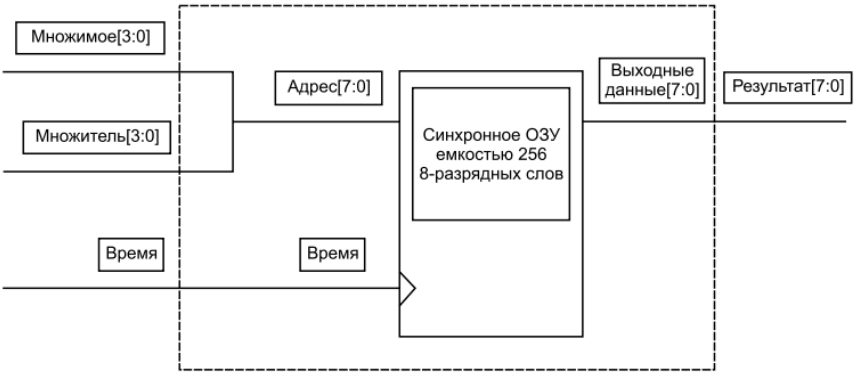


Рис. 1.26. Программный умножитель размерностью 4x4 на базе ОЗУ емкостью 256 8-разрядных слов (256x8) фирмы Actel

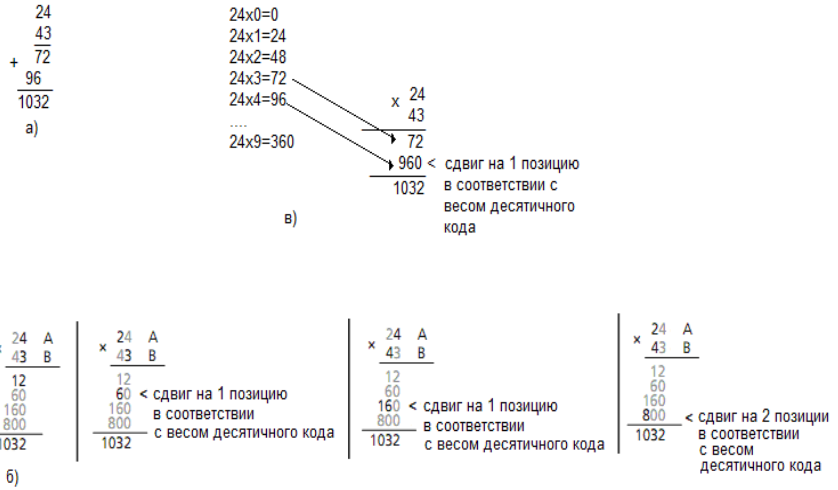


Рис. 1.27. Принцип умножения: а) “в столбик” по правилу умножения десятичных чисел; б) с использованием частичных произведений; в) умножение на константу

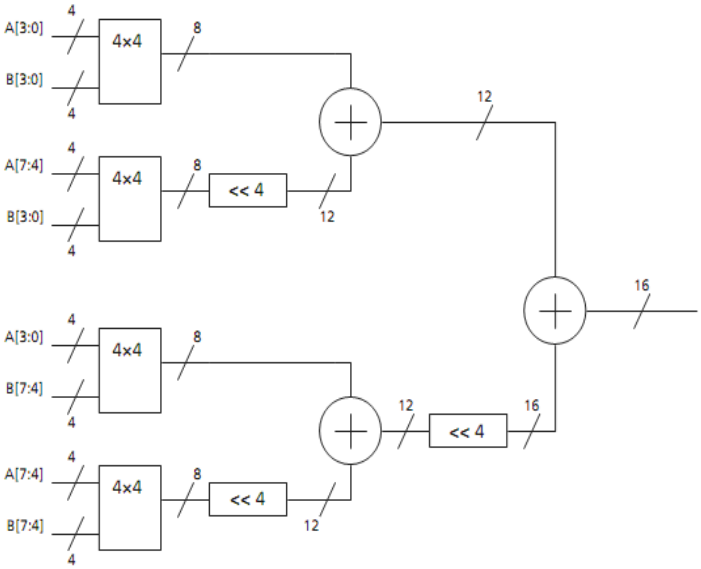


Рис. 1.28. Структурная схема умножителя размерностью 8x8 с использованием четырех умножителей размерностью 4x4

Рассмотрим программные умножители на константу. Одна из наиболее распространенных операций цифровой обработки сигналов - умножение числа на константу. Для перемножения двух чисел достаточно иметь таблицу произведений множимого (константы) на весь ранг возможных цифр множителя (табл. 1.5) и осуществить корректное суммирование полученных частичных произведений.

Программные умножители фирмы Actel реализуются на 8-входовых LUT. Множимое (константа) в этом случае предопределено. В этом случае необходимы два блока памяти емкостью 256 8-разрядных слов позволяющих организовать массив памяти 256 16-разрядных слов из двух блоков емкостью 256 8-разрядных слов, выходная шина которого и есть 16-разрядный результат умножения двух 8-разрядных чисел (рис. 1.29).

Программные умножители фирмы Altera. Наличие встроенной блочной памяти TriMatrix™ в ПЛИС фирмы Altera, например, типа M9K используемой в качестве LUT, в которых хранятся частичные произведения, позволяет реализовывать параллельные умножители, экономя при этом не только аппаратные умножители, но и ресурсы логических блоков. Наличие программных и аппаратных умножителей приводит в целом к увеличению общего числа возможных умножителей.

Таблица 1.5

Умножение 4-разрядного числа на константу 24 (рис. 1.27, в)

Входы			X[3]	
X[2]	X[1]	X[0]	0	1
000			0	192
001			24	216
010			48	240
011			72	264
100			96	288
101			120	312
110			144	336
111			168	360

Использовать программные и аппаратные умножители в проекте пользователя возможно через мегафункции. Мегафункции lpm_mult, altmult_add и altmult_accum позволяют использовать аппаратные умножители в ПЛИС. Рассмотрим мегафункцию ALTMEMMULT – программный умножитель. Мегафункция ALTMEMMULT позволяет осуществлять процесс умножения числа на константу C, при этом константа может храниться в блочной памяти ПЛИС либо загружается с внешнего порта.

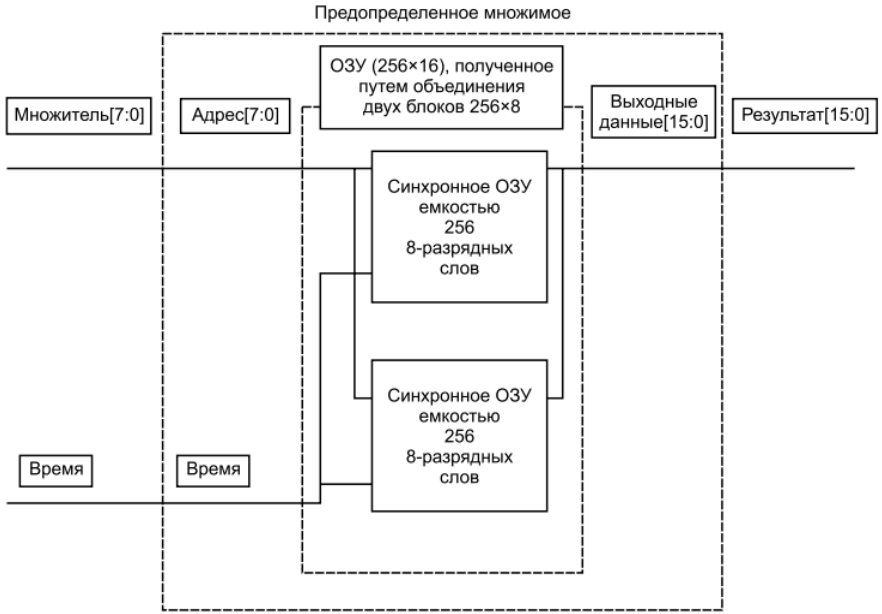


Рис. 1.29. Программный умножитель размерностью 8x8 числа на константу на базе ОЗУ емкостью 256 16-разрядных слов фирмы Actel

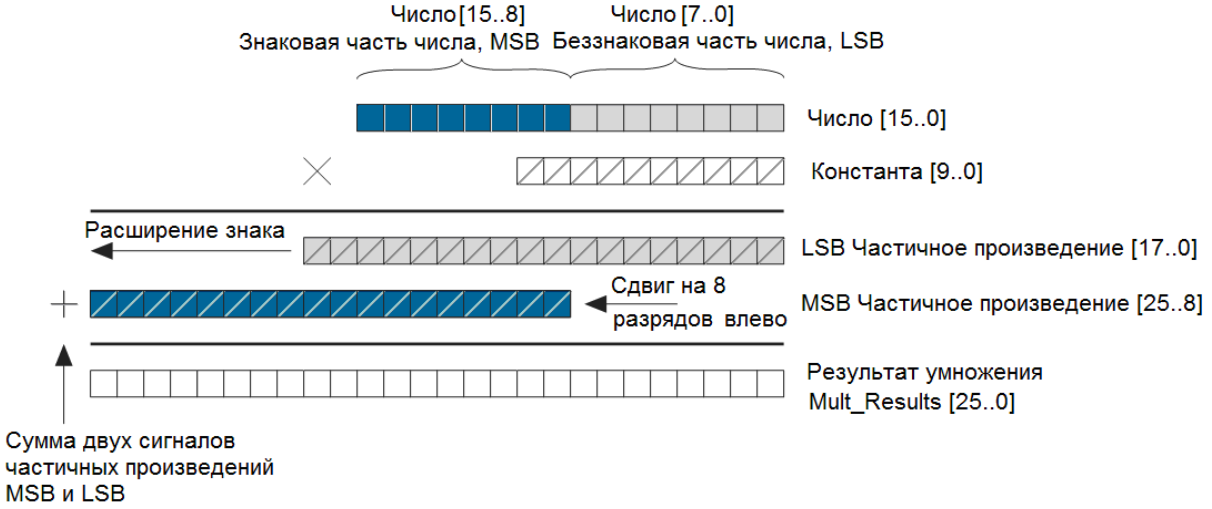


Рис. 1.30. Идея параллельного умножения 16-разрядного числа на 10-разрядную константу

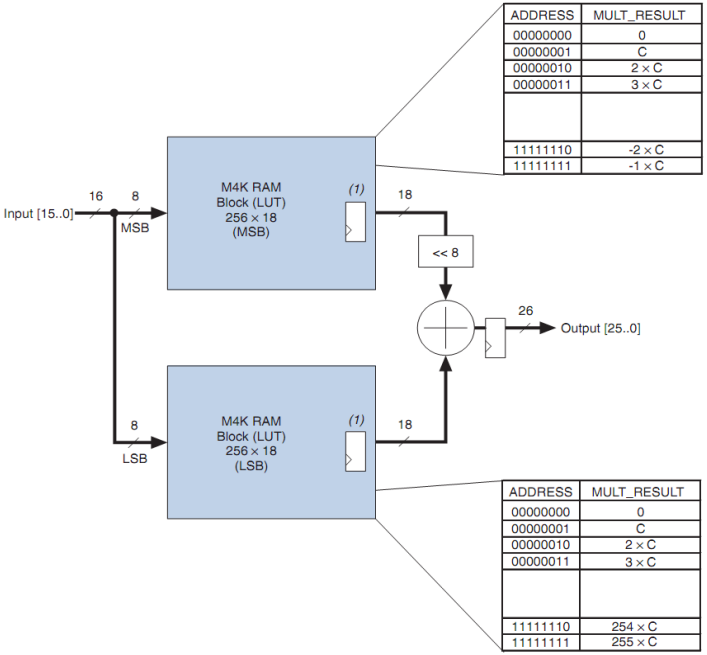


Рис. 1.31. Программный параллельный умножитель 16-разрядного числа на 10-разрядную константу размерностью 16x10 с использованием двух блоков памяти типа M4K в качестве LUT фирмы Altera

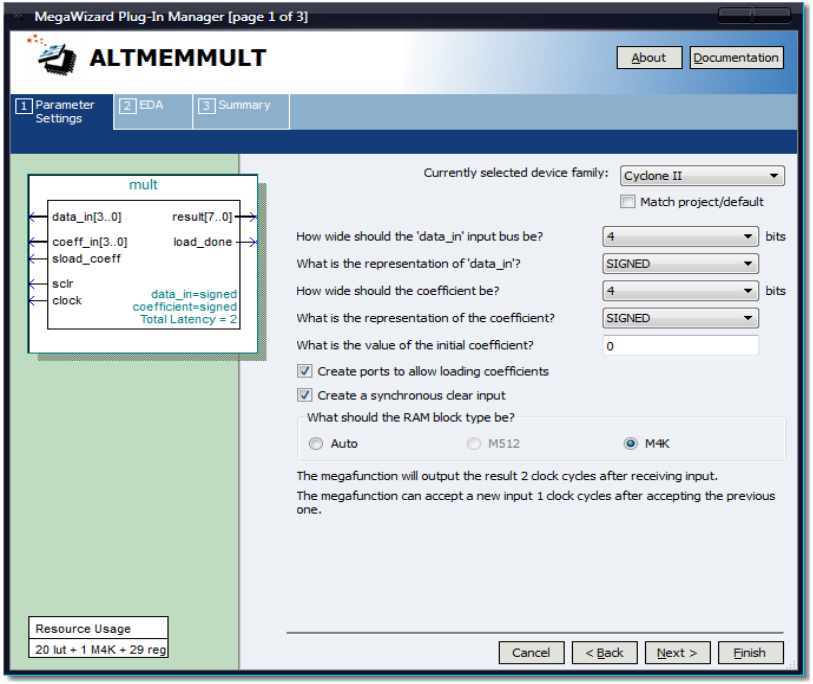
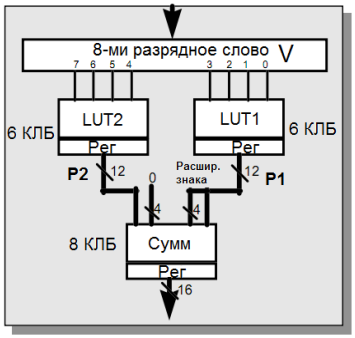


Рис. 1.32. Мегафункция ALTMEMMULT, настроенная для реализации программного умножителя 4x4



$v_7\ v_6\ v_5\ v_4$ 0 0 0 0	V1
0 0 0 0 $v_3\ v_2\ v_1\ v_0$	V0

V=V1+V0

VxC = V1xConst +V0xConst = P2+P1

	ADRS	P2 LUT2	P1 LUT1
0	0000	0	0
1	0001	1 x Const	1 x Const
2	0010	2 x Const	2 x Const
3	0011	3 x Const	3 x Const
4	0100	4 x Const	4 x Const
5	0101	5 x Const	5 x Const
6	0110	6 x Const	6 x Const
7	0111	7 x Const	7 x Const
8	1000	-8 x Const	8 x Const
9	1001	-7 x Const	9 x Const
A	1010	-6 x Const	A x Const
B	1011	-5 x Const	B x Const
C	1100	-4 x Const	C x Const
D	1101	-3 x Const	D x Const
E	1110	-2 x Const	E x Const
F	1111	-1 x Const	F x Const
	SIGN BIT	16 старших частичных произведений разрядностью 12 бит	16 младших частичных произведений разрядностью 12 бит

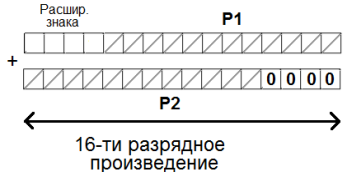


Рис. 1.33. Программный умножитель на константу размерностью 8x8 с использованием двух 4-входовых LUT ПЛИС серии XC4000

Для ПЛИС серии Cyclone II возможно использовать только память M4K (128x36 бит) или режим Avto. Например, ПЛИС EP2C70 содержит 250 блоков M4K. На рис. 1.30 показана идея умножения числа на константу. Число и константа могут быть как со знаком, так и без него. На рис. 1.31 показан принцип построения программного умножителя 16-разрядного числа на 10-разрядную константу (обозначена буквой C) с использованием блоков памяти типа M4K большего размера, чем у БИС ПЛ фирмы Actel для случая, когда константы хранятся в блочной памяти, т.е. отсутствует возможность их загрузки из вне (отсутствуют адресные порты для загрузки коэффициентов).

Входной 16-разрядный сигнал разделяется на два 8-разрядных сигнала с именами LSB (младший значащий разряд) и MSB (старший значащий разряд). Сигнал LSB адресуется к блоку памяти M4K с одноименным названием LSB, а сигнал MSB адресуется к блоку памяти M4K с одноименным названием MSB. В блочной памяти LSB хранятся 256 предварительно вычисленных частичных произведений с именем “LSB Частичное произведение [17..0]” (младшее частичное произведение) разрядностью 18 бит с диапазоном от 0 до 255xC, а в памяти MSB с диапазоном от 0 до (-1)xC хранятся 256 предварительно вычисленных частичных произведений с именем “MSB Частичное произведение [25..8]” (старшее частичное произведение). Далее старшее частичное произведение необходимо сдвинуть на восемь разрядов влево, а затем осуществить сквозное суммирование.

Умножение фактически осуществляется за один такт синхроимпульса, необходимый для загрузки входных значений сигналов LSB и MSB в адресные порты блоков памяти. Еще два такта требуются для конвейеризации задержки вычислений, т.к. выходные значения блоков памяти, представляющие собой частичные произведения, должны быть

еще просуммированы с соответствующими весами для получения 26-разрядного результата умножения.

В качестве примера на рис. 1.32 показаны настройки мегафункции ALTMEMMULT для умножения 4-разрядного числа, представленного дополнительным кодом, и 4-разрядной константы, загружаемой из внешнего порта. В этом случае требуется 20 LUT логических блоков плюс 1 блок памяти типа M4K и 20 триггеров (20 lut+1M4K+20 reg). В случае загрузки константы из блочной памяти требуется всего лишь 1M4K.

Принцип построения на рис. 1.30 не раскрывает все тонкости такого умножителя. В частности, не показана операция расширения знака числа. На рис. 1.33 показан принцип построения программного умножителя на константу размерностью 8x8 с использованием двух 4-входовых LUT ПЛИС серии XC4000.

На рис. 1.33 обозначено: V - входной 8-разрядный сигнал; P1 и P2 - младшее и старшее частичные произведения; C - константа. В частности, показано, как на практике осуществляется сдвиг на четыре разряда влево. Для умножителя требуются 25 конфигурируемых логических блоков (КЛБ). Объединяя такие умножители в секции (одна секция на отвод фильтра), можно построить высокопроизводительный параллельный КИХ-фильтр, работающий на частотах 50-70 МГц.

1.6. Разработка проекта умножителя размерностью 4x4 в базисе ПЛИС с помощью учебного лабораторного стенда LESO2.1

В разделе 1.4.1 рассмотрено умножение целых положительных чисел, представленных в прямом коде, размерностью 4x4 методом правого сдвига и сложения, а в разделе 1.4.2 - умножение целых чисел со знаком, представленных в дополнительном коде.

Рассмотрим проектирование цифрового автомата более простым способом - методом умножения в столбик. Управляющий автомат умножителя разработаем с помощью редактора состояний САПР Quartus II (State Machine Viewer). Далее реализуем умножитель размерностью 4x4 в базисе ПЛИС типа ППВМ серии Cyclone EP1C3T144C8N фирмы Altera с помощью учебного лабораторного стенда LESO2.1 (Лаборатории электронных средств обучения, ЛЭСО ГОУ ВПО «СибГУТИ») отечественной разработки. Учебный лабораторный стенд предназначен для обучения основам проектирования цифровой техники на основе ПЛИС.

Так как САПР Quartus II Web Edition version 13.0.1 сборка 232 не поддерживает ПЛИС серии Cyclone, то необходимо перейти на более раннюю версию Quartus II Web Edition version 9.1.

На рис. 1.34 и рис. 1.35 показаны верхний и нижний уровни иерархии проекта умножителя (P=B(множимое)*A(множитель)) размерностью 4x4. Сигнал A (множитель) следует рассматривать как число, а сигнал B - как константу (множимое). Умножитель настроен на умножение двух чисел 10x10. Умножитель состоит из двух однотипных регистров ShiftN, сдвигающих влево или вправо в зависимости

от сигнала DIR задающего направление сдвига (пример 1), детектора нуля AllZero, управляющего автомата avt на пять состояний (пример 2), 8-разрядного сумматора на мегафункции lpm_add_sub, шинного мультиплексора на мегафункции lpm_mux и 8-разрядного регистра на мегафункции lpm_dff, выполняющего роль аккумулятора. Один из регистров ShiftN (DIR=0), на вход которого подается число A, работает как преобразователь параллельного кода в последовательный, параллельный выход SRA[7..0] нужен лишь для детектирования нуля.

Рис. 1.36 демонстрирует принцип работы управляющего автомата. Автомат принимает пять состояний с именами Check_FS, Init_FS, Adder_FS, shift_FS, End_mult. В каждом из состояний активным является один из сигналов Init, Add, Shift и Done. Автомат разработан по “классической” схеме с использованием одного оператора Process (однопроцессный шаблон) для описания памяти состояний и логики переходов.

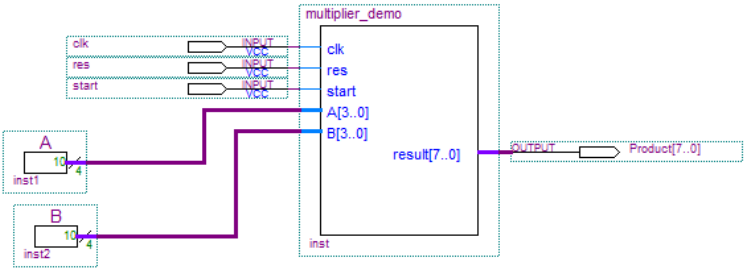


Рис. 1.34. Умножитель размерностью 4x4. Верхний уровень иерархии схемы

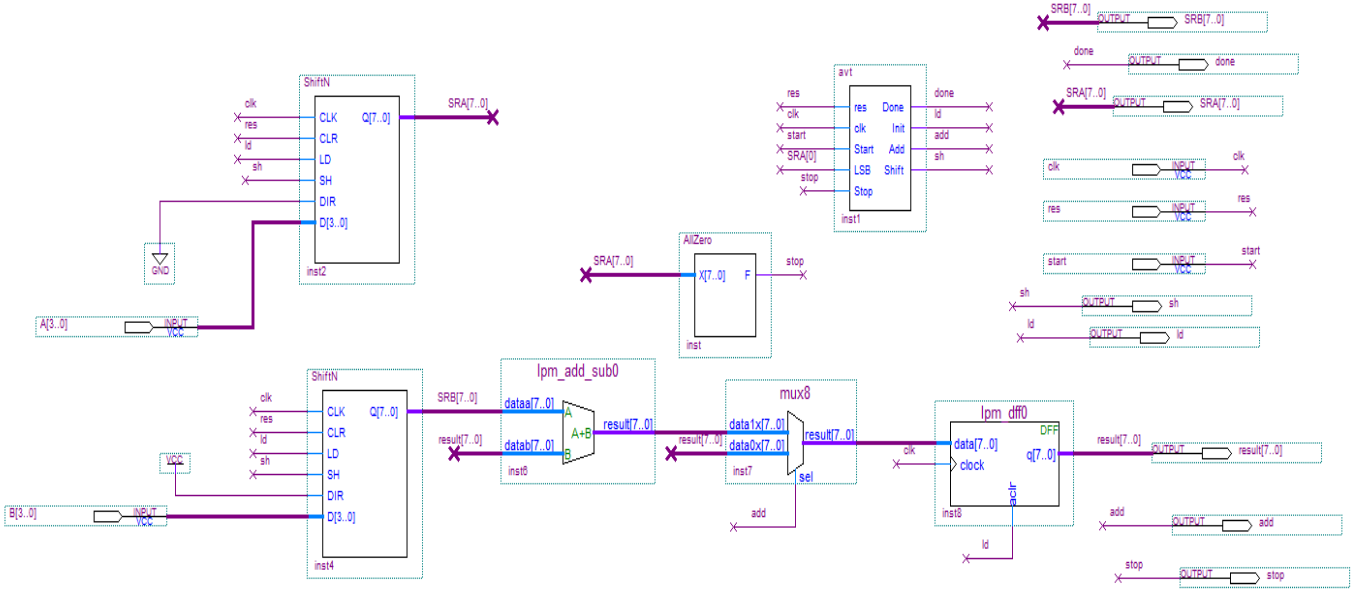


Рис. 1.35. Умножитель размерностью 4x4. Нижний уровень иерархии

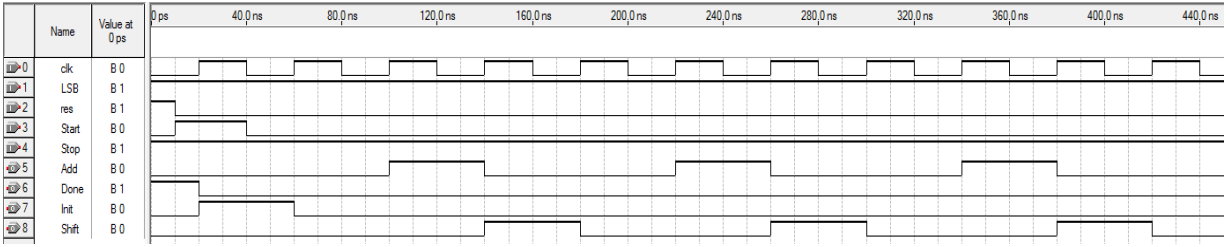


Рис. 1.36. Тест цифрового автомата

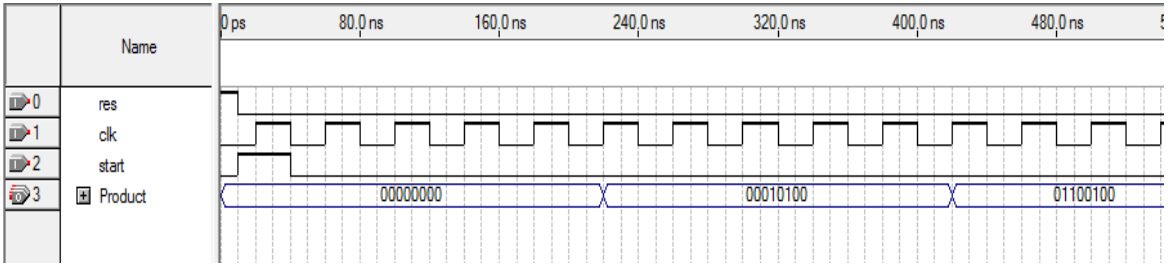


Рис. 1.37. Тестирование умножителя на примере умножения 10x10. Результат 100

Автомат инициализируется высоким уровнем сигнала Start синхронизируемого синхросигналом clk, переводящим выход Init в активное состояние. При высоком уровне сигнала Init происходит загрузка обоих сдвиговых регистров параллельным кодом. Если на вход LSB все время будет поступать логическая 1 (младший разряд SRA[0] 8-разрядного сигнала SRA[7..0]) с выхода сдвигового регистра ShiftN при DIR=0, то управляющий автомат будет вырабатывать сигналы “сложить” (Add) и “сдвинуть” (Shift). Это возможно, например, при загрузке числа 15D (1111BIN). На рис. 1.37 показан пример умножения чисел 10x10. Результат 100. По окончании процесса умножения вырабатывается сигнал готовности Done.

Разработаем цифровой автомат с использованием встроенного редактора состояний конечного автомата (рис. 1.38) и извлечем код языка VHDL в автоматическом режиме. Используется двухпроцессный шаблон.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
entity ShiftN is
port(CLK, CLR, LD, SH, DIR: in STD_LOGIC;
D: in std_logic_vector(3 downto 0);
Q: out std_logic_vector(7 downto 0));
end ShiftN;
architecture a of ShiftN is
  begin
  process (CLR, CLK)
  variable St: std_logic_vector(7 downto 0);
  subtype InB is natural range 3 downto 0;
      begin
```

```
            if CLR = '1' then
                St := (others => '0'); Q <= St;
            elsif CLK'EVENT and CLK='1' then
                if LD = '1' then
                    St:=(others=>'0');
                    St(InB) := D;
                    Q <= St;
                elsif SH = '1' then
                    case DIR is
               when '0' => St := '0' & St(St'LEFT downto 1);
               when '1' => St := St(St'LEFT-1 downto 0) & '0';
                    end case;
                    Q <= St;
                end if;
            end if;
        end process;
end a;
```

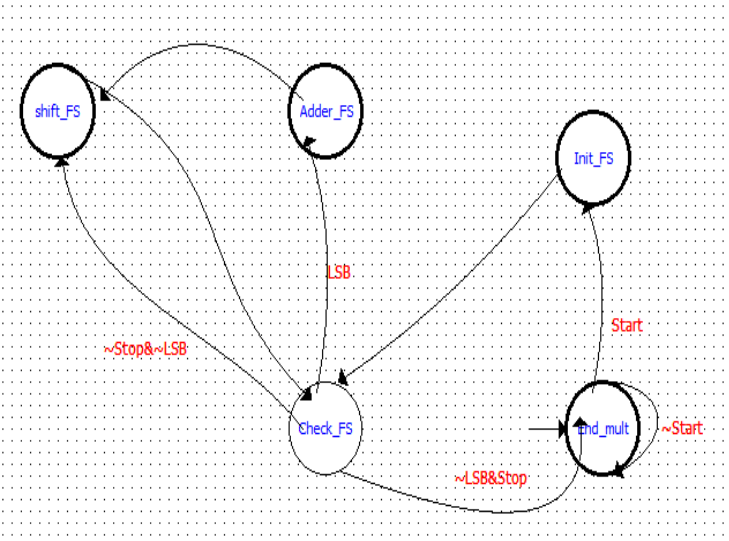
Пример 1. Сдвиговый регистр на языке VHDL

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY avt IS
   PORT (
      res : IN STD_LOGIC;
      clk : IN STD_LOGIC;
      Start : IN STD_LOGIC;
      LSB : IN STD_LOGIC;
      Stop : IN STD_LOGIC;
      Done : OUT STD_LOGIC;
      Init : OUT STD_LOGIC;
      Add : OUT STD_LOGIC;
      Shift : OUT STD_LOGIC);
END avt;
ARCHITECTURE BEHAVIOR OF avt IS
```

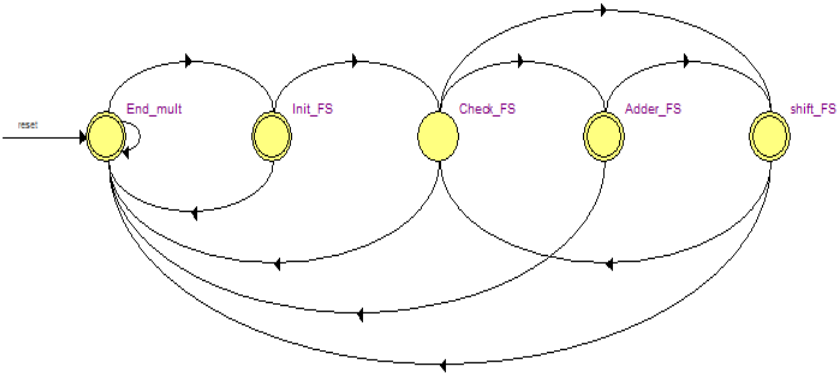
```
TYPE type_fstate IS (Check_FS,Init_FS,Adder_FS,shift_FS,End_mult);
   SIGNAL fstate : type_fstate;
   SIGNAL reg_fstate : type_fstate;
BEGIN
Init <='1' when reg_fstate = Init_FS else '0';
Add <='1' when reg_fstate = Adder_FS else '0';
Shift <='1' when reg_fstate = shift_FS else '0';
Done <='1' when reg_fstate = End_mult else '0';
process (clk, res) begin
if res = '1' then reg_fstate <= End_mult;
elsif clk'event and clk = '1' then
case reg_fstate is
when  Init_FS => reg_fstate <= Check_FS;
when Check_FS =>
                    if LSB = '1' then reg_fstate <= Adder_FS;
                    elsif Stop ='0' then reg_fstate <= shift_FS;
                    else reg_fstate <= End_mult;
                    end if;
when Adder_FS => reg_fstate <= shift_FS;
when shift_FS => reg_fstate <= Check_FS;
when End_mult => if Start = '1' then reg_fstate <= Init_FS; end if;
end case;
end if;
end process;
END BEHAVIOR;
```

Пример 2. Код языка VHDL управляющего автомата

Первый оператор Process описывает блок регистров (память состояний) для хранения состояний автомата. Второй оператор Process используется для описания логики переходов и логики формирования выхода (пример 3). Тестирование умножителя на примере умножения 5x5 показано на рис. 1.39. Общие сведения по числу задействованных ресурсов в проекте показаны в табл. 1.6.



а)



б)

Рис. 1.38. Граф-автомат, разработанный с помощью редактора состояний (а) и синтезированный граф-автомат (меню Netlist Viewers/State Machine Viewer)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY avt_flow IS
PORT (
reset : IN STD_LOGIC := '0';
clock : IN STD_LOGIC;
Start : IN STD_LOGIC := '0';
LSB : IN STD_LOGIC := '0';
Stop : IN STD_LOGIC := '0';
Done : OUT STD_LOGIC;
Init : OUT STD_LOGIC;
Add : OUT STD_LOGIC;
Shift : OUT STD_LOGIC
   );
END avt_flow;
ARCHITECTURE BEHAVIOR OF avt_flow IS
TYPE type_fstate IS (Check_FS,Init_FS,Adder_FS,shift_FS,End_mult);
   SIGNAL fstate : type_fstate;
   SIGNAL reg_fstate : type_fstate;
BEGIN
   PROCESS (clock,reg_fstate)
   BEGIN
      IF (clock='1' AND clock'event) THEN
         fstate <= reg_fstate;
      END IF;
   END PROCESS;
   PROCESS (fstate,reset,Start,LSB,Stop)
   BEGIN
      IF (reset='1') THEN
         reg_fstate <= End_mult;
         Done <= '0';
         Init <= '0';
         Add <= '0';
         Shift <= '0';
      ELSE
         Done <= '0';
         Init <= '0';
         Add <= '0';
         Shift <= '0';
```

```
CASE fstate IS
   WHEN Check_FS =>
      IF ((NOT((LSB = '1')) AND (Stop = '1'))) THEN
         reg_fstate <= End_mult;
      ELSIF ((LSB = '1')) THEN
         reg_fstate <= Adder_FS;
      ELSIF ((NOT((Stop = '1')) AND NOT((LSB = '1')))) THEN
         reg_fstate <= shift_FS;
      -- Inserting 'else' block to prevent latch inference
      ELSE
         reg_fstate <= Check_FS;
      END IF;
   WHEN Init_FS =>
      reg_fstate <= Check_FS;
      Init <= '1';
   WHEN Adder_FS =>
      reg_fstate <= shift_FS;
      Add <= '1';
   WHEN shift_FS =>
      reg_fstate <= Check_FS;
      Shift <= '1';
   WHEN End_mult =>
      IF ((Start = '1')) THEN
         reg_fstate <= Init_FS;
      ELSIF (NOT((Start = '1'))) THEN
         reg_fstate <= End_mult;
      -- Inserting 'else' block to prevent latch inference
      ELSE
         reg_fstate <= End_mult;
      END IF;
      Done <= '1';
```

```
            WHEN OTHERS =>
               Done <= 'X';
               Init <= 'X';
               Add <= 'X';
               Shift <= 'X';
               report "Reach undefined state";
         END CASE;
      END IF;
   END PROCESS;
END BEHAVIOR;
```

Пример 3. VHDL-код, извлеченный в автоматическом режиме из граф-автомата, созданного с помощью редактора состояний в САПР Quartus II

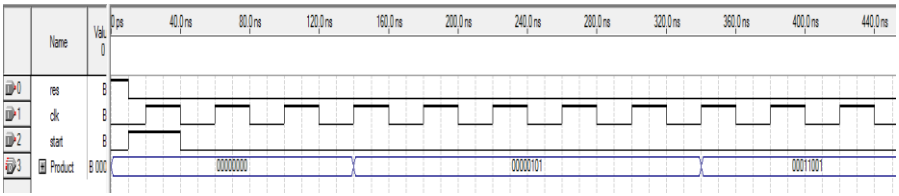


Рис. 1.39. Тестирование умножителя на примере умножения 5x5. Результат 25

Таблица 1.6

Общие сведения по числу задействованных ресурсов ПЛИС Cyclone EP1C3T144C8N

Логические элементы (Logic Cells, ЛЭ)	Триггеры логических элементов (LC Registers)	Таблицы перекодировок (LUT-only LC)	Рабочая частота в наихудшем случае Fmax, МГц
47	41	6	275

Стенд подключается к персональному компьютеру через порт USB. Для записи файла конфигурации в память ПЛИС через порт USB персонального компьютера требуется преобразовать *.sof- файл в формат с расширением *.rbf. Загрузка конфигурационного файла в ПЛИС производится с помощью отдельной программы – загрузчика (l2flash.exe).

Входные и выходные контакты к внешним выводам ПЛИС подключены с помощью меню Assignments/Pins (рис. 1.40). Из-за того, что стенд имеет 8 переключателей S1-S8, пришлось отказаться от загрузки чисел с внешних портов (4-разрядные сигналы A и B) и от сигнала Done, так как доступно всего лишь 8 светодиодов. Умножаемые числа предварительно сохраняются в константах (мегафукция LPM_constant). Светодиоды LED1-LED8 отображают результат умножения (8-разрядный сигнал Product[7..0]). В проекте принято, что LED8 (pin 121) - младший значащий разряд.

Для подачи тактовых импульсов с помощью кнопки Bottom необходимо использовать фильтр (блок Antitinkling). Данный блок предназначен для подавления дребезга контактов. Из-за этого явления непосредственное подключение кнопки с механическим замыканием контактов к цифровой схеме не всегда допустимо. Суть дребезга заключается в многократном неконтролируемом замыкании и размыкании контактов в момент коммутации, в результате чего на цифровую схему подается множество импульсов вместо одного.

Частота тактового генератора в учебных стендах LESO2 равна 6 МГц, в стендах LESO2.1 и LESO2.3 – 50 МГц. Делитель частоты должен обеспечить интервал между импульсами больше, чем длительность дребезга и менее чем длительность нажатия кнопки. На рис. 1.41 показана схема подавителя дребезга с использованием суммирующего

счетчика-делителя частоты и временные диаграммы его работы. В нашем случае 19-разрядный счетчик обеспечивает коэффициент счета 524287 и выходной сигнал cout с пониженной частотой 95,37 Гц (100 Гц – период 10 мс). Время дребезга кнопки примерно составляет 2 мс.

На рис. 1.42 и 1.43 показано тестирование умножителя на примере умножения 5x5. Тестирование осуществляется следующим образом. Согласно рис. 1.39 щелкаем переключателем S2 (pin 50), выполняющим роль асинхронного сигнала res. Переводим в верхнее положение переключатель S3 (pin 51) – сигнал start, далее нажимаем на кнопку Button (pin 37) один раз, происходит загрузка чисел в умножитель. Переводим переключатель S3 в нижнее положение. Щелкаем три (рис. 1.42) и пять раз (рис. 1.43) кнопкой Button для имитации подачи синхросигнала. Итоговый результат умножения десятичное число 25, а процесс умножения осуществляется за 9 тактов синхрочастоты.

Запрограммировать ПЛИС возможно с помощью Altera USB Blaster без предварительного преобразования *.sof- файла в формат *.rbf. Программирование осуществляется непосредственно в САПР Quartus II (меню Tools/Programmer). В этом случае питание лабораторного стенда LESO2.1 осуществляется через USB-кабель а программирование осуществляется через JTAG-интерфейс.

Учебный лабораторный стенд LESO2.1 отечественной разработки содержит хороший функциональный набор для занятий по цифровой схемотехнике и может быть использован для изучения основ проектирования комбинационных и последовательностных устройств в базисе ПЛИС.

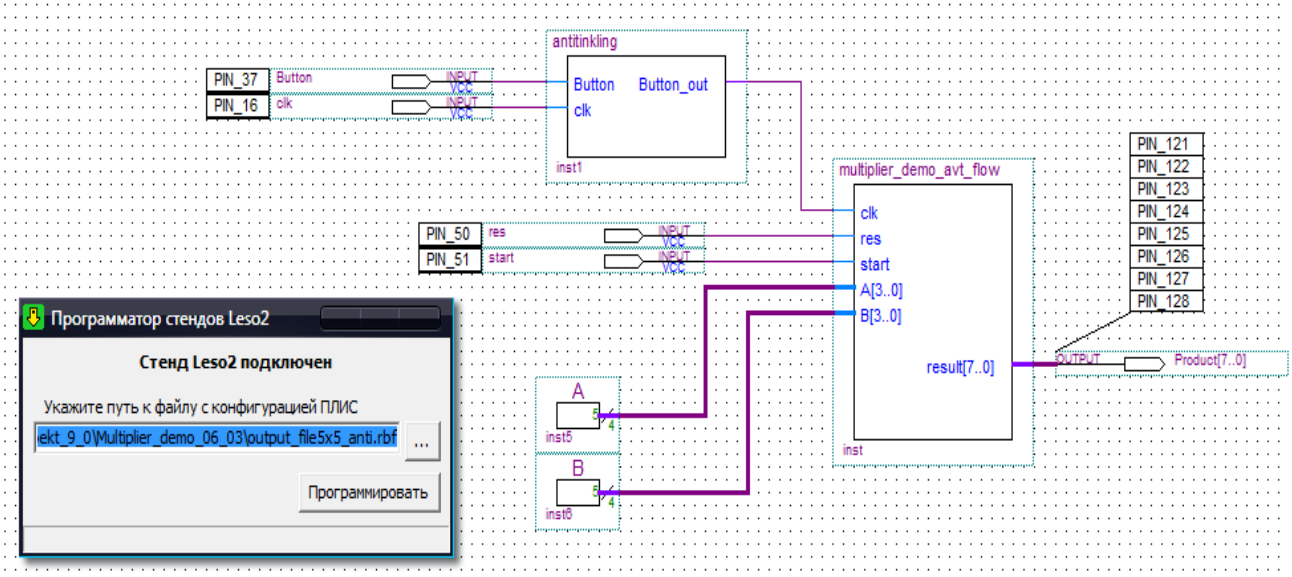
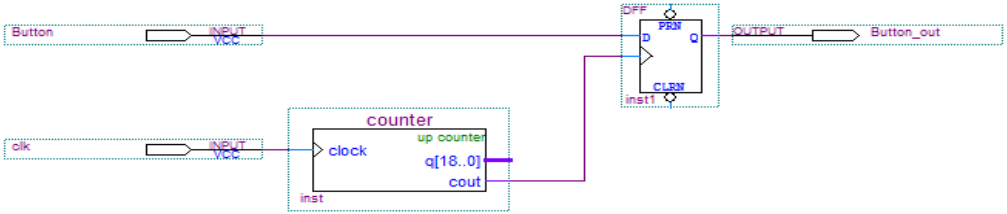
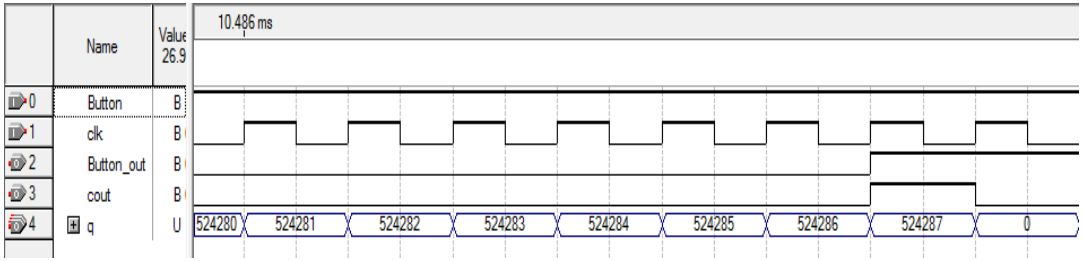


Рис. 1.40. Схема умножителя с подключенными внешними выводами



а)



б)

Рис. 1.41. Подавитель дребезга с использованием суммирующего счетчика-делителя частоты (а) и временные диаграммы его работы (б)

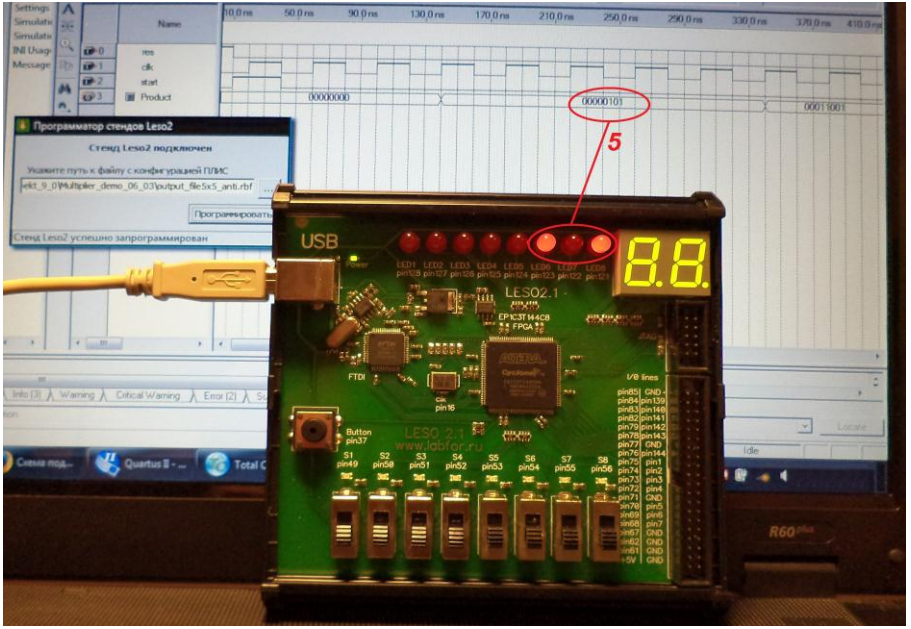


Рис. 1.42. Тестирование умножителя на примере умножения 5x5. Промежуточный результат 5

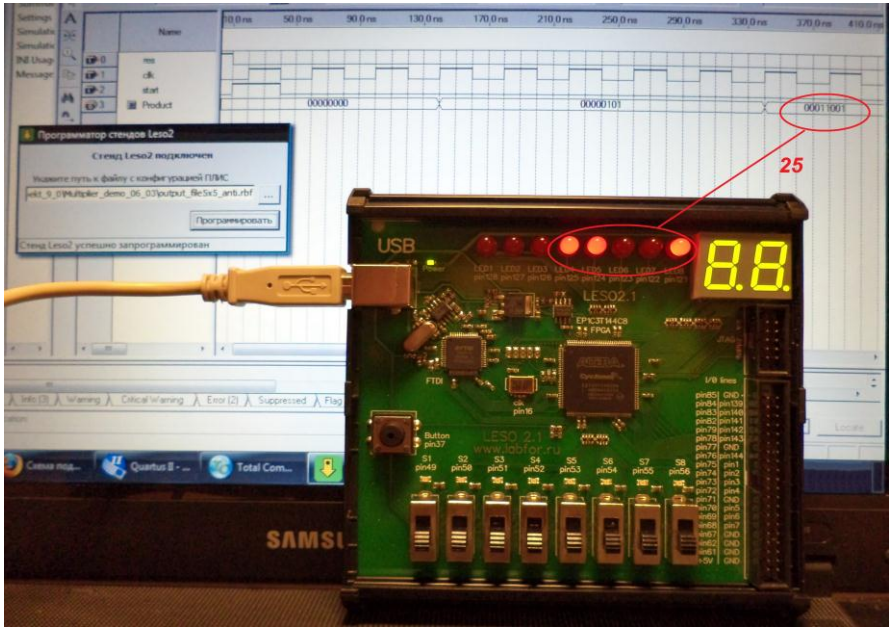


Рис. 1.43. Тестирование умножителя на примере умножения 5x5. Итоговый результат 25

2. ПРОЕКТИРОВАНИЕ ЦИФРОВЫХ ФИЛЬТРОВ В БАЗИСЕ ПЛИС

2.1. Расчет параметров КИХ-фильтров с использованием среды FDATool системы визуально-имитационного моделирования Matlab/Simulink

Рассмотрим особенности проектирования КИХ-фильтра в системе Matlab/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. 2.1 показана амплитудно-частотная характеристика (АЧХ) КИХ-фильтра. Серые области на рис. 2.1 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:

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

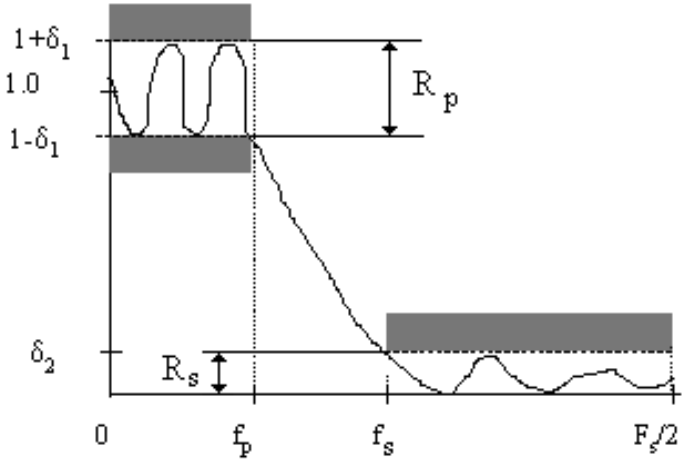


Рис. 2.1. Амплитудно-частотная характеристика фильтра нижних частот

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. 2.1. Пример расчета КИХ-фильтра в среде FDATool показан на рис. 2.2.

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 2.1

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. Преимущество мегаядра в том, что порядок проектируемого КИХ-фильтра (число отводов) оценивается автоматически, но синтез АЧХ осуществляется только методом окон.

Этот недостаток компенсируется возможностью загрузки коэффициентов проектируемого фильтра, полученных, например, с использованием среды FDATool. 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 зададим порядок КИХ-фильтра. Порядок КИХ-фильтра зададим тот, который рекомендует выбрать мегафункция Mega Core FIR. Мегафункция также предлагает и метод синтеза (окно Blackman - Блекмена). Расчет фильтра осуществляется нажатием кнопки Design Filter. На рис. 2.2 показана АЧХ, вычисленная с использованием формата с плавающей (штрих-

пунктирная линия) и формата с фиксированной запятой (непрерывная линия).

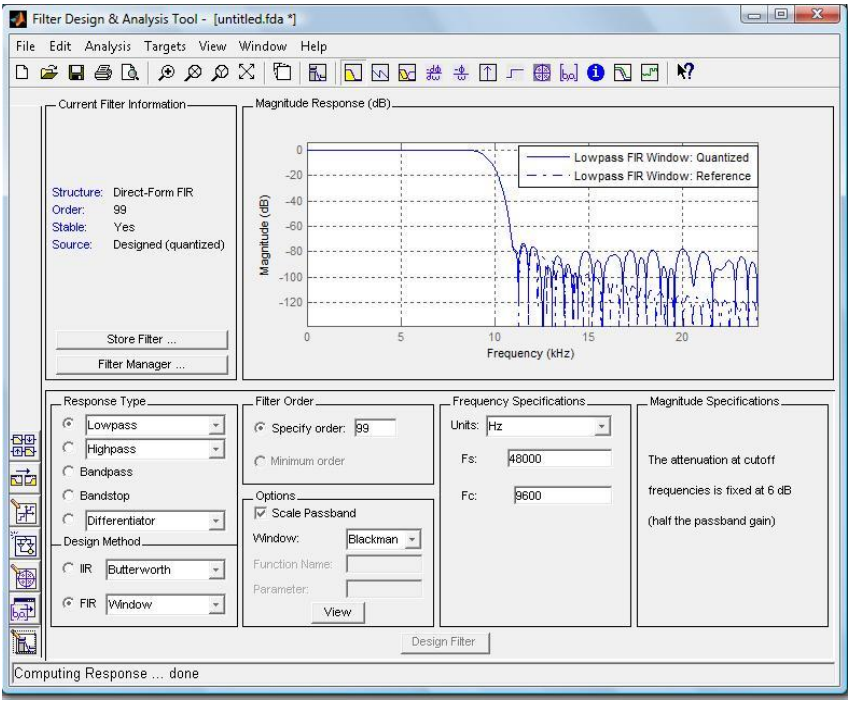


Рис. 2.2. Интерфейс среды FDATool. Пример расчета АЧХ КИХ-фильтра

На рис. 2.3 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 настройках мегаядра Mega Core FIR Compiler задается переходная полоса (Transition Bandwith) равная 2400 Гц и частота среза равная 9600 Гц (обозначается как cutoff freq (1)).

В методе окон $|H(f)|$ обратное преобразование Фурье этой характеристики дает бесконечную в обе стороны последовательность отсчетов импульсной характеристики. 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.

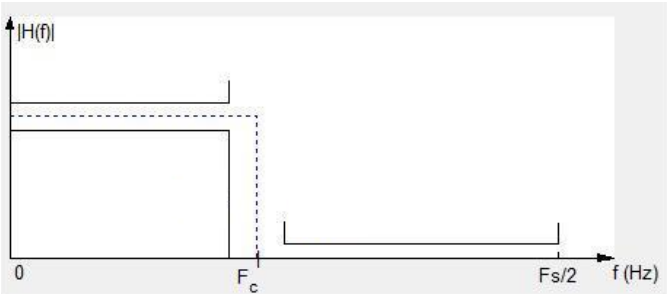


Рис. 2.3. Характеристики синтезируемой АЧХ (окно Blackman) КИХ-фильтра в среде FDATool

Вкладка Realize Model позволяет импортировать спроектированный КИХ-фильтр (модель) в Simulink (рис. 2.2). В этом случае строится модель КИХ-фильтра с использованием базовых элементов (задержка, сумма, коэффициент усиления). Меню Targets опция Generate HDL позволяют сгенерировать код фильтра на языке VHDL. Можно заказать параллельную архитектуру КИХ-фильтра, обладающего высокой производительностью.

В данном разделе рассмотрены расчет спецификации КИХ-фильтров в системе Matlab/Simulink с применением пакета Signal Processing среды FDATool.

2.2. Проектирование квантованных КИХ-фильтров

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

На рис. 2.4 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, зашумленный шумом (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 Matlab.

```
>> 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); зашумленный сигнал
>> plot(s1);
```

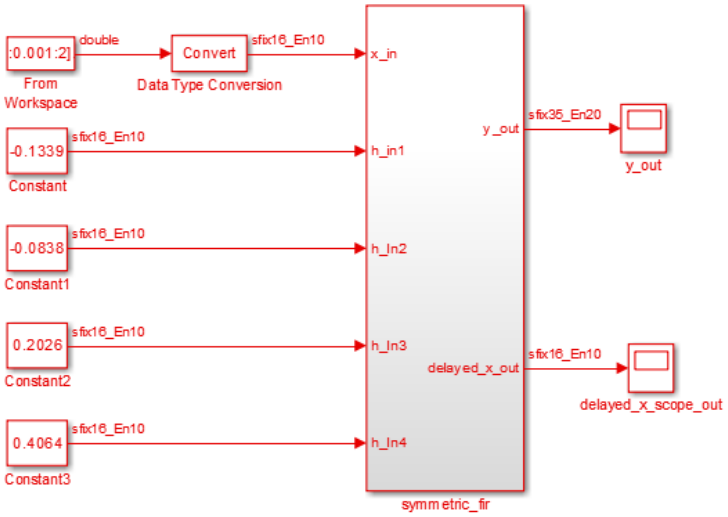
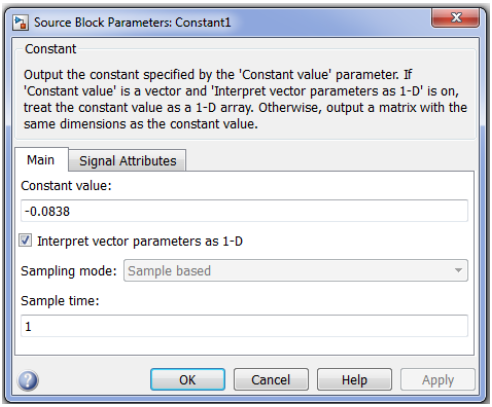


Рис. 2.4. Имитационная модель симметричного КИХ-фильтра на восемь отводов. Верхний уровень иерархии

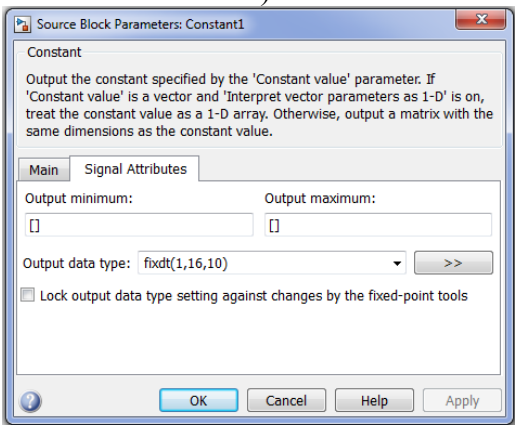
Предположим, что коэффициенты фильтра известны и хранятся в функциональных блоках Constant с одноименными именами Constant – Constant3. Выходные сигналы представляются в формате с фиксированной запятой fixdt(1,16,10): h1 = -0.1339; h2 = -0.0838; h3 = 0.2026; h4 = 0.4064. Преобразовывать значения из формата с плавающей запятой в формат с фиксированной запятой позволяют “автоматизированные мастера”, встроенные в функциональные блоки (рис. 2.5).

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. 2.5. Настройка функционального блока Constant с именем Constant1: а) задается вещественное число -0.083, 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 дальнейшим 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. 2.6. 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, т.к. 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. 2.6).

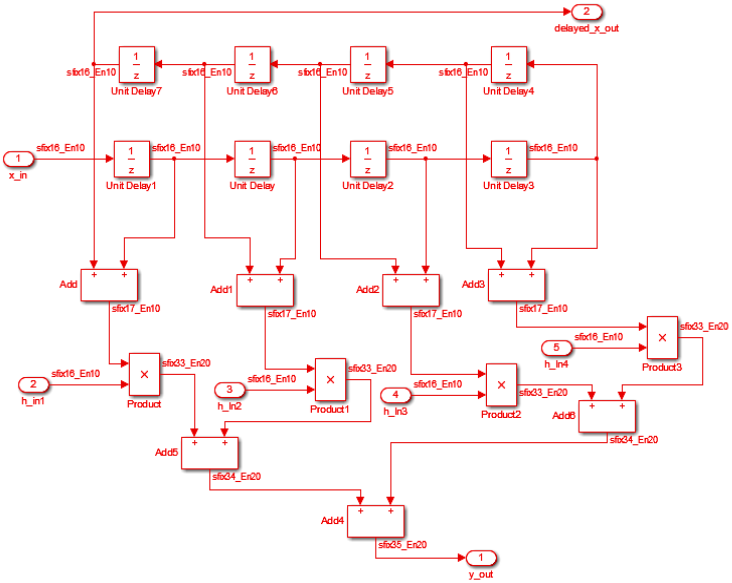


Рис. 2.6. Имитационная модель симметричного КИХ-фильтра на восемь отводов с указанием формата сигналов. Нижний уровень иерархии. Структура фильтра в элементах системы Matlab/Simulink

Далее с помощью приложения Simulink HDL Coder системы Matlab/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. Разработаем имитационную модель симметричного КИХ-фильтра на восемь отводов в формате с фиксированной запятой с использованием fi-объектов и языка М-файлов системы Matlab (рис. 2.7 и рис. 2.8). 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 использование 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);
```

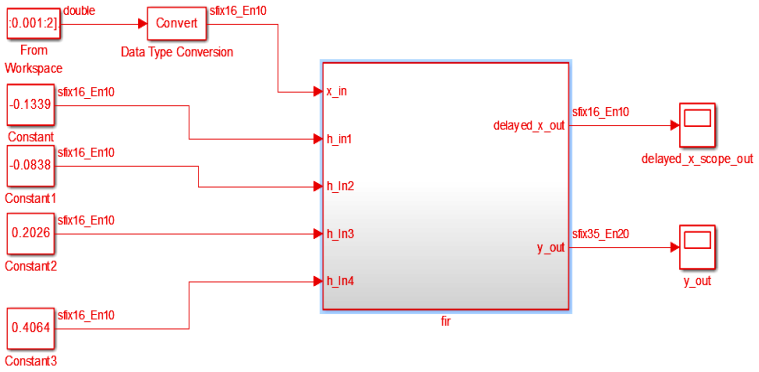


Рис. 2.7. Верхний уровень иерархии имитационной модели симметричного КИХ-фильтра на восемь отводов с использованием М-файла

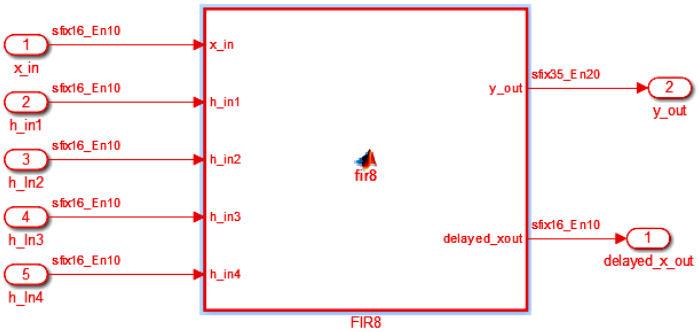


Рис. 2.8. Подсистема симметричного КИХ-фильтра на восемь отводов (подключение входных и выходных портов к М-файлу)

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. 2.9 показан сигнал до и после фильтрации.

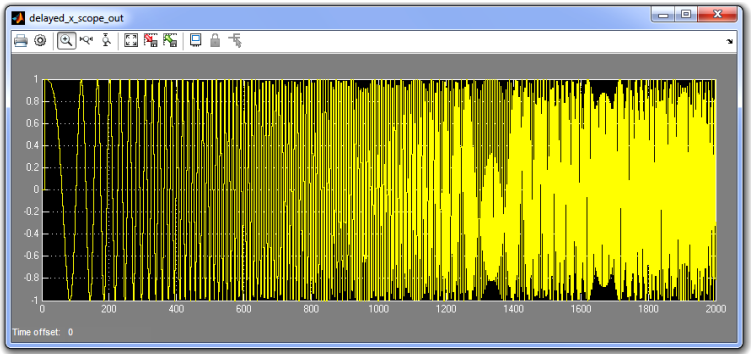
```
%#codegen
function [y_out, delayed_xout] = fir8(x_in, h_in1, h_in2, h_in3, h_in4)
% Symmetric FIR Filter
% HDL specific fimath
hdl_fm = fimath(...
'RoundMode', 'floor',...
'OverflowMode', 'wrap',...
```

```
'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;
```

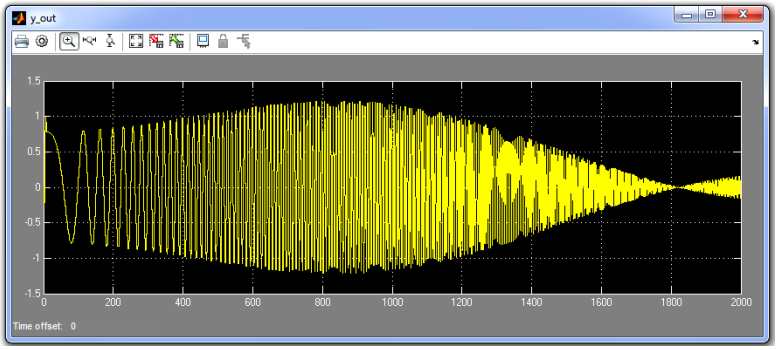
```
ud1 = fi(x_in, 1, 16, 10, hdl_fm);
end
```

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

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 (пример 2).



а)



б)

Рис. 2.9. Результаты имитационного моделирования: а – исходный сигнал; б - профильтрованный сигнал

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.numeric_std.ALL;
USE work.fir_tb_pkg.ALL;
PACKAGE fir_tb_data IS
CONSTANT Data_Type_Conversion_out1_force :
Data_Type_Conversion_out1_type;
CONSTANT Constant_out1_force : std_logic_vector(15
DOWNTO 0);
CONSTANT Constant1_out1_force : std_logic_vector(15
DOWNTO 0);
CONSTANT Constant2_out1_force : std_logic_vector(15
DOWNTO 0);
CONSTANT Constant3_out1_force : std_logic_vector(15
DOWNTO 0);
CONSTANT delayed_x_out_expected :
Data_Type_Conversion_out1_type;
CONSTANT y_out_expected : y_out_type;
END fir_tb_data;
PACKAGE BODY fir_tb_data IS
        -- Входной сигнал
CONSTANT Data_Type_Conversion_out1_force :
Data_Type_Conversion_out1_type :=
            (
                X"0400",
                X"03ff",
                X"03ff",
                X"03ff",
                X"03ff",
                X"03ff",
                X"03fe",
```

```
                X"03fd",
                X"03fc",
                X"03fb",
                X"03f9",
                X"03f7",
                X"03f5",
                X"03f2",
                X"03ef",
                X"03eb",
        -- Коэффициенты фильтра
CONSTANT    Constant_out1_force    :    std_logic_vector(15
DOWNTO 0) :=( X"ff77");
CONSTANT    Constant1_out1_force    :    std_logic_vector(15
DOWNTO 0) :=(X"ffaa");
CONSTANT    Constant2_out1_force    :    std_logic_vector(15
DOWNTO 0) :=(X"00cf");
CONSTANT    Constant3_out1_force    :    std_logic_vector(15
DOWNTO 0) :=(X"01a0");
CONSTANT delayed_x_out_expected :
Data_Type_Conversion_out1_type :=
            (  X"0000",
               X"0000",
                X"0000",
                X"0000",
                X"0000",
                X"0000",
                X"0000",
                X"0000",
                X"0400",
                X"03ff",
                X"03ff",
```

```
        X"03ff",
-- Отклик фильтра, профильтрованные значения
CONSTANT y_out_expected : y_out_type :=
   (    SLICE(X"000000000",35),
        SLICE(X"7fffddc00",35),
        SLICE(X"7fffc8489",35),
        SLICE(X"7ffffc0df",35),
        SLICE(X"000064010",35),
        SLICE(X"0000cbe70",35),
        SLICE(X"0000ff8d0",35),
        SLICE(X"0000ea08a",35),
        SLICE(X"0000c7dbf",35),
        SLICE(X"0000c7e58",35),
        SLICE(X"0000c7cc8",35),
……
```

Пример 2. Фрагмент тестбенча полученный с помощью приложения Simulink HDL Coder из имитационной модели симметричного КИХ-фильтра на восемь отводов на основе М-файлов и fi-объектов

Рассмотрим функциональное моделирование КИХ-фильтра с использованием симулятора ModelSim-Altera STARTER EDITION. На рис. 2.10 показано моделирование фильтра с использованием тестбенча (пример 2), полученного с помощью приложения Simulink HDL Coder из имитационной модели симметричного КИХ-фильтра на восемь отводов на основе М-файлов и fi-объектов.

Используя полученный код, разработаем функциональную модель в САПР ПЛИС Quartus II (рис. 2.11). Входной сигнал сформируем с помощью векторного редактора (рис. 2.12) в соответствии с примером 4. Проект размещен в ПЛИС EP2C5AF256A7 серии Cyclone II. На рис. 2.13 показано распределение задействованных ресурсов проекта по

кристаллу ПЛИС EP2C5AF256A7. Используются восемь аппаратных умножителей размерностью операндов 9x9. Что равносильно использованию четырех умножителей размерностью операндов 18x18 (табл. 2.2).

Таблица 2.2

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 9x9 (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. 2.10 и рис. 2.12, 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.

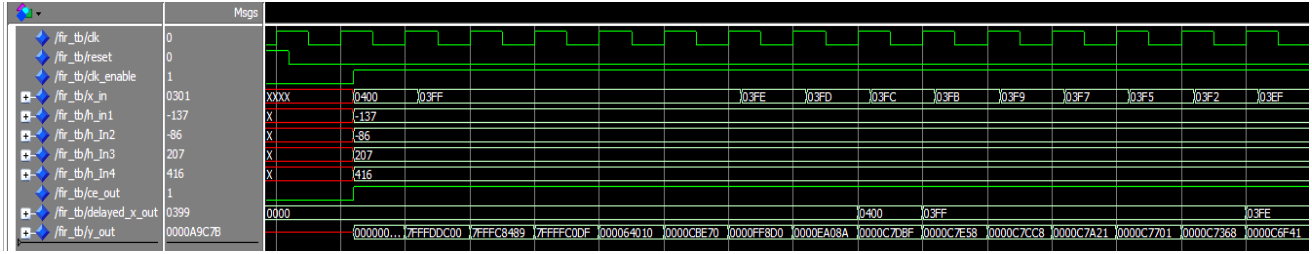


Рис. 2.10. Функциональное моделирование фильтра с использованием тестбенча, полученного с помощью приложения Simulink HDL Coder из имитационной модели симметричного КИХ-фильтра на восемь отводов на основе М-файлов и fi-объектов

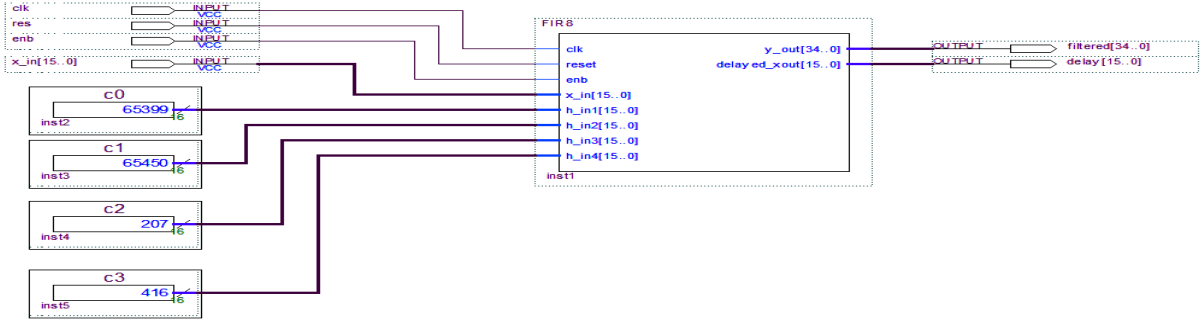


Рис. 2.11. Функциональная модель КИХ-фильтра на восемь отводов, построенная с использованием кода языка VHDL, извлеченного в автоматическом режиме с помощью приложения Simulink HDL Coder из имитационной модели симметричного КИХ-фильтра на восемь отводов на основе М-файлов и fi-объектов в САПР ПЛИС Quartus II версии 13.0

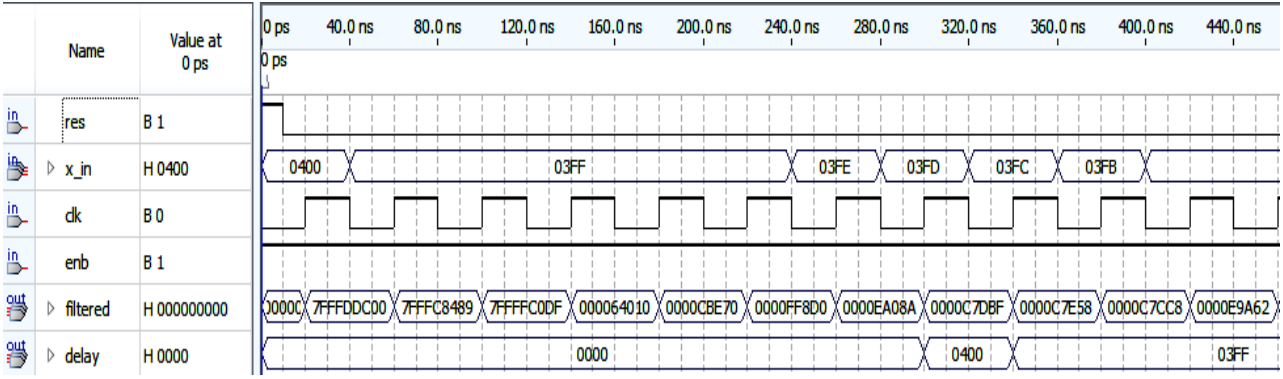


Рис. 2.12. Функциональное моделирование КИХ-фильтра с использованием кода языка VHDL, извлеченного в автоматическом режиме с помощью приложения Simulink HDL Coder из имитационной модели симметричного КИХ-фильтра на восемь отводов на основе M-файлов и fi-объектов

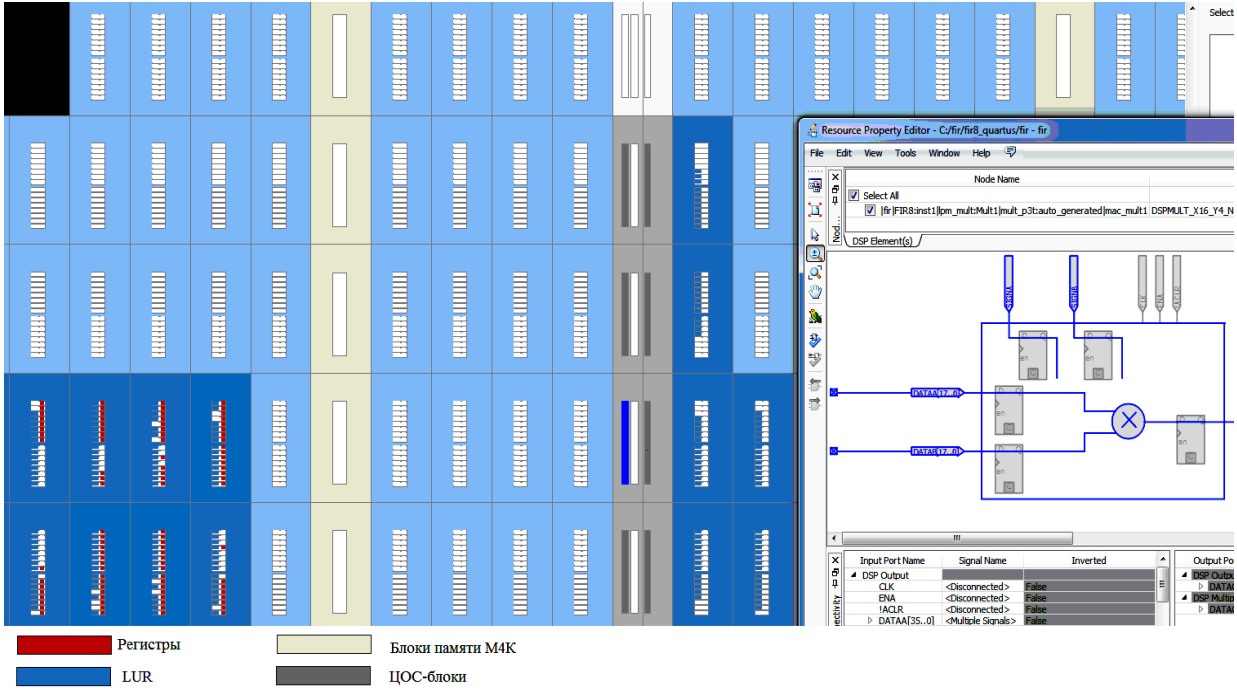


Рис. 2.13. Распределение задействованных ресурсов проекта по кристаллу ПЛИС EP2C5AF256A7

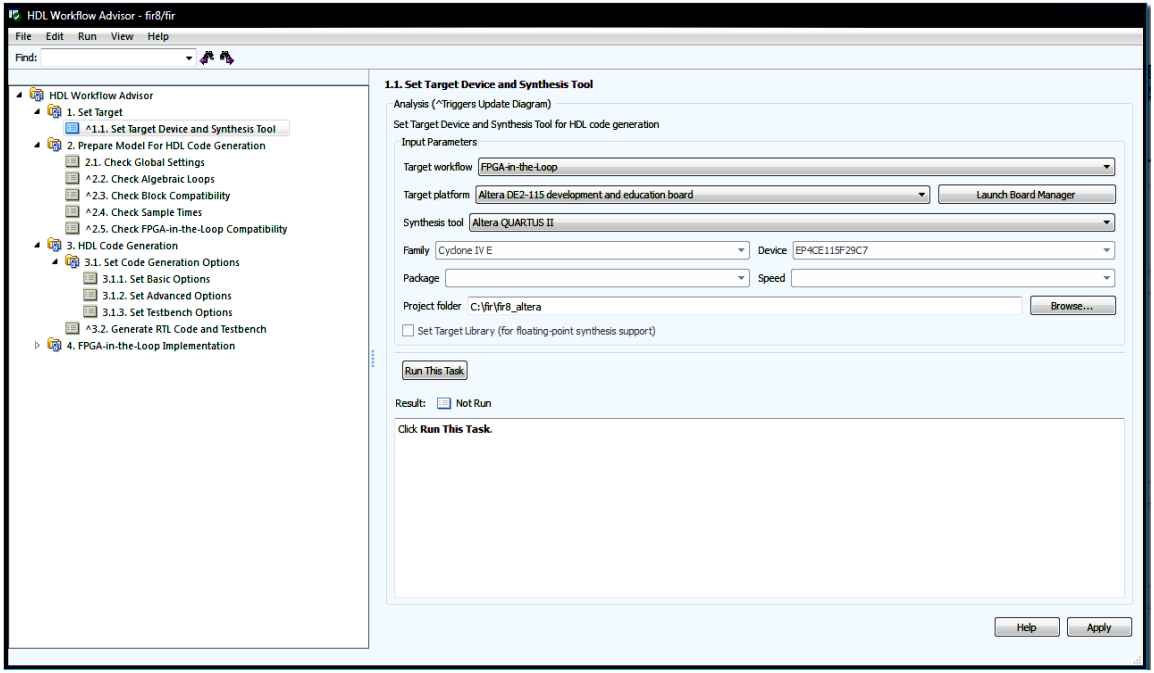


Рис. 2.14. Программный инструмент Workflow Advisor расширения Simulink

На рис. 2.14 показан программный инструмент HDL Workflow Advisor расширения Simulink. Более подробную информацию можно получить на сайте matlab.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. Проверка 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 Matlab/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.

2.3. Проектирование параллельных КИХ-фильтров

На рис. 2.15 показаны структуры фильтров, характерные для реализации в базисе сигнальных цифровых процессоров, а на рис. 2.16 показаны структуры фильтров, характерные для реализации в базисе ПЛИС. В качестве матричных умножителей могут быть использованы параллельные векторные умножители. На рис. 2.17 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 (рис. 2.18).

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.

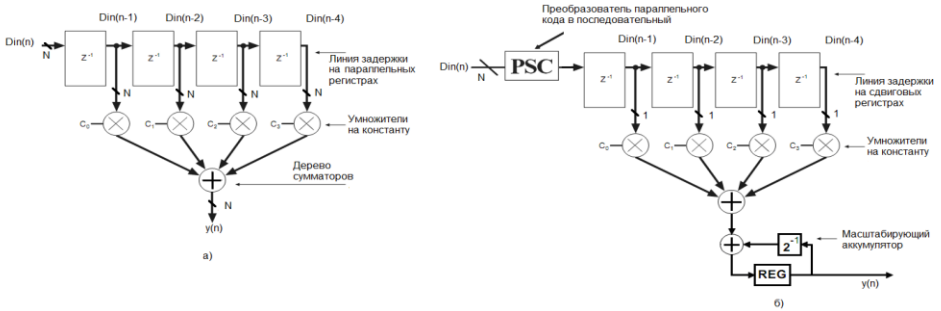


Рис. 2.15. Параллельный (а) и последовательный фильтры (б) на четыре отвода для реализации в базисе цифровых сигнальных процессоров

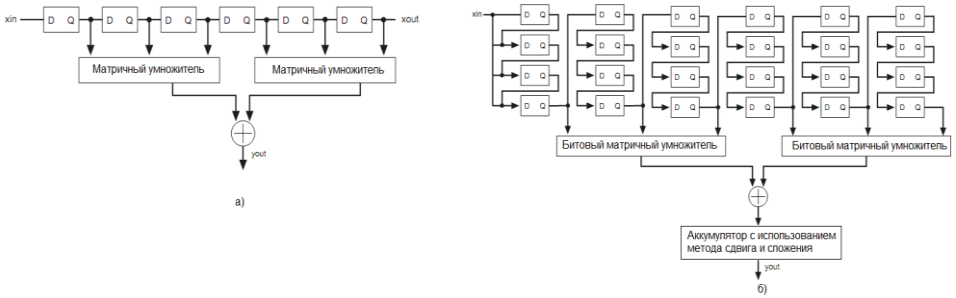


Рис. 2.16. Обобщенное представление структур КИХ-фильтров: а) параллельных; б) последовательных

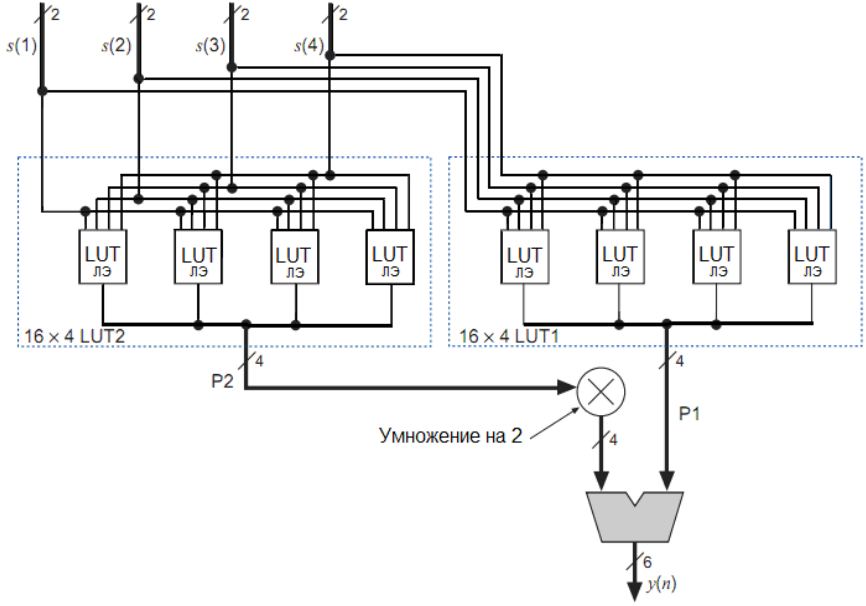


Рис. 2.17. Параллельный векторный умножитель четырех 2-разрядных сигналов на четыре 2-разрядные константы с использованием LUT ЛЭ в ПЛИС серии FLEX

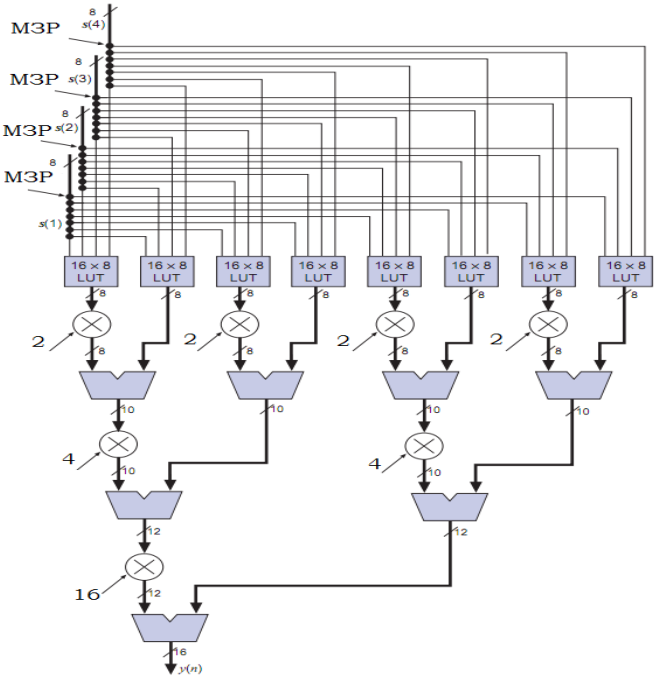


Рис. 2.18. Параллельный векторный умножитель четырех 8-разрядных сигналов на четыре 8-разрядные константы с использованием LUT ЛЭ в ПЛИС серии FLEX

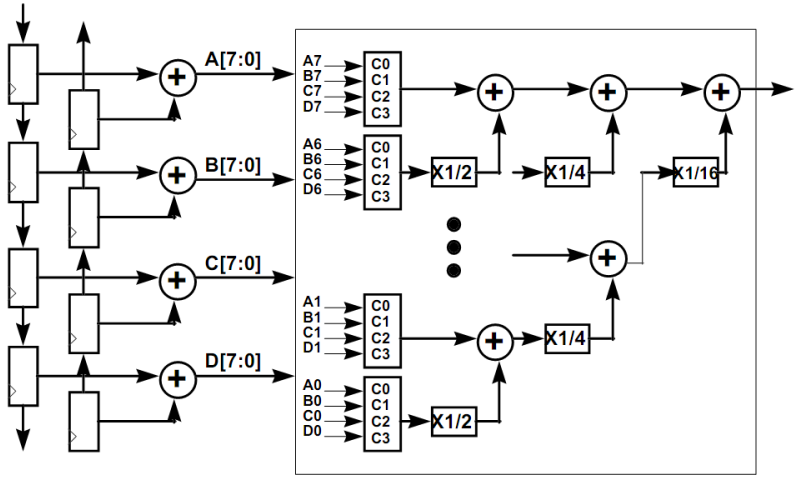
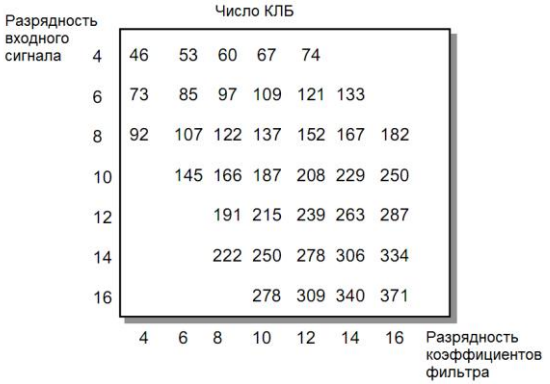
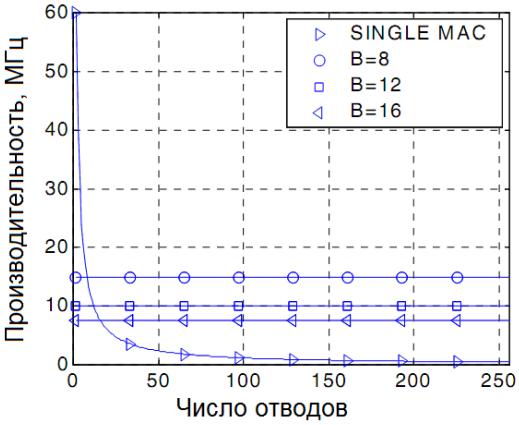


Рис. 2.19. Использование параллельного векторного умножителя четырех 8-разрядных сигналов на четыре 8-разрядные константы в составе КИХ-фильтра на восемь отводов с симметричными коэффициентами в базисе ПЛИС XC4000, построенного с использованием параллельной распределенной арифметики



а)



б)

Рис. 2.20. 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

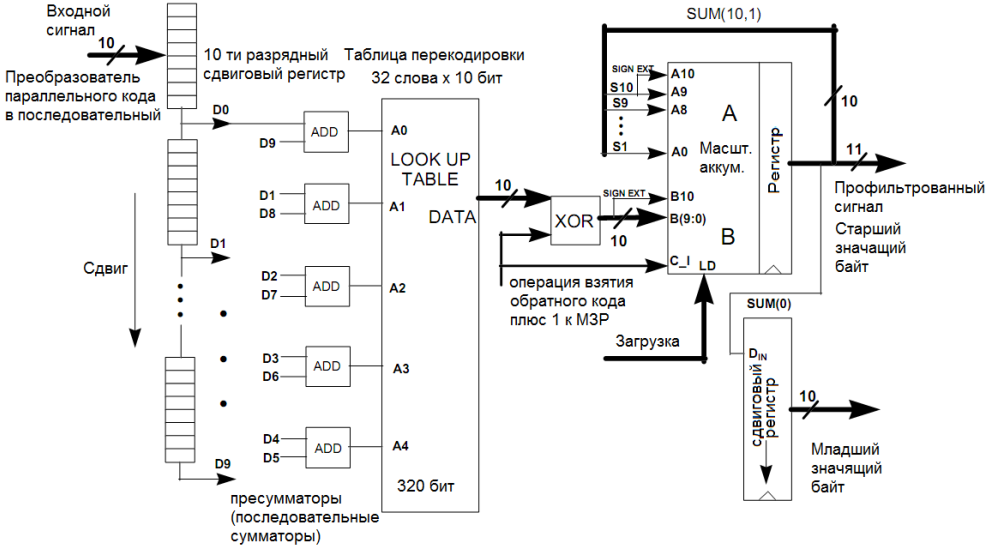


Рис. 2.21. Симметричный КИХ-фильтр со структурой десять отводов десять бит с точностью представления коэффициентов 10 бит с использованием последовательной распределенной арифметики

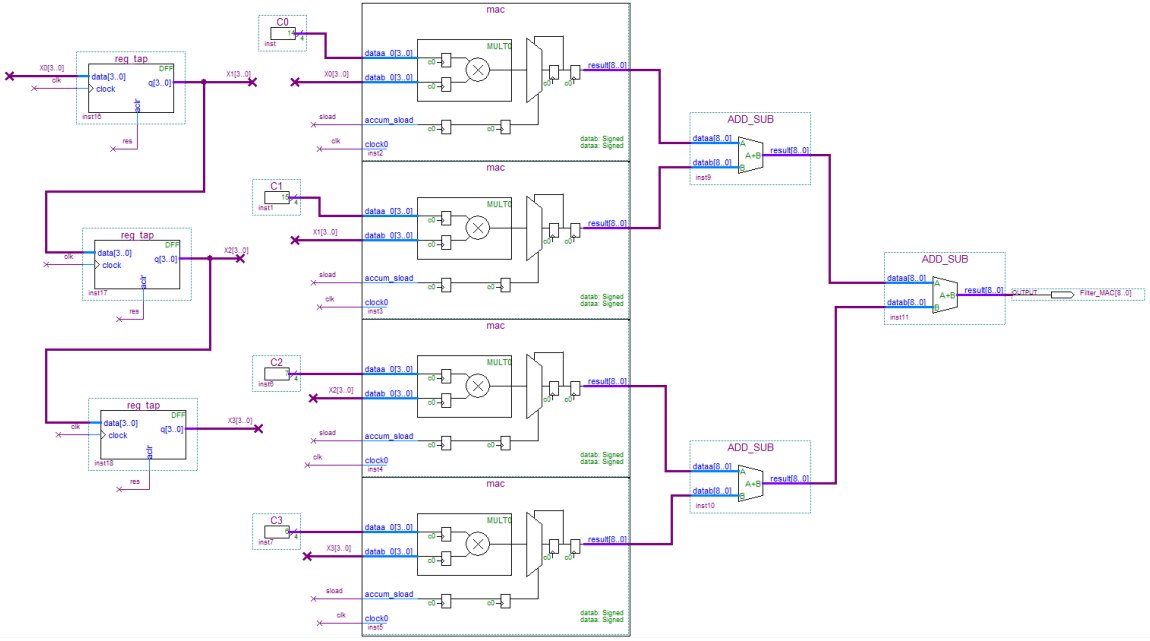


Рис. 2.22. Параллельная реализация КИХ-фильтра на четыре отвода с использованием четырех блоков в САПР ПЛИС Quartus II (мегафункция ALTMULT_ACCUM)

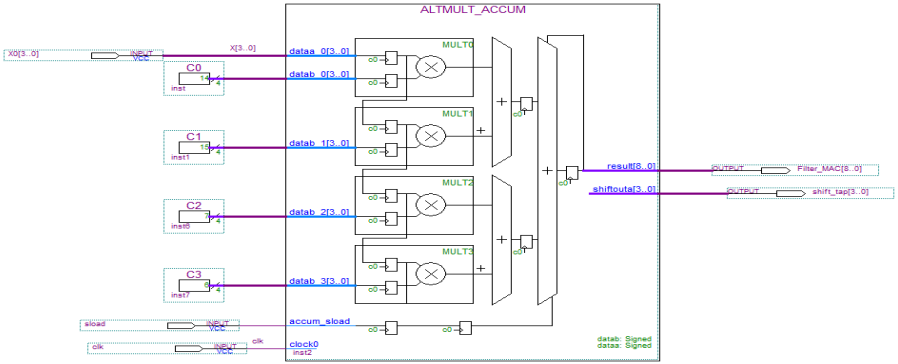


Рис. 2.23. Параллельная реализация КИХ-фильтра на четыре отвода с использованием четырех перемножителей в блоке (мегафункция ALTMULT_ACCUM, линия задержки построена на внутренних регистрах перемножителей MULT0-MULT3)

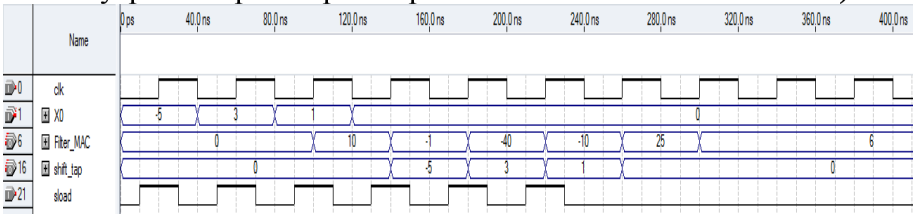


Рис. 2.24. Временные диаграммы работы фильтра на четыре отвода с использованием мегафункции ALTMULT_ACCUM

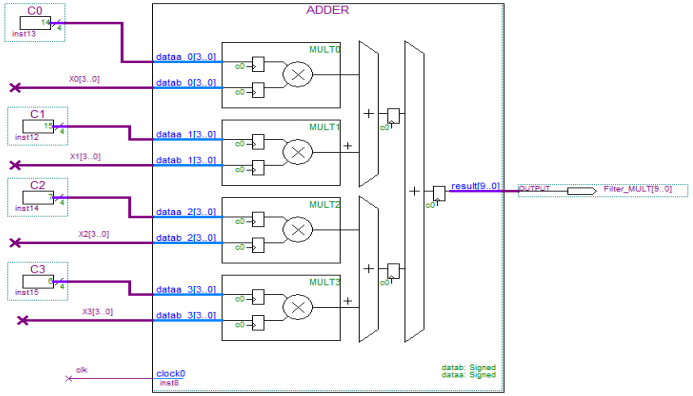


Рис. 2.25. Параллельная реализация КИХ-фильтра на четыре отвода с использованием четырех перемножителей в блоке (мегафункция ALTMULT_ADD, линия задержки такая же, как и на рис. 2.13)

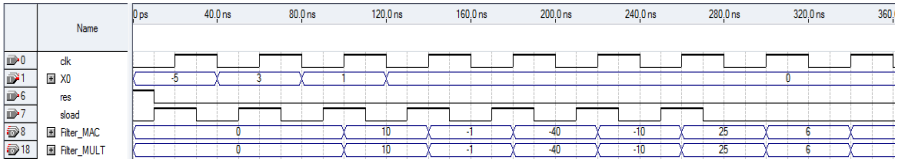


Рис. 2.26. Временные диаграммы работы параллельных фильтров на четыре отвода с использованием мегафункции ALTMULT_ACCUM и ALTMULT_ADD

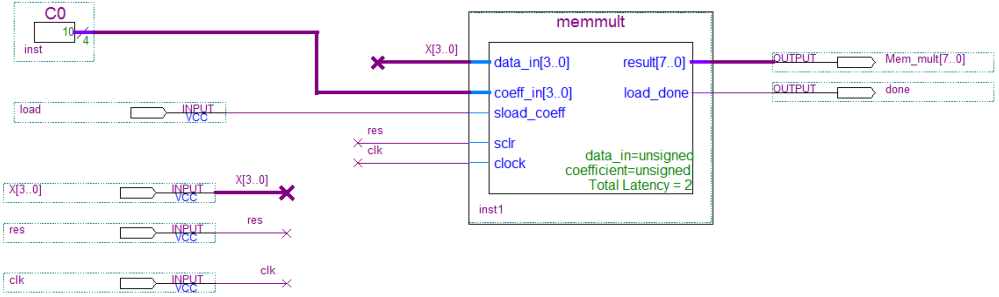


Рис. 2.27. Умножение 11 на 10 с помощью мегафункции ALTMEMMULT

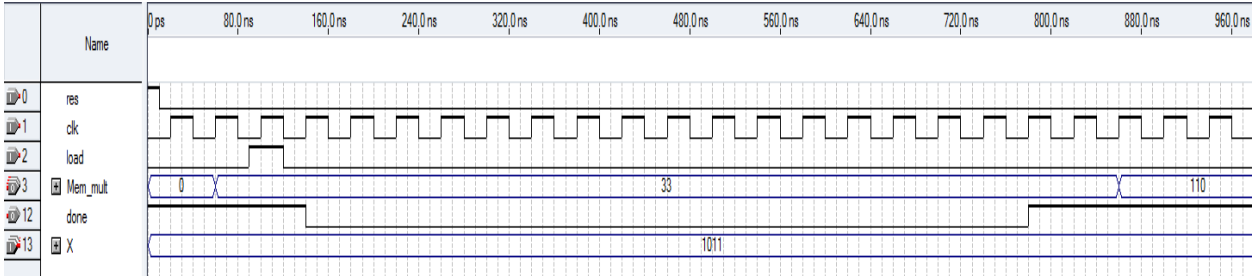


Рис. 2.28. Временные диаграммы умножения 11 на 10. По умолчанию в мегафункцию загружена константа 3. Результат 110

Рис. 2.19 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. Рис. 2.20, *а* 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. 2.20, *б*) применяется одна единственная LUT для вычисления частичных произведений (рис. 2.21). 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. Рис. 2.20, *б* 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.

Перемножители сигналов играют ключевую роль в проектировании высокопроизводительных цифровых фильтров.

Покажем различные варианты реализации КИХ-фильтров с использованием перемножителей 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.

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

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

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: $y = C_0 x_0 + C_1 x_1 + C_2 x_2 + C_3 x_3$ с использованием мегафункций САПР ПЛИС Quartus II компании Altera, объединенных общей идеей использования перемножителей 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 $y = C_0 x_0 + C_1 x_1 + C_2 x_2 + C_3 x_3$.

Первый вариант. Параллельная реализация КИХ-фильтра на четыре отвода с использованием четырех блоков умножения с накоплением. В проекте используются четыре мегафункции ALTMULT_ACCUM (рис. 2.22). Каждый блок использует один перемножитель 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. На рис. 2.22 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 КИХ-фильтра позволяет иное использование мегафункции ALTMULT_ACCUM. Фактически это одна мегафункция (блок с четырьмя перемножителями), в которой линия задержки организована на внутренних регистрах располагающихся на входах перемножителей. В мегафункции используются встроенные два сумматора и один сумматор-аккумулятор (рис. 2.23). В мегафункции ALTMULT_ACCUM используется три встроенных сумматора. Профильтрованные значения показаны на рис. 2.24.

Второй вариант. Параллельная реализация КИХ-фильтра на четыре отвода с использованием четырех перемножителей в блоке на мегафункции ALTMULT_ADD (функция умножения и сложения) в САПР ПЛИС Quartus II показана на рис. 2.25. Сравнивая временные диаграммы, видим, что профильтрованные значения на выходе у двух фильтров, построенных на разных мегафункциях, совпадают. Временные диаграммы работы фильтров, показанные на рис. 2.26, не отличаются от диаграмм на рис. 2.24.

Третий вариант. Рассмотрим умножение десятичного числа 11 на 10 на примере мегафункции ALTMEMMULT (рис. 2.27). 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, 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. 2.28).

Применяя мегафункцию ALTMEMMULT, разработаем параллельную реализацию КИХ-фильтра на четыре отвода с использованием четырех перемножителей (4 блока по 1 перемножителю в каждом) в САПР ПЛИС Quartus II и дерева сумматоров (рис. 2.29). Внешняя линия задержки состоит не из трех регистров, как в первых двух вариантах, а из четырех регистров. Дополнительно требуется, как и в первом варианте, три однотипных многоразрядных сумматора.

Латентность каждого умножителя равна двум. В каждый умножитель по умолчанию загружено число 0. Временные диаграммы работы фильтра на четыре отвода с использованием мегафункции ALTMEMMULT показаны на рис. 2.30. 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. 2.31). Временные диаграммы работы фильтра на четыре отвода с использованием мегафункции ALTMEMMULT показаны на

рис. 2.32. Сравнивая рис. 2.32 и рис. 2.30 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, т.к. требует 16 аппаратных перемножителей, три дополнительных сумматора и внешнюю линию задержки.

Наиболее оптимальным по числу используемых ресурсов ПЛИС является модификация варианта 1 (рис. 2.23), которая позволяет построить параллельный КИХ-фильтр на четыре отвода с использованием всего лишь одного блока со встроенными перемножителями в количестве четырех штук, двумя сумматорами, сумматором-аккумулятором и линией задержки. Мегафункции ALTMULT_ADD (рис. 2.25) так же позволяет построить параллельный КИХ-фильтр с использованием всего лишь одного блока со встроенными перемножителями и сумматорами. Использование внешней линии задержки из трех регистров приводит к незначительному увеличению ресурсов и не сказывается на быстродействие. Экономия ресурсов ПЛИС в первом модифицированном и во втором вариантах достигается за счет использования встроенных четырех аппаратных перемножителей размерностью 18x18.

Фильтр на мегафункции ALTMEMMULT с загрузкой коэффициентов из внешнего порта обладает пониженным быстродействием (рис. 2.20). Использование же блочной памяти (рис. 2.22) 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.

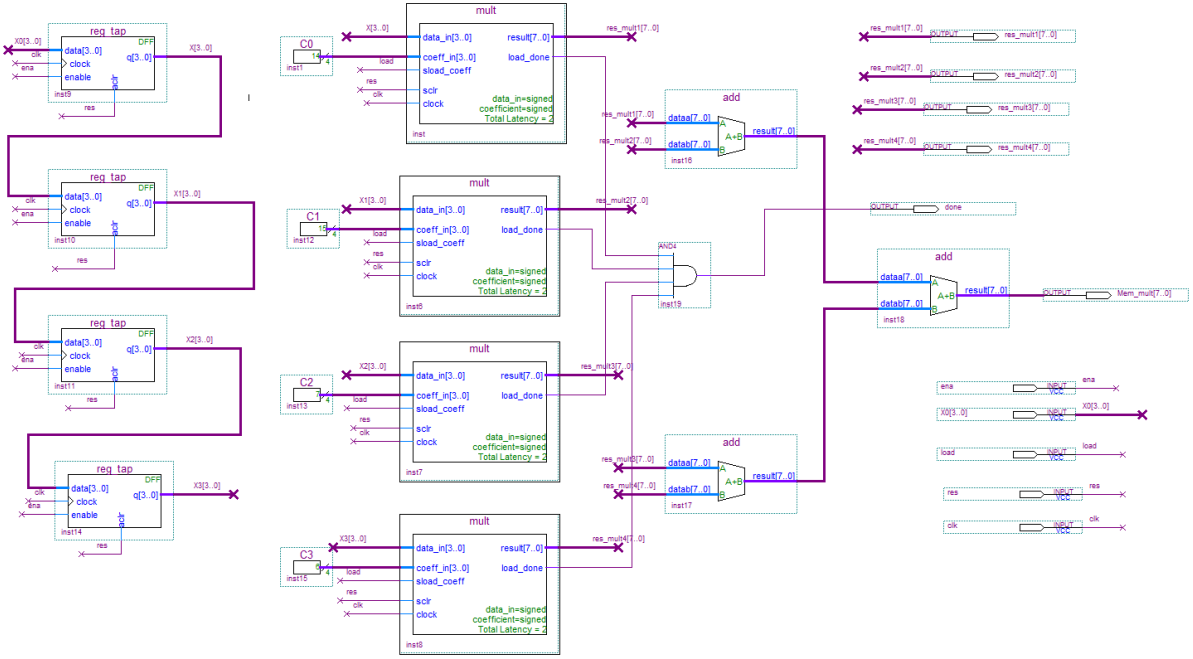


Рис. 2.29. Параллельная реализация КИХ-фильтра на четыре отвода с использованием четырех умножителей в САПР ПЛИС Quartus II (мегафункция ALTMEMMULT, линия задержки построена на четырех регистрах, коэффициенты фильтра загружаются из внешнего порта)

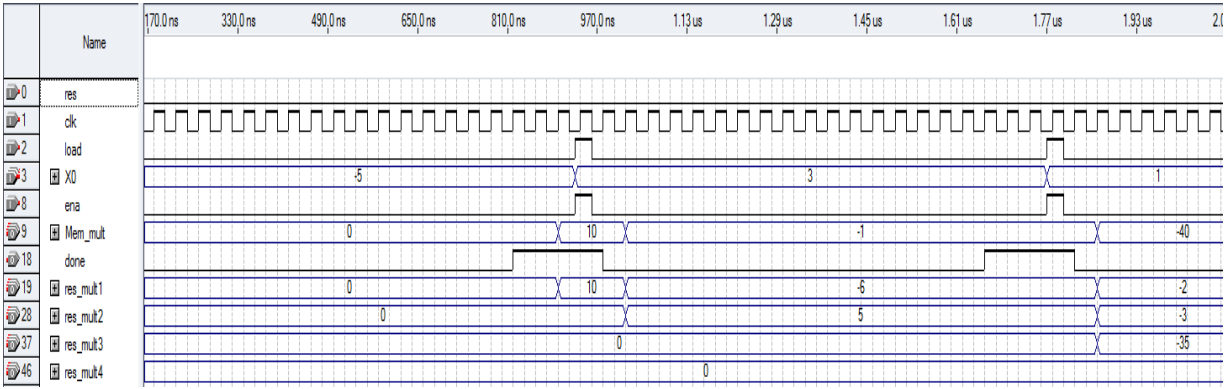


Рис. 2.30. Временные диаграммы работы фильтра на четыре отвода с использованием мегафункции ALTMEMMULT (коэффициенты фильтра загружаются из внешнего порта)

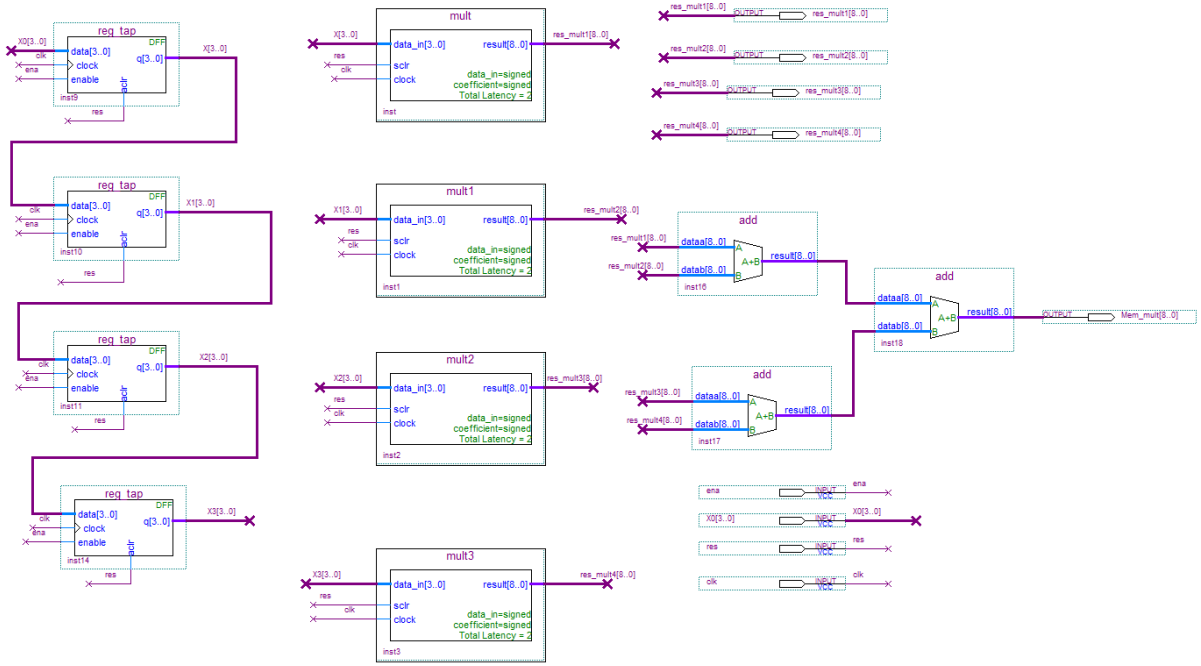


Рис. 2.31. Параллельная реализация КИХ-фильтра на четыре отвода с использованием четырех перемножителей в САПР ПЛИС Quartus II (мегафункция ALTMEMMULT, линия задержки построена на четырех регистрах, коэффициенты фильтра загружаются из блочной памяти)

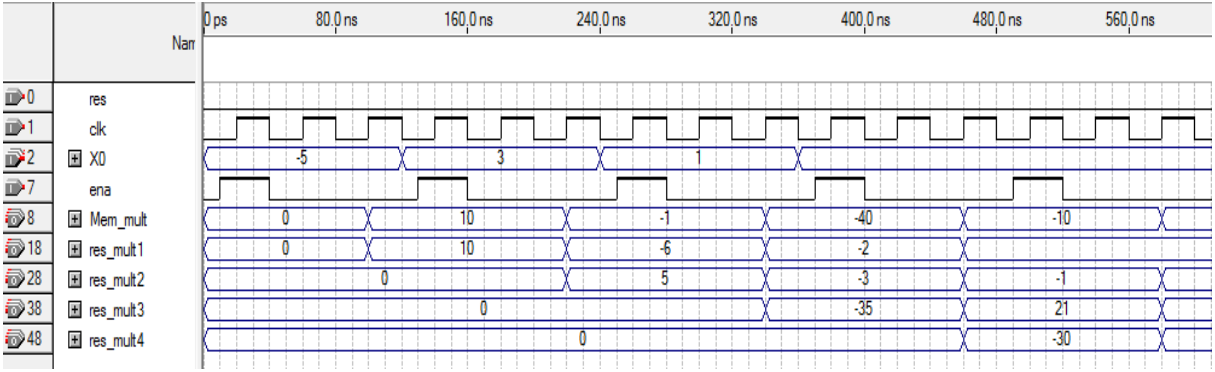


Рис. 2.32. Временные диаграммы работы фильтра на четыре отвода с использованием мегафункции ALTMEMMULT (коэффициенты фильтра загружаются из блочной памяти ПЛИС)

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 так же позволяет построить параллельный КИХ-фильтр с использованием всего лишь одного блока со встроенными перемножителями и сумматорами выполняющего функцию умножения и сложения. Использование внешней линии задержки из трех регистров приводит к незначительному увеличению ресурсов и не сказывается на быстродействие. 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. Использование же блочной памяти (рис. 2.22) 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.

2.4. Проектирование последовательных КИХ-фильтров

Рассмотрим проектирование последовательных КИХ-фильтров в САПР ПЛИС Quartus II. 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, управляющий автомат разработаны на языке Verilog, умножитель с применением мегафункции lpm_mult а аккумулятор представляет из себя иерархическое описание на языке verilog, нижний уровень которого представляет собой verilog-файл сгенерированный мегафункцией LPM_ADD_SUB. Умножитель 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. 2.33 показан иерархический проект последовательного КИХ-фильтра на четыре отвода в САПР Quartus II версии 13.0, а на рис. 2.34 структурная схема проекта представленная на уровне регистровых передач, полученная с помощью RTL-viewer. Входной сигнал и коэффициенты фильтра представлены в дополнительном коде с 8- и 4-битной точностью, поэтому умножитель и аккумулятор так же 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. 2.35 показана структурная схема линии задержки на базе регистров. Пример 2 демонстрирует verilog-код ПЗУ для хранения коэффициентов фильтра, а на рис. 2.36 показана структурная схема ПЗУ на базе дешифратора. Пример 3 демонстрирует verilog-код управляющего автомата КИХ-фильтра. На рис. 2.37 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, HOT), т.е. в каждый конкретный момент времени активным (hot) может быть только один триггер состояния.

Метод ОНЕ 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. 2.38 показана структурная схема умножителя размерностью операндов 8x4 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 9x9 – 46 шт.

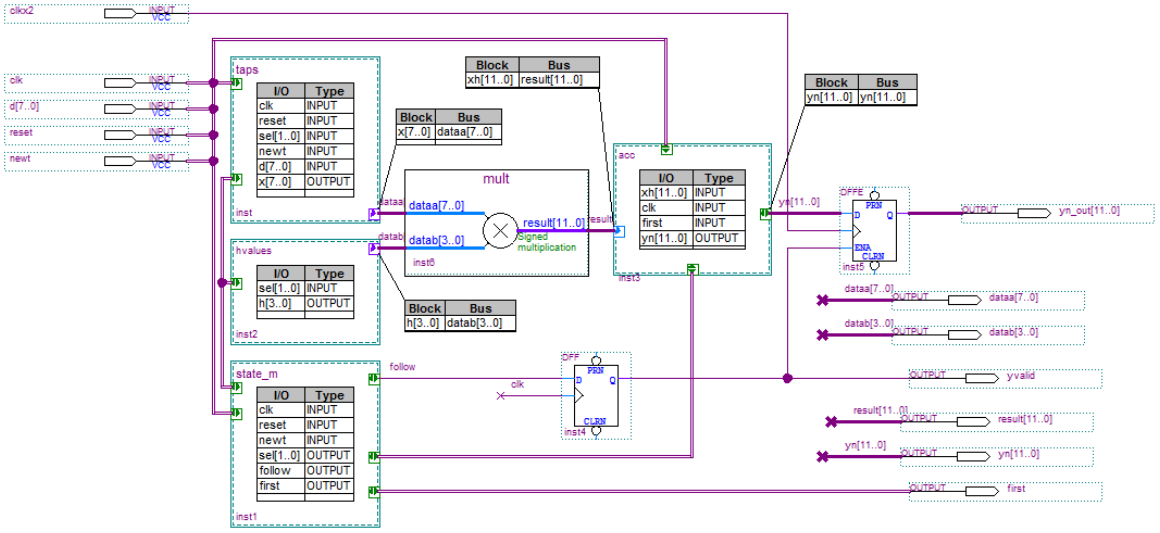


Рис. 2.33. Иерархический проект последовательного КИХ-фильтра на четыре отвода

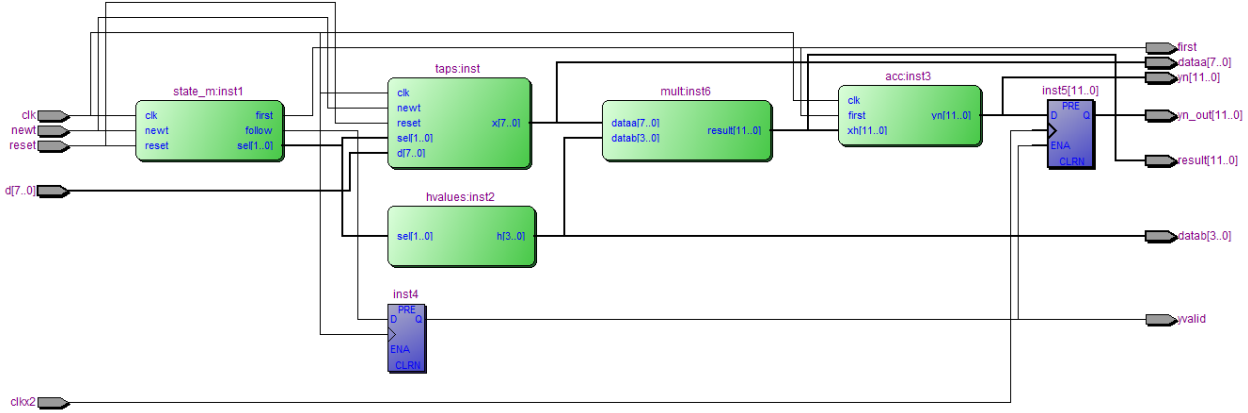


Рис. 2.34. Структурная схема проекта, полученная с помощью RTL-viewer

```
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

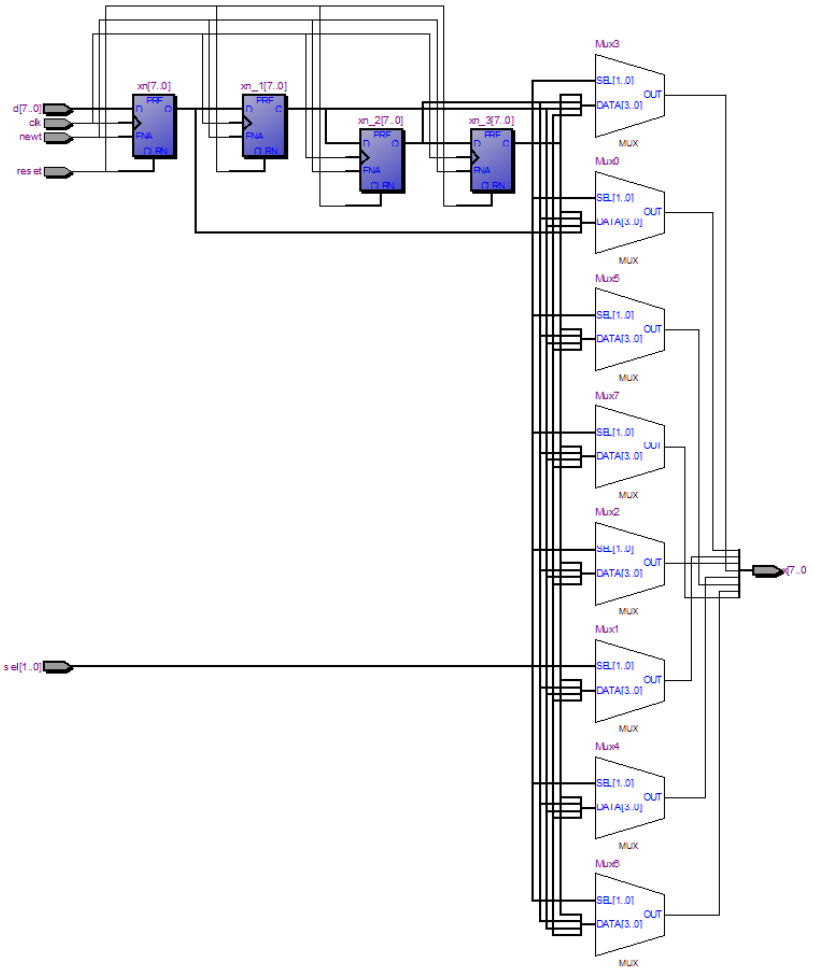


Рис. 2.35. Структурная схема линии задержки КИХ-фильтра

```
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

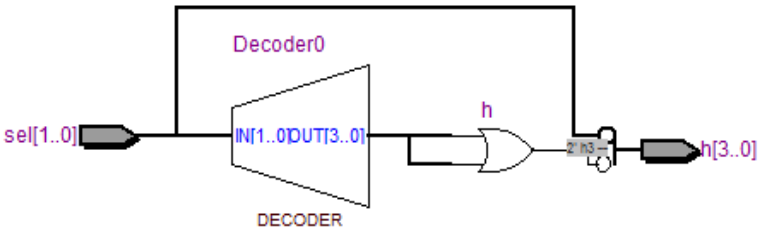


Рис. 2.36. Структурная схема ПЗУ

```
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;
```

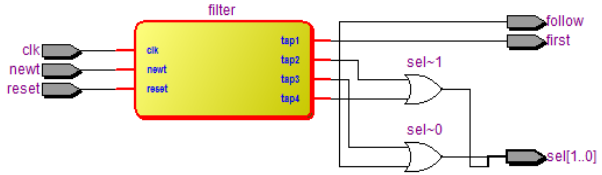
```
                        follow = 1'b0;
                        first = 1'b0;
                end
        tap1: begin
                        sel = 2'b0;
                        follow = 1'b0;
                        first = 1'b1;
                end
        tap2: begin
                        sel = 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
```

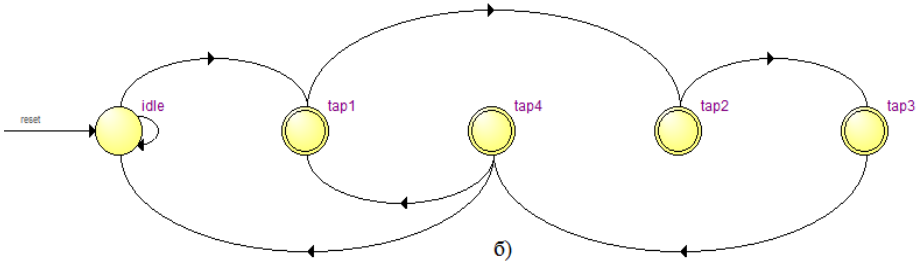
Пример 3. Verilog-код управляющего автомата КИХ-фильтра



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

Рис. 2.38. Структурная схема умножителя размерностью операндов 8x4 на базе мегафункции LPM_MULT

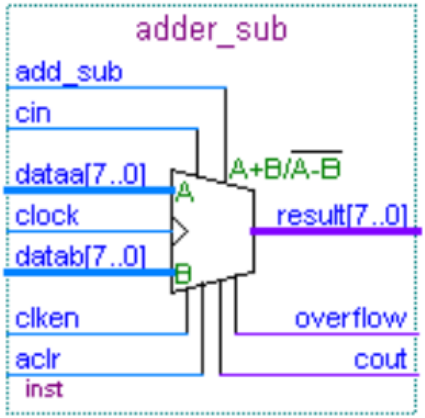


Рис. 2.39. Сумматор/вычитатель на базе мегафункции LPM_ADD_SUB

На рис. 2.39 показан сумматор/вычитатель на базе мегафункции LPM_ADD_SUB. Сумматор/вычитатель способен работать как в асинхронном, так и в синхронном режимах. 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. 2.40 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.

На рис. 2.41 и 2.42 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

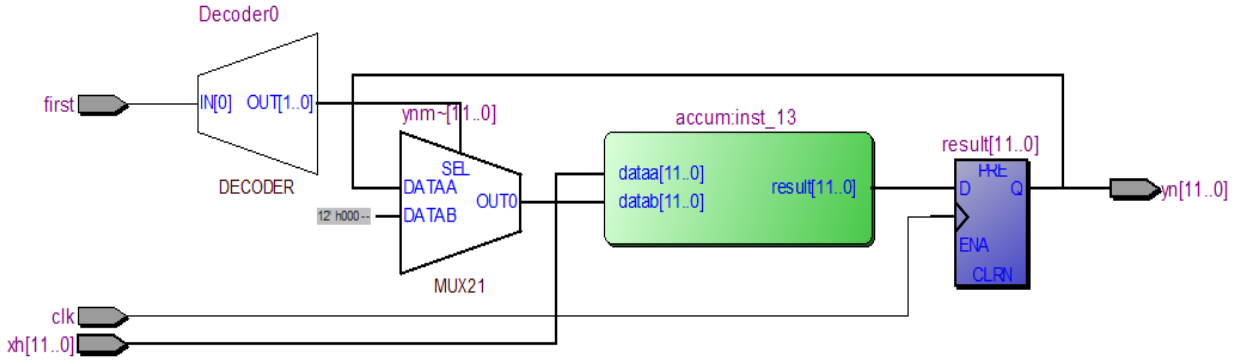


Рис. 2.40 Структурная схема аккумулятора на мегафункции LPM_ADD_SUB

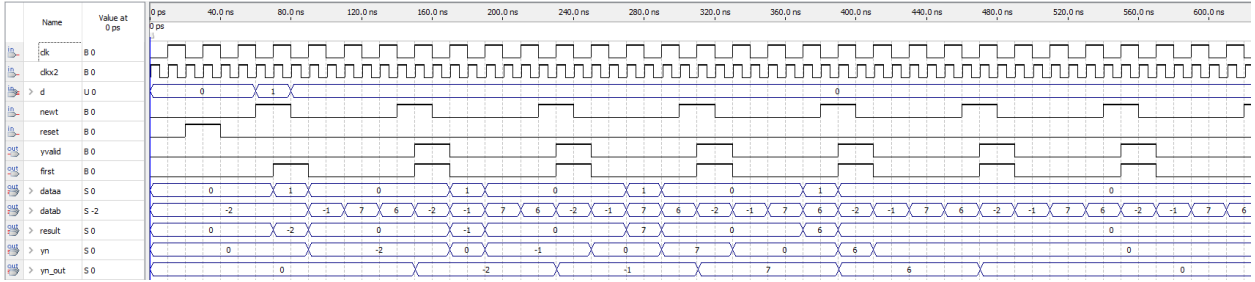


Рис. 2.41. Функциональное моделирование импульсной характеристики КИХ-фильтра на четыре отвода

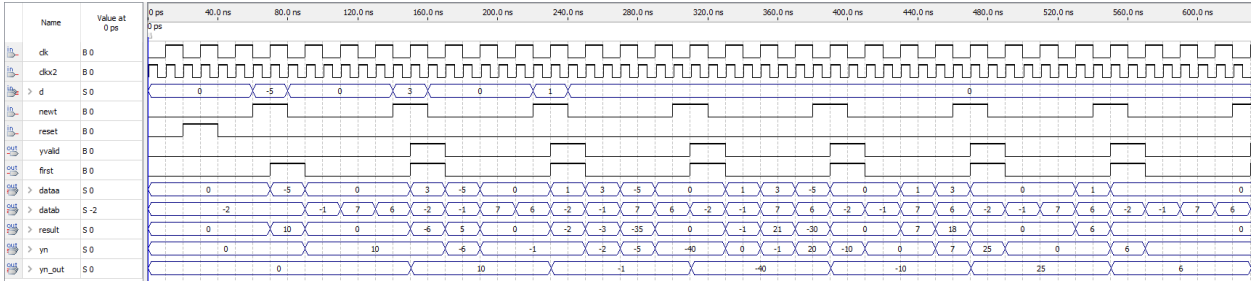


Рис. 2.42. Функциональное моделирование прохождения сигнала по структуре КИХ-фильтра на четыре отвода. На вход фильтра поступает сигнал -5, 3, 1, 0, 0 и 0 т.д. Правильные значения на выходе фильтра: 10, -1, -40, -10, 25, 6 и 0 т.д.

Переработаем проект (рис.2.33) с использованием нескольких вариантов. Рассмотрим вариант, когда умножитель и аккумулятор синхронизуются общим синхросигналом clk но реализуются с использованием мегафункции умножения LPM_MULT и накопления ALTACCUMULATE. Следует заметить, что мегафункция ALTACCUMULATE не входит в разработчик мегафункций (MegaWizard Plug-In Manager) для САПР Quartus II версии 13.1 и старше. Реализуется на базе логических элементов для ПЛИС серии Cyclone III и на базе адаптивных логических элементов для ПЛИС серии Stratix III.

Входной сигнал и коэффициенты фильтра представим с 8-разрядной точностью. Все функциональные блоки проекта на верхнем уровне иерархии представим символами. Линия задержки, ПЗУ и управляющий автомат представлены verilog-кодом и корректировки не подлежат. На рис.2.43 показан проект последовательного КИХ-фильтра на четыре отвода с использованием аккумулятора на мегафункции ALTACCUMULATE а на рис.2.44 демонстрируется функциональное моделирование прохождения сигнала по структуре КИХ-фильтра. Латентность умножителя и аккумулятора составляют 1 такт синхрочастоты.

Для согласования работы управляющего автомата и аккумулятора необходимо выходной сигнал first автомата задержать на один такт синхрочастоты с помощью двухтактного триггера выход которого необходимо подключить ко входу сигнала синхронной загрузки аккумулятора sload.

Результат умножения и вычисления суммы произведений представляются с 16-разрядной точностью. Для получения правильных значений профильтрованного сигнала на выходе yn_out[15..0] необходимо выходной сигнал follow задержать на три такта синхрочастоты и подключить его ко входу разрешения тактирования ena регистра результата.

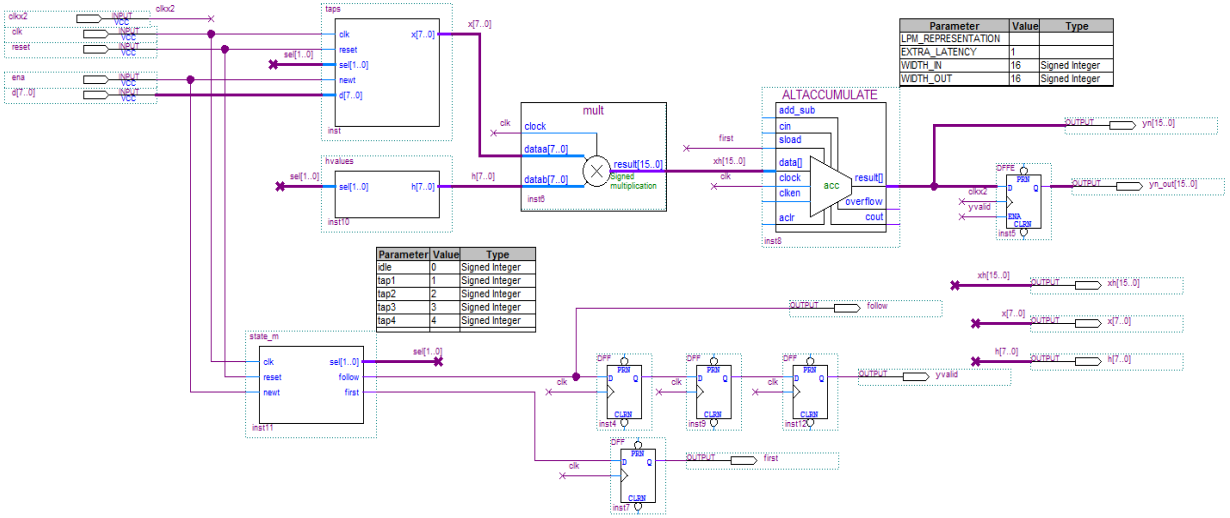


Рис. 2.43. Проект последовательного КИХ-фильтра на четыре отвода с использованием аккумулятора на мегафункции ALTACCUMULATE

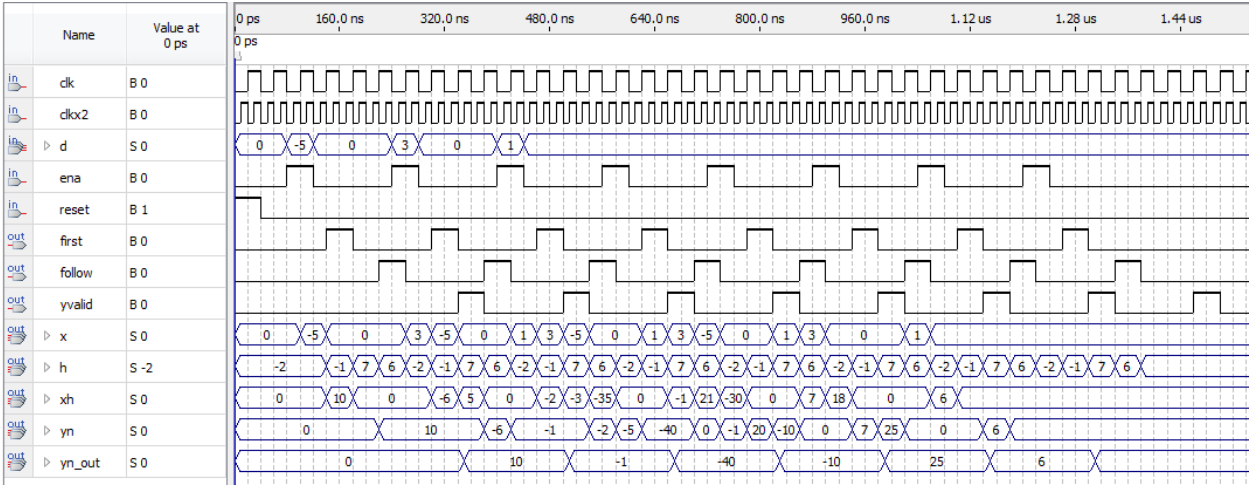


Рис. 2.44. Функциональное моделирование прохождения сигнала по структуре КИХ-фильтра на четыре отвода в случае использования аккумулятора на мегафункции ALTACCUMULATE

Рассмотрим вариант, когда умножитель и аккумулятор реализованы на мегафункции умножения и накопления ALTMULT_ACCUM (рис. 2.45 и рис. 2.46). Данный вариант является более предпочтительным для современных ПЛИС, так как предполагает использование встроенных ЦОС-блоков ПЛИС позволяя экономить логические ресурсы. Выходной сигнал автомата first необходимо подключить ко входу синхронной загрузки аккумулятора accum_sload на прямую. Последовательный КИХ-фильтр на мегафункции ALTMULT_ACCUM имеет латентность пять тактов синхрочастоты сигнала clk и результат фильтрации обновляется через каждый четвертый такт синхрочастоты (на пятый, рис. 2.46).

В данном разделе показаны проекты последовательных КИХ-фильтров на четыре отвода с применением различных инструментов проектирования САПР Quartus II, таких как цифровые автоматы, мегафункции, verilog-коды из описаний мегафункций, символьное и блочное представление функциональных блоков в графическом редакторе.

Рассмотрен проект КИХ-фильтра на умножителе (мегафункция LPM_MULT) и аккумуляторе (накапливающий сумматор на мегафункции LPM_ADD_SUB) работающие в асинхронных режимах, а также два проекта, когда умножитель и аккумулятор работают в синхронных режимах.

В первом варианте умножитель и аккумулятор реализованы на мегафункциях LPM_MULT и ALTACCUMULATE. Во втором варианте умножитель и аккумулятор реализованы на одной мегафункции умножения и накопления ALTMULT_ACCUM. Наиболее экономичным и простым в реализации оказался проект с использованием мегафункции ALTMULT_ACCUM.

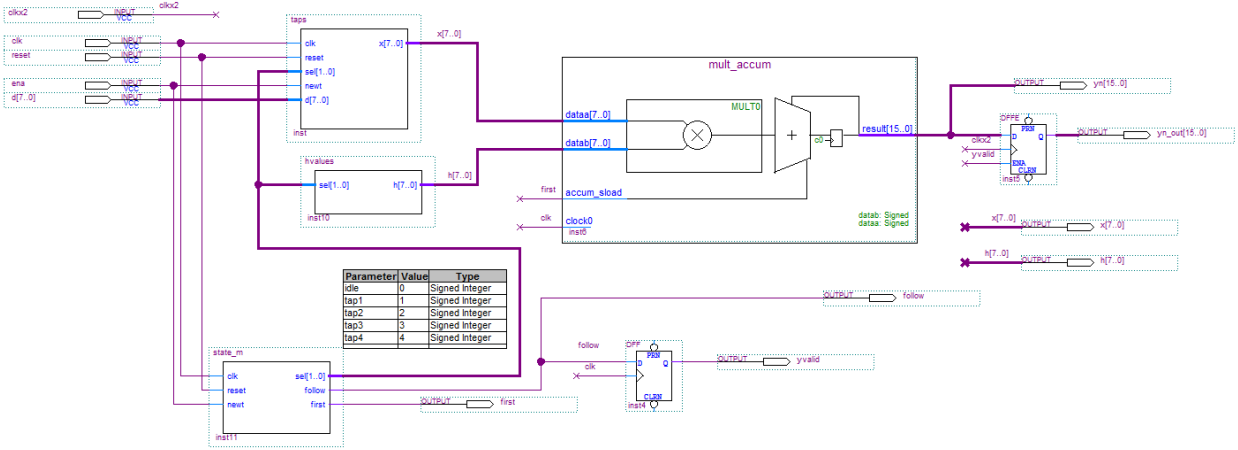


Рис. 2.45. Проект последовательного КИХ-фильтра на четыре отвода с использованием умножителя и аккумулятора на мегафункции ALTMULT_ACCUM

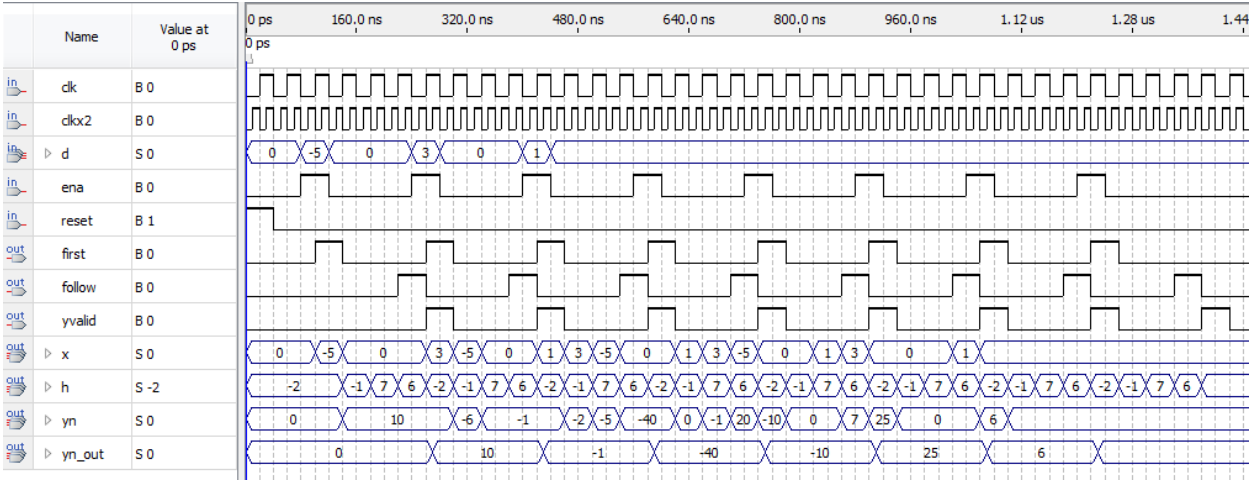


Рис. 2.46. Функциональное моделирование прохождения сигнала по структуре КИХ-фильтра на четыре отвода в случае использования умножителя и аккумулятора на мегафункции ALTMULT_ACCUM

3. ПРОЕКТИРОВАНИЕ КИХ-ФИЛЬТРОВ НА РАСПРЕДЕЛЕННОЙ АРИФМЕТИКЕ

3.1. 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.

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

$$y = \sum_{k=0}^{K-1} C_k \cdot x_k , \qquad (3.1)$$

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. 3.1 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. 3.2 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. 3.3 показаны 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. Рис. 3.4 показывает умножение десятичного числа 11 на 5 методом правого сдвига с накоплением по рекуррентной формуле, показанной на рис. 3.3.

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. 3.5).

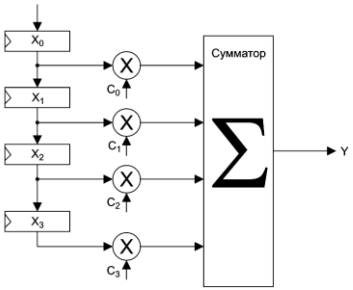


Рис. 3.1. Прямая реализация КИХ-фильтра на четыре отвода

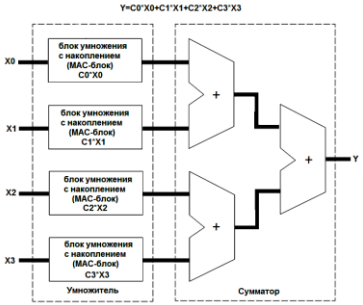


Рис. 3.2. Параллельный алгоритм реализации уравнения КИХ-фильтра на четыре отвода с использованием четырех блоков умножения с накоплением

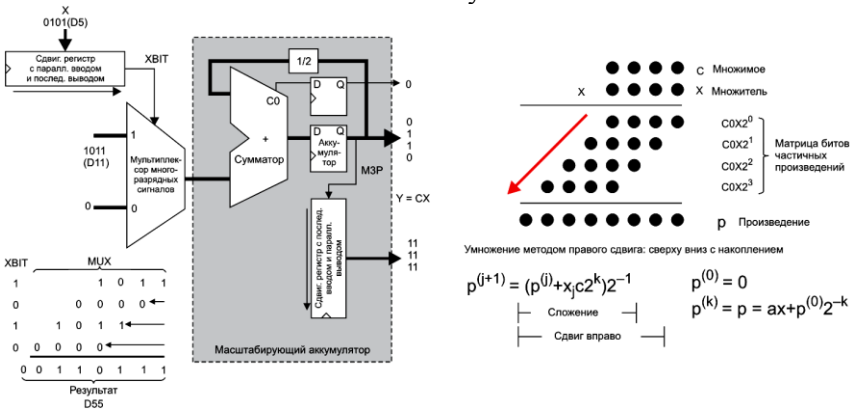


Рис. 3.3. Блок умножения с накоплением и алгоритм реализации умножения методом сдвига и сложения с накоплением

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

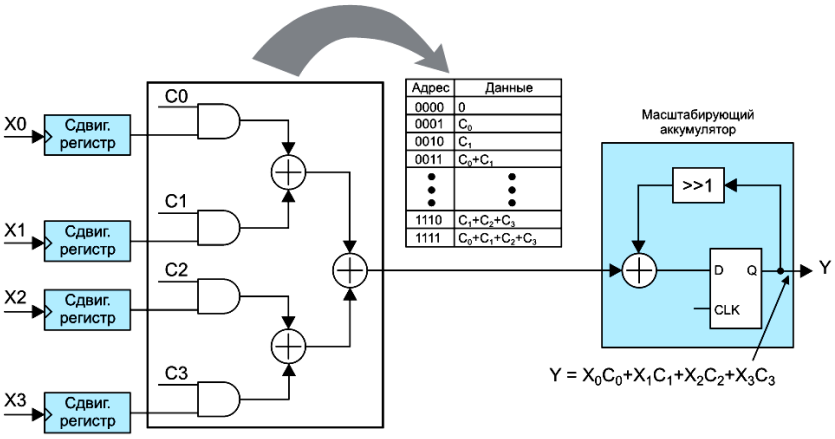


Рис. 3.5. Идея использования распределенной арифметики на примере КИХ-фильтра на четыре отвода

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

$$x_k = -x_{k(\text{B}-1)} 2^{B-1} + \sum_{b=0}^{B-2} x_{kb} \cdot 2^b , \qquad (3.2)$$

где B – разрядность кода. Подставим выражение (3.2) в (3.1), получим:

$$\begin{aligned} y &= \sum_{k=0}^{K-1} C_k \cdot \left[- x_{k(\text{B}-1)} 2^{B-1} + \sum_{b=0}^{B-2} x_{kb} \cdot 2^b \right] = \\ &= -\sum_{k=0}^{K-1} x_{k(B-1)} 2^{B-1} C_k + \sum_{k=0}^{K-1} \sum_{b=0}^{B-2} x_{kb} 2^b C_k \end{aligned} . \qquad (3.3)$$

Раскроем все суммы в выражении (3.3) и сгруппируем числа по степеням B:

$$\begin{aligned} y &= \left[x_{00} \cdot C_0 + x_{10} \cdot C_1 + x_{20} \cdot C_2 + ... + x_{(K\text{-}1)0} \cdot C_{K-1} \right] \\ &+ \left[x_{01} \cdot C_0 + x_{11} \cdot C_1 + x_{21} \cdot C_2 + ... + x_{(K-1)1} \cdot C_{K-1} \right] \cdot 2^1 \\ &+ \left[x_{02} \cdot C_0 + x_{12} \cdot C_1 + x_{22} \cdot C_2 + ... + x_{(K-1)2} \cdot C_{K-1} \right] \cdot 2^2 \\ &.... \\ &.... \\ &+ \left[x_{0(B-2)} \cdot C_0 + x_{1(B-2)} \cdot C_1 + x_{2(B-2)} \cdot C_2 + ... + x_{(K-1)(B-2)} \cdot C_{K-1} \right] \cdot 2^{B-2} \\ &- \left[x_{0(B-1)} \cdot C_0 + x_{1(B-1)} \cdot C_1 + x_{2(B-1)} \cdot C_2 + ... + x_{(K-1)(B-1)} \cdot C_{K-1} \right] \cdot 2^{B-1} \end{aligned} \qquad (3.4)$$

Выражение (3.4) для КИХ-фильтра на четыре отвода ($K = 4$), в котором входная переменная $x_k(n)$ 4-разрядная ($B = 4$), запишем в виде:

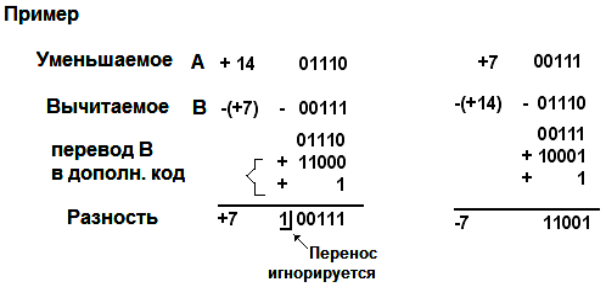
$$y = P_0 + P_1 \cdot 2^1 + P_2 \cdot 2^2 - P_3 \cdot 2^3 .$$

$$\begin{aligned} P_0 &= X_{00}C_0 + X_{10}C_1 + X_{20}C_2 + X_{30}C_3 \\ P_1 &= X_{01}C_0 + X_{11}C_1 + X_{21}C_2 + X_{31}C_3 \\ P_2 &= X_{02}C_0 + X_{12}C_1 + X_{22}C_2 + X_{32}C_3 \\ P_3 &= X_{03}C_0 + X_{13}C_1 + X_{23}C_2 + X_{33}C_3 \end{aligned}, \qquad (3.5)$$

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. 3.6). Одна четырех входовая LUT-таблица обеспечивает 16 частичных произведений, которые являются комбинациями сумм коэффициентов КИХ-фильтров.

Еще раз обратим внимание на то, что 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.3.6).



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

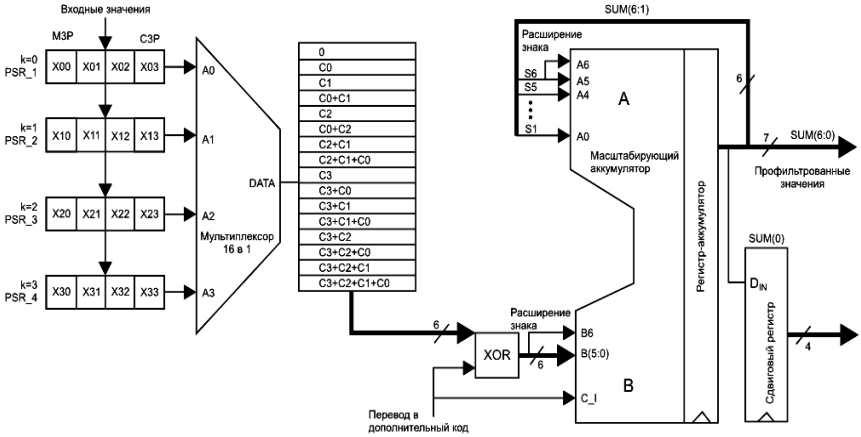


Рис. 3.6. Упрощенная схема КИХ-фильтра на распределенной арифметике

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 можно было воспользоваться инструментом FDATool (Filter Design & Analysis Tool) системы Matlab/Simulink.

В момент времени $n = 0$ на вход КИХ-фильтра подается входная переменная $x_0(n)$ (отсчет, например, число десятичное число равное -5, представленное в дополнительном четырехзначном двоичном коде как 1011), которое 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. 3.7). Из LUT-таблицы извлекается коэффициент C_0, представленный в дополнительном коде 111110 с расширением знака на два разряда и поступает в масштабирующий аккумулятор, где происходит его сложение с нулем. Операция расширения знака для чисел, представленных в дополнительном коде показана на рис. 3.6.

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. 3.8) 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. 3.9). Четвертый цикл обработки

заканчивается вычитанием всех битов знакового разряда всех k входных переменных LUT-таблицы (рис. 3.10).

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. 3.11.

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 $f = x_0 C_0 + x_1 C_1 + x_2 C_2 + x_3 C_3$.

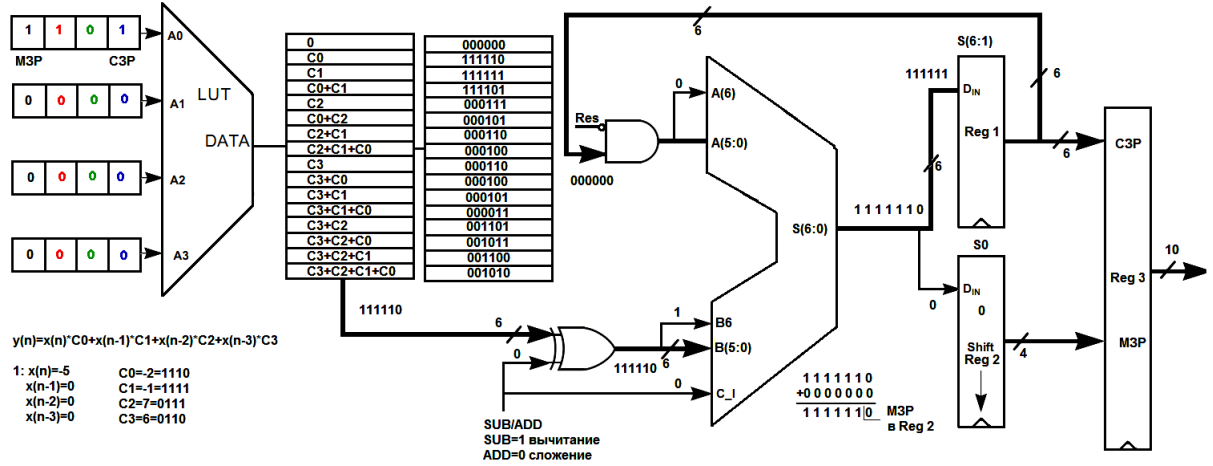


Рис. 3.7. На вход КИХ-фильтра подается число десятичное число равное -5, представленное в дополнительном четырехзначном двоичном коде как 1011, первый цикл обработки (адресация 0001). Суммирование частичного произведения P_0 с весом 2^0 с 0

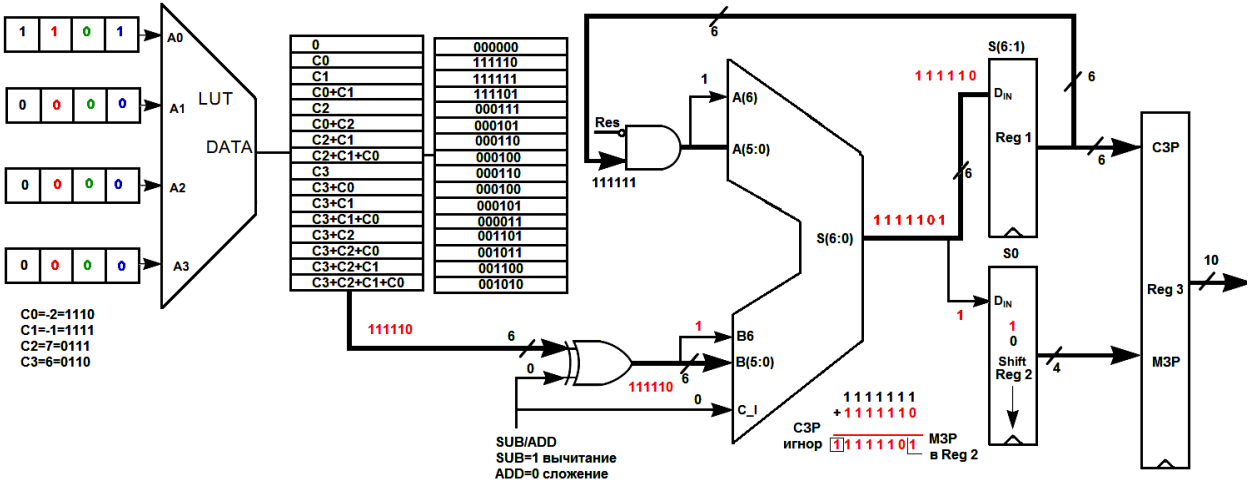


Рис. 3.8. Второй цикл обработки (адресация 0001). Суммирование частичного произведения P_1 с весом 2^1 с частичным произведением P_0 с весом 2^0

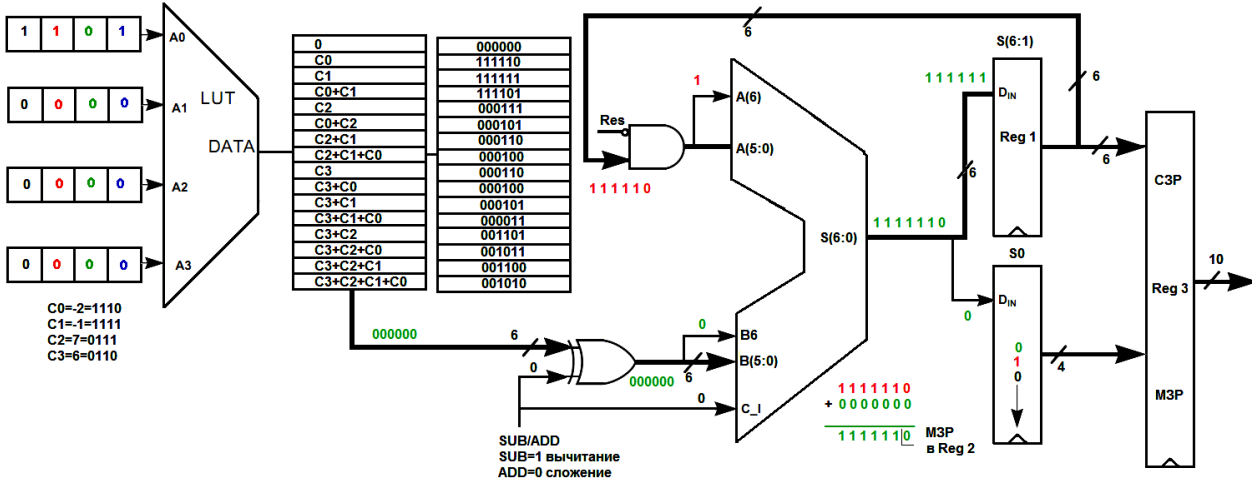


Рис. 3.9. Третий цикл обработки (адресация 0000). Суммирование частичного произведения P_2 с весом 2^2 с суммой частичных произведений P_0 с весом 2^0 и P_1 с весом 2^1

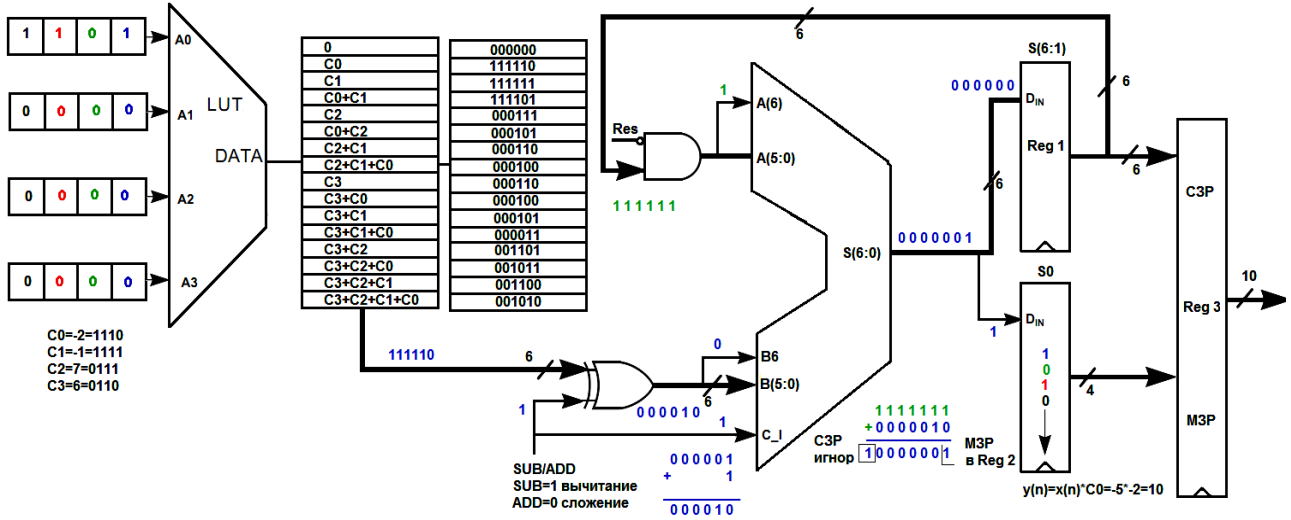


Рис. 3.10. Четвертый цикл обработки (адресация 0001). Суммирование (вычитание) частичного произведения P_3 с весом -2^3, представленного в дополнительном коде с суммой частичных произведений P_2 с весом 2^2, P_1 с весом 2^1 и P_0 с весом 2^0

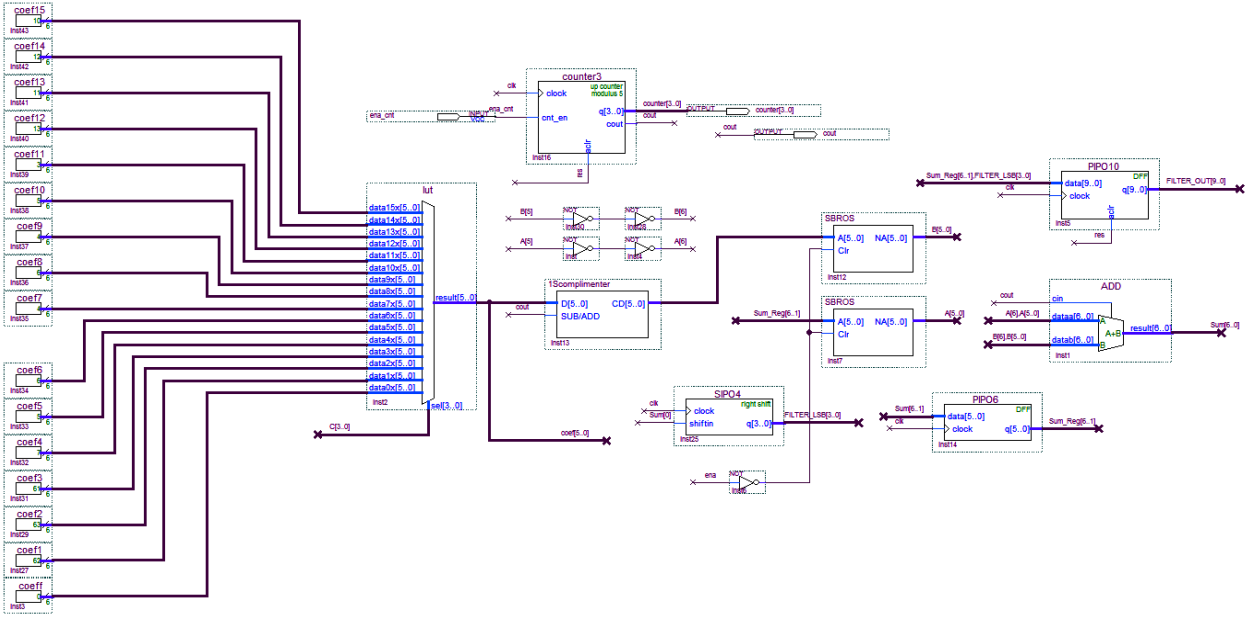


Рис. 3.11. Фрагмент схемы КИХ-фильтра на четыре отвода. Показаны мультиплексор 16 в 1 для хранения комбинации сумм коэффициентов, блок вычисления обратного кода, два блока очистки данных на входах сумматора, счетчик с модулем счета 5 и вспомогательные регистры

На адресные входы мультиплексора поступают биты младшего значащего разряда всех k входных переменных LUT-таблицы. Перед началом работы регистры линии задержки (рис. 3.12) 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 так далее. 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. 3.13 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. 3.14 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. Выбираются 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. Задаются также входная и выходная спецификации фильтра (разрядность представления входных и выходных данных). На рис. 3.15 показана тестовая схема КИХ-фильтра с использованием мегаядра FIR Compiler, а на рис. 3.16 - временные диаграммы работы. Входные отсчеты -5, 3, 1, 0. Профильтрованные значения на выходе: 10, -1, -40, 25.

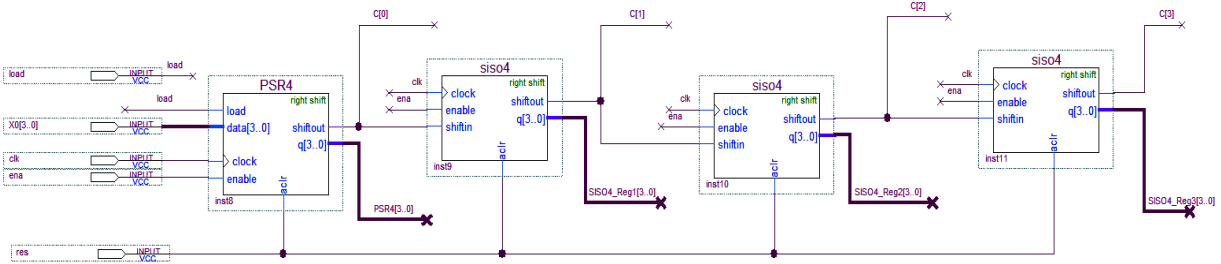


Рис.3.12. Фрагмент схемы КИХ-фильтра. Линия задержки

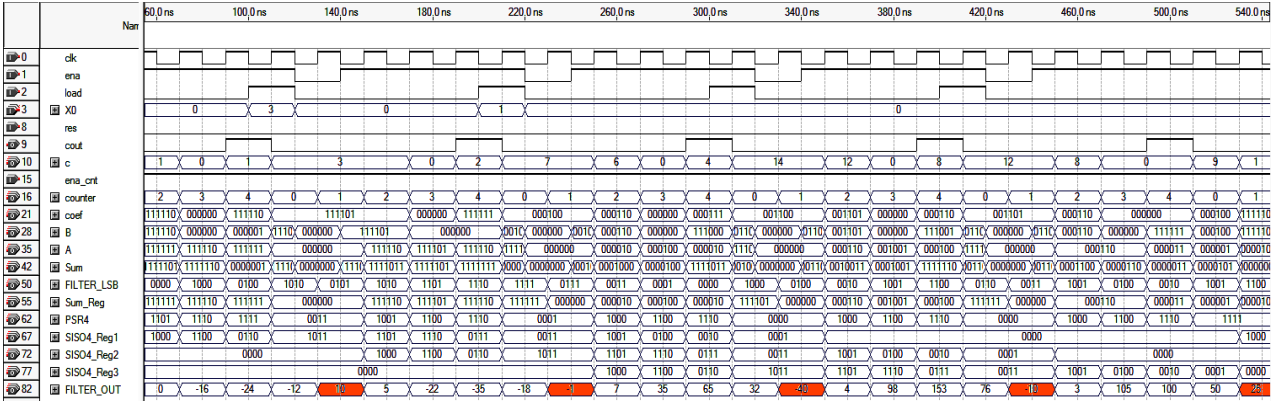


Рис. 3.13. Временные диаграммы работы КИХ-фильтра на распределенной арифметике

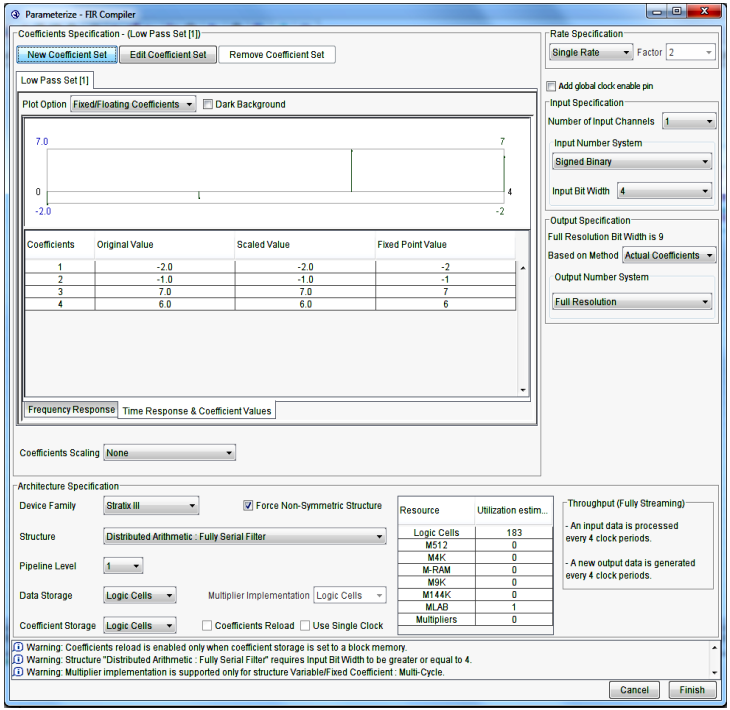


Рис. 3.14. Настройки мегаядра FIR Compiler САПР ПЛИС Quartus II

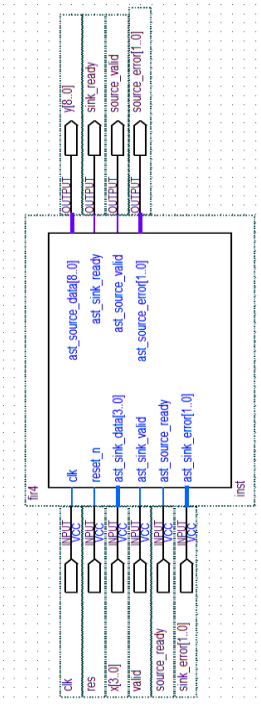


Рис. 3.15.Тестовая схема КИХ-фильтра с использованием мегаядра FIR Compiler

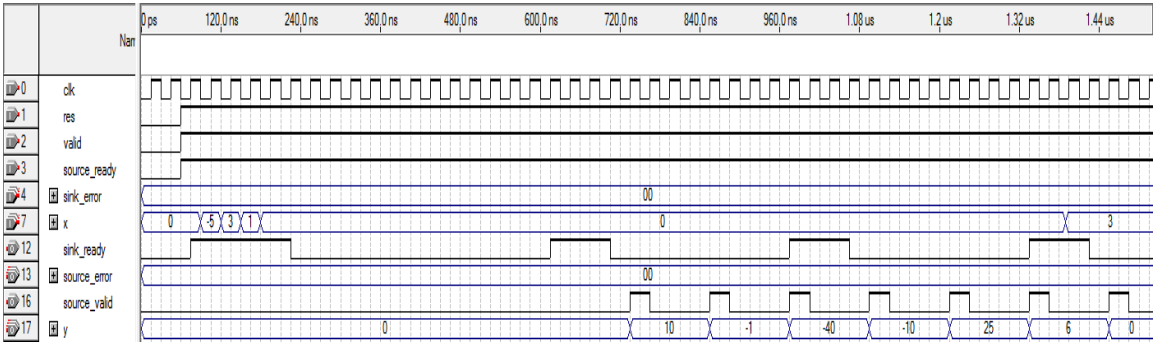


Рис. 3.16. Временные диаграммы работы КИХ-фильтра на мегаядре FIR Compiler

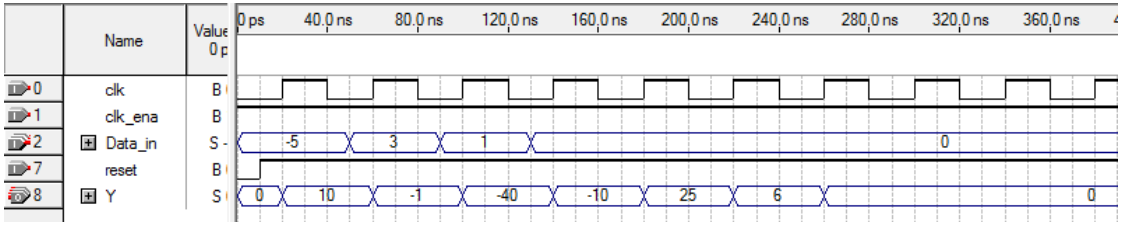


Рис. 3.17. Временные диаграммы работы КИХ-фильтра на четыре отвода по коду языка VHDL (пример 2)

Разработаем код высокоуровневого языка описания аппаратных средств VHDL КИХ-фильтра (пример 1 и пример 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. 3.17). Сравнивая временные диаграммы (рис. 3.13 и рис. 3.16 с рис. 3.17), 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 подразделе на примере проектирования простейшего КИХ-фильтра на четыре отвода показаны основы распределенной арифметики, широко используемой для разработки высокопроизводительных цифровых устройств цифровой обработки сигналов.

3.2. КИХ-фильтры на параллельной распределенной арифметике

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:

$$y = \sum_{k=1}^{K} C_k \cdot x_k , \qquad (3.6)$$

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

$$x_k = -x_{k(\mathrm{B}-1)} 2^{B-1} + \sum_{b=0}^{B-2} x_{kb} \cdot 2^b , \qquad (3.7)$$

где $B-$разрядность кода; $x_{k(\mathrm{B}-1)} 2^{B-1} -$знаковый разряд. Подставим выражение (3.7) в (3.6), получим:

$$\begin{aligned} y &= \sum_{k=1}^{K} C_k \cdot \left[- x_{k(\mathrm{B}-1)} 2^{B-1} + \sum_{b=0}^{B-2} x_{kb} \cdot 2^b \right] = \\ &= -\sum_{k=1}^{K} x_{k(B-1)} 2^{B-1} C_k + \sum_{k=1}^{K} \sum_{b=0}^{B-2} x_{kb} 2^b C_k \end{aligned} \qquad (3.8)$$

Раскроем все суммы в выражении (3.8) и сгруппируем числа по степеням B:

$$
\begin{aligned}
y = &\left[x_{10}\cdot C_1 + x_{20}\cdot C_2 + x_{30}\cdot C_3 + ... + x_{K0}\cdot C_K\right]\cdot 2^0 \\
&+\left[x_{11}\cdot C_1 + x_{21}\cdot C_2 + x_{31}\cdot C_3 + ... + x_{K1}\cdot C_K\right]\cdot 2^1 \\
&+\left[x_{12}\cdot C_1 + x_{22}\cdot C_2 + x_{32}\cdot C_3 + ... + x_{K2}\cdot C_K\right]\cdot 2^2 \\
&.... \\
&.... \\
&+\left[x_{1(B-2)}\cdot C_1 + x_{2(B-2)}\cdot C_2 + x_{3(B-2)}\cdot C_3 + ... + x_{K(B-2)}\cdot C_K\right]\cdot 2^{B-2} \\
&-\left[x_{1(B-1)}\cdot C_1 + x_{2(B-1)}\cdot C_2 + x_{3(B-1)}\cdot C_3 + ... + x_{K(B-1)}\cdot C_K\right]\cdot 2^{B-1}
\end{aligned}
\qquad (3.9)
$$

Каждое выражение в квадратной скобке 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 квадратной скобке для индекса b может быть записано в виде

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

и для фильтра с симметричными коэффициентами:

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

Рассмотрим построение КИХ-фильтра на основе параллельной распределённой арифметики на примере структуры восемь отводов восемь бит. Перепишем уравнение (3.9) в следующем виде:

$$
\begin{aligned}
y = &\,[sum0] + [sum1]2^1 + [sum2]\cdot 2^2 + ... + \\
&+ [sum6]\cdot 2^6 - [sum7]\cdot 2^7
\end{aligned}
\qquad (3.10)
$$

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 (3.10) в массив подобных сумм на основе двухвходовых сумматоров:

$$y = \{[sum0]+[sum1]2^1\}+\{[sum2]+[sum3]2^1\}2^2+\{[sum4]+[sum5]2^1\}2^4 + \{[sum6]-[sum7]2^1\}2^6 \quad (3.11)$$

или

$$y = \{\{[sum0]+[sum1]2^1\}+\{[sum2]+[sum3]2^1\}2^2\}+ \{\{[sum4]+[sum5]2^1\}+\{[sum6]-[sum7]2^1\}2^2\}2^4 \quad (3.12)$$

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.

Выше приведенные уравнения (3.11) и (3.12) 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

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

$$y(n) = \sum_{k=1}^{K} C_k \cdot \left[-x_{k0} + \sum_{b=1}^{B-1} x_{kb} \cdot 2^{-b} \right] = \\ = -\sum_{k=1}^{K} x_{k0} \cdot C_k + \sum_{k=1}^{K} \sum_{b=1}^{B-1} x_{kb} \cdot C_k \cdot 2^{-b} \quad . \tag{3.13}$$

Аналогично выражению (3.9) уравнение КИХ-фильтра для формата с фиксированной запятой будет иметь вид

$$\begin{aligned} y(n) = &-\left[x_{10} \cdot C_1 + x_{20} \cdot C_2 + x_{30} \cdot C_3 + ... + x_{K0} \cdot C_K\right] \cdot 2^0 + \\ &+ \left[x_{11} \cdot C_1 + x_{21} \cdot C_2 + x_{31} \cdot C_3 + ... + x_{K1} \cdot C_K\right] \cdot 2^{-1} + \\ &+ \left[x_{12} \cdot C_1 + x_{22} \cdot C_2 + x_{32} \cdot C_3 + ... + x_{K2} \cdot C_K\right] \cdot 2^{-2} + \\ &.... \\ &.... \\ &+ \left[x_{1(B-2)} \cdot C_1 + x_{2(B-2)} \cdot C_2 + x_{3(B-2)} \cdot C_3 + ... + x_{K(B-2)} \cdot C_K\right] \cdot 2^{-(B-2)} + \\ &+ \left[x_{1(B-1)} \cdot C_1 + x_{2(B-1)} \cdot C_2 + x_{3(B-1)} \cdot C_3 + ... + x_{K(B-1)} \cdot C_K\right] \cdot 2^{-(B-1)}. \end{aligned} \tag{3.14}$$

Переформулируем выражение (3.14) и представим его в следующем виде и сравним с уравнением (3.13):

$$y(n) = -\sum_{k=1}^{K} C_k x_{k0} + \sum_{b=1}^{B-1}\left[\sum_{k=1}^{K} C_k x_{kb}\right]2^{-b} = \\ = -P_0 + \left[\sum_{b=1}^{B-1} 2^{-b} P_b\right] \quad , \qquad (3.15)$$

где P – частичные произведения (значения в квадратных скобках выражения (3.14), 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. 3.18). По формуле (3.13) получается фильтр с использованием операций умножения с накоплением, а по формуле (3.15) - 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

$$y(n) = \left[\sum_{b=0}^{B-2} 2^b P_b\right] - 2^{B-1} P_{B-1} \qquad (3.16)$$

Дополнительный код, дробные значения

$$y(n) = -P_0 + \left[\sum_{b=1}^{B-1} 2^{-b} P_b\right] \qquad (3.17)$$

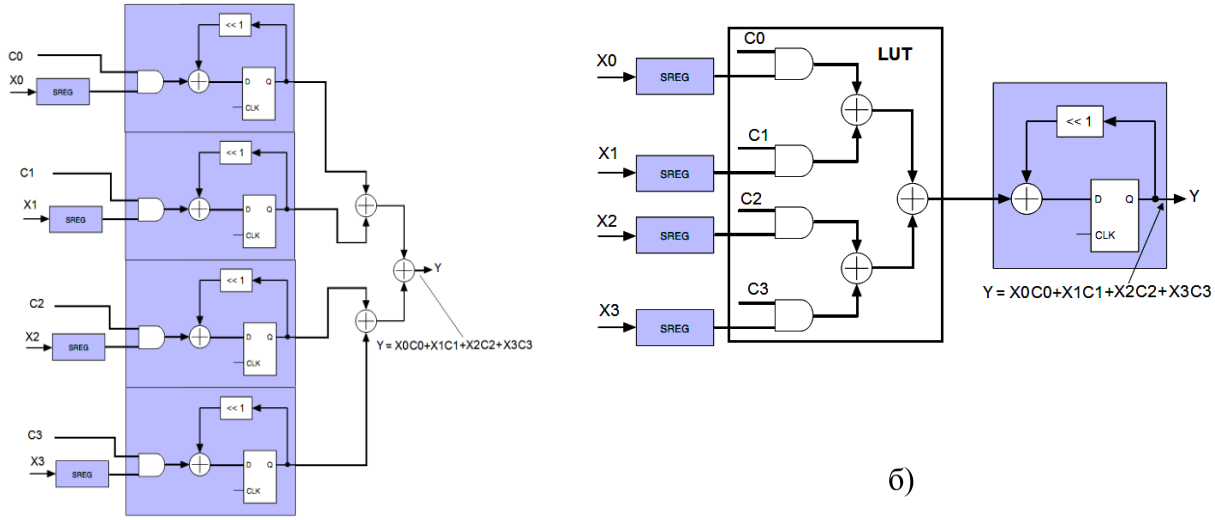


Рис. 3.18. КИХ-фильтр на четыре отвода: а) аппаратная реализация фильтра по формуле (3.13); б) по формуле (3.15) с использованием LUT

Рассмотрим КИХ-фильтр (рис. 3.19) с симметричными коэффициентами (выбираются симметрично относительно центральной величины). В основе стуктуры КИХ-фильтра лежит параллельный векторный перемножитель, в качестве которого из-за постоянства коэффициентов используют таблицу перекодировок (LUT). Рассмотрим простейший параллельный векторный перемножитель 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. 3.20). Умножение и сложение происходят параллельно с использованием LUT (табл. 3.1).

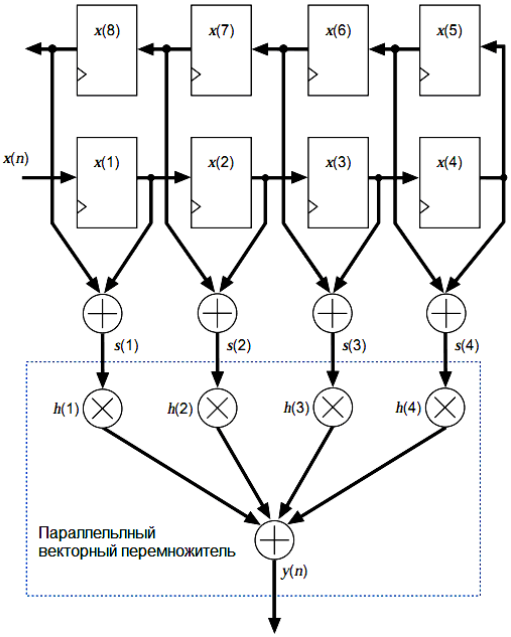


Рис. 3.19. Структура КИХ-фильтра на восемь отводов с симметричными коэффициентами, в основе которой лежит параллельный векторный перемножитель

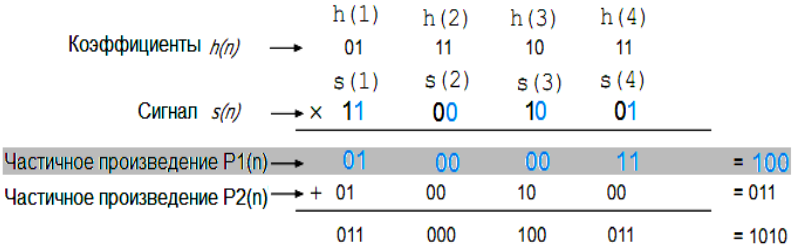


Рис. 3.20. Принцип параллельного векторного перемножения

Принцип формирования частичного произведения P1(n) показан на рис. 3.20. Булева функция $y = [s(1)h(1)] + [s(2)h(2)] + [s(3)h(3)] + [s(4)h(4)]$ для формирования P1(n) реализуется таблицей истинности, которая хранится в LUT. 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. 3.21 показан параллельный векторный перемножитель четырех 2-разрядных сигналов. Таким образом, требуются две идентичные таблицы, двоичный сдвиг влево и операция суммирования.

На рис. 3.22 показана структура КИХ-фильтра восемь отводов восемь бит на распределенной арифметике с несимметричными и с симметричными коэффициентами, обеспечивающими точность вычислений от 8 до 19 бит (полная точность), в основе которой лежит параллельный векторный перемножитель.

Таблица 3.1

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

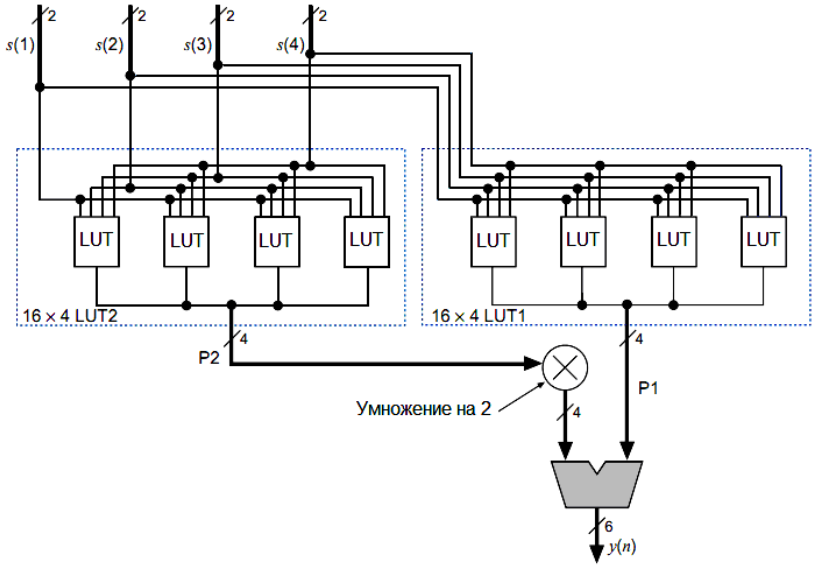
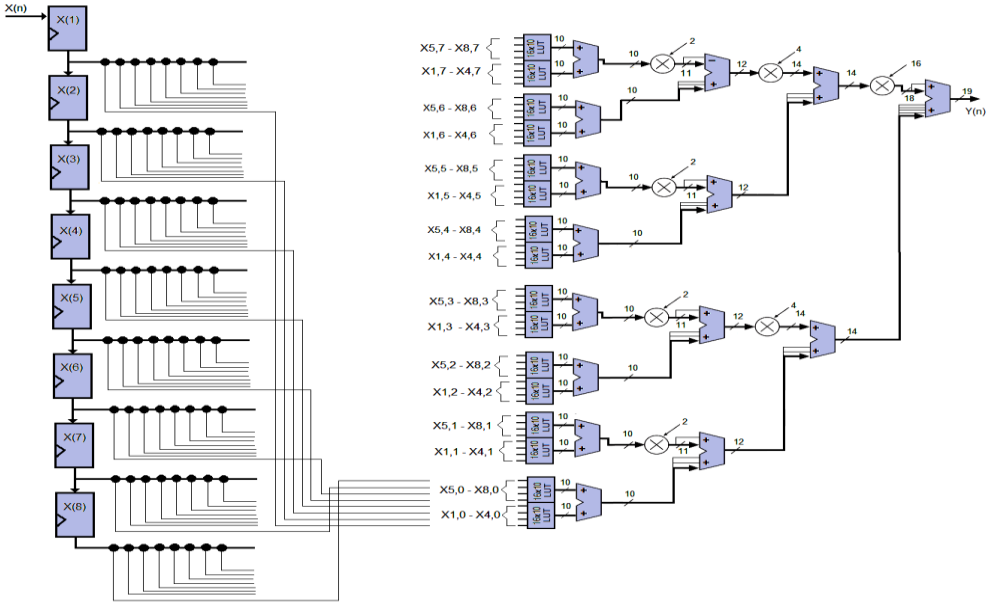
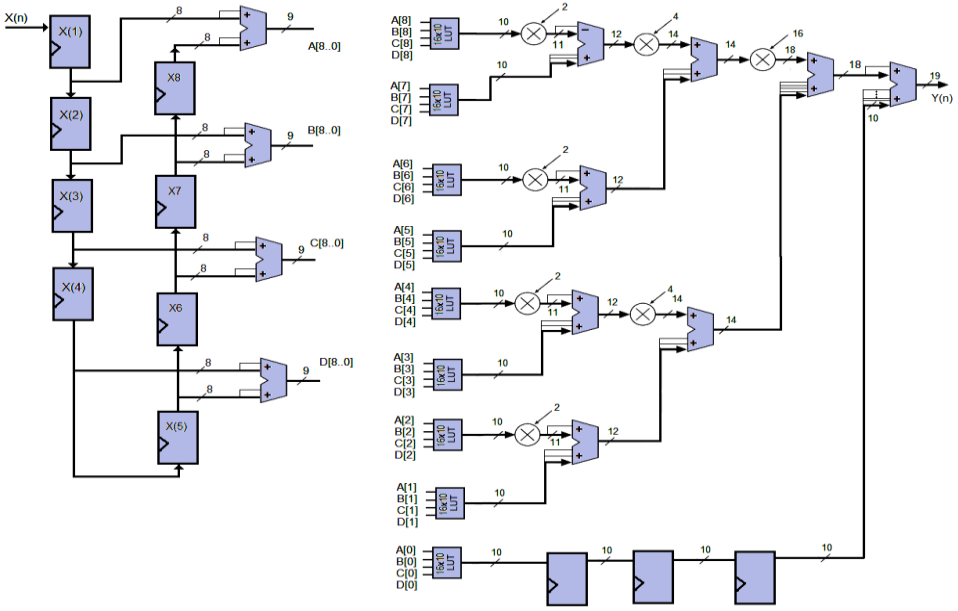


Рис. 3.21. Параллельный векторный перемножитель четырех 2-разрядных сигналов на четыре 2-разрядные константы



а)

Рис. 3.22. Структура КИХ-фильтра восемь отводов восемь бит на распределенной параллельной арифметике: а) с несимметричными коэффициентами; б) с симметричными



б)

Рис. 3.22. Структура КИХ-фильтра восемь отводов восемь бит на распределенной параллельной арифметике: а) с несимметричными коэффициентами; б) с симметричными (продолжение)

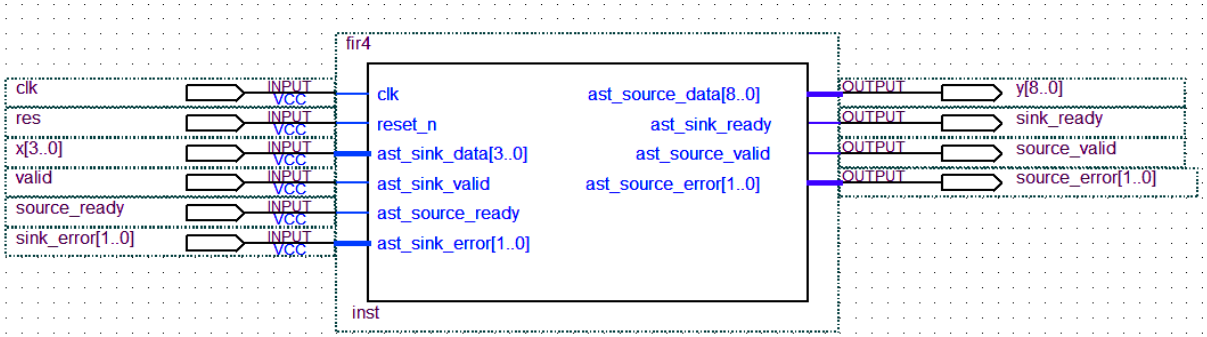


Рис. 3.23.Тестовая схема КИХ-фильтра с использованием мегаядра FIR Compiler на последовательной и параллельной распределенной арифметике

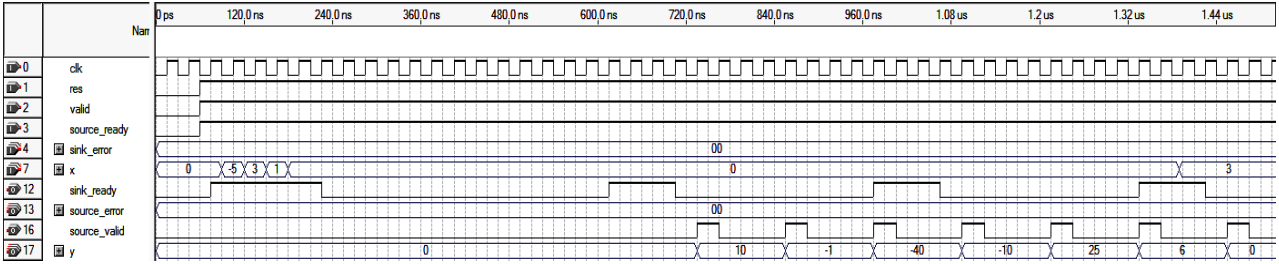


Рис. 3.24. Временные диаграммы работы КИХ-фильтра с использованием последовательной распределенной арифметики на мегаядре FIR Compiler

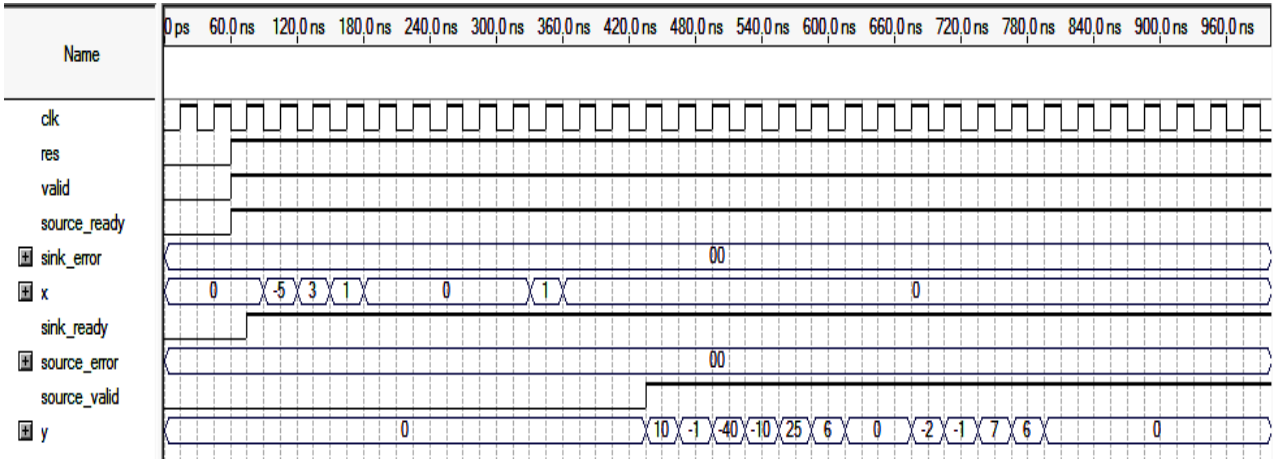


Рис. 3.25. Временные диаграммы работы КИХ-фильтра с использованием параллельной распределенной арифметики на мегаядре FIR Compiler

Входные данные на линии задержки представлены с 8-битной точностью параллельного кода. Для фильтра с симметричными коэффициентами требуются на выходах линии задержки 4 параллельных сумматора, которые своими выходами непосредственно адресуют 4-входовые LUT. Для того, чтобы переполнение 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 восьмью входами можно использовать 16 LUT с четырьмя входами).

Частичные произведения, представляющие комбинацию сумм 8-разрядных коэффициентов, хранящиеся в LUT представлены с 10-битной точность 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. 3.23 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. 3.24 и рис. 3.25 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. 3.2 показывает, что при числе отводов равным четырем существенной разницы между последовательной и параллельной арифметиками нет, т.к. для фильтра на последовательной арифметики 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 арифметики 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 3.2

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

3.3. Пример реализации КИХ-фильтра на параллельной распределенной арифметике

Уравнение КИХ-фильтра со структурой четыре отвода четыре 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 виде

$$P_{out} = 2^0 * P_0 + 2^1 * P_1 + 2^2 * P_2 - 2^3 * P_3 \qquad (3.18)$$

Частичные произведения P_0, P_1, P_2 и P_3:

$$P_0 = C_1 d_1(0) + C_2 d_2(0) + C_3 d_3(0) + C_4 d_4(0) = \sum_{n=1}^{4} C_n d_n(0); \qquad (3.19)$$

$$P_1 = C_1 d_1(1) + C_2 d_2(1) + C_3 d_3(1) + C_4 d_4(1) = \sum_{n=1}^{4} C_n d_n(1); \qquad (3.20)$$

$$P_2 = C_1 d_1(2) + C_2 d_2(2) + C_3 d_3(2) + C_4 d_4(2) = \sum_{n=1}^{4} C_n d_n(2); \qquad (3.21)$$

$$P_3 = C_1 d_1(3) + C_2 d_2(3) + C_3 d_3(3) + C_4 d_4(3) = \sum_{n=1}^{4} C_n d_n(3). \qquad (3.22)$$

В случае параллельной распределённой арифметики для КИХ-фильтра на четыре отвода необходимо иметь четыре идентичных массивов памяти, параллельно адресуемых всеми битами всех входных переменных и свертывающееся иерархического 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. 3.26 показана линия задержки КИХ-фильтра, а на рис. 3.27 - принцип подключения выходов линии задержки

КИХ-фильтра на четыре 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 представленные с 8-битной точностью).

Заполнение 4-входовой LUT показано в табл. 3.3, а описание на языке VHDL демонстрирует пример 3. На рис. 3.28 показана структура КИХ-фильтра на четыре отвода четыре 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 3.19-3.22. Для суммирований значений с выходов LUT в соответствии с их весом (формула 3.18) используются два 12- и один 14-разрядные сумматоры с соответствующей коррекцией разрядности на входах для того, чтобы гарантировано предотвратить переполнение.

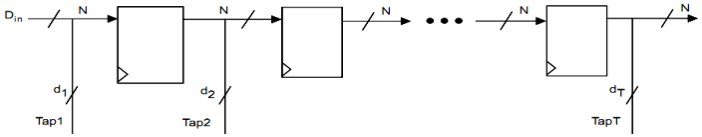


Рис. 3.26. Линия задержки КИХ-фильтра на регистрах

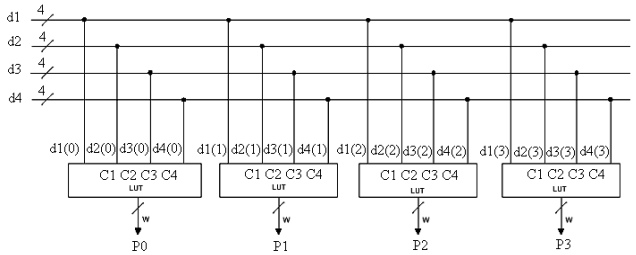


Рис. 3.27. Принцип подключения выходов линии задержки КИХ-фильтра на четыре отвода к 4-входовым LUT

Таблица 3.3

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 четыре отвода

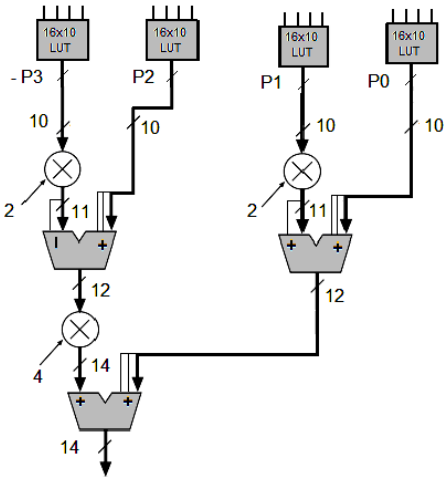


Рис. 3.28. Структура КИХ-фильтра на четыре отвода четыре бита на распределенной параллельной арифметике с указанием операции расширения знака

На рис. 3.29 показана функциональная модель КИХ-фильтра на четыре отвода четыре бита на распределенной параллельной арифметике в САПР ПЛИС Quartus II с использованием линии задержки на регистрах (рис. 3.29, *а*) и линии задержки на базе встроенных блоков ОЗУ (рис. 3.29, *б*). 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-разрядного сумматора к шине питания.

Прохождение единичного импульса по структуре КИХ-фильтра в случае использования линии задержки на регистрах показано на рис. 3.30. На выходе фильтра видим коэффициенты фильтра $C_0 = -2$, $C_1 = -1$, $C_2 = 7$ и $C_3 = 6$. На рис. 3.31 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. 3.32 и 3.33 показаны временные диаграммы в случае использования линии задержки на базе встроенных блоков ОЗУ.

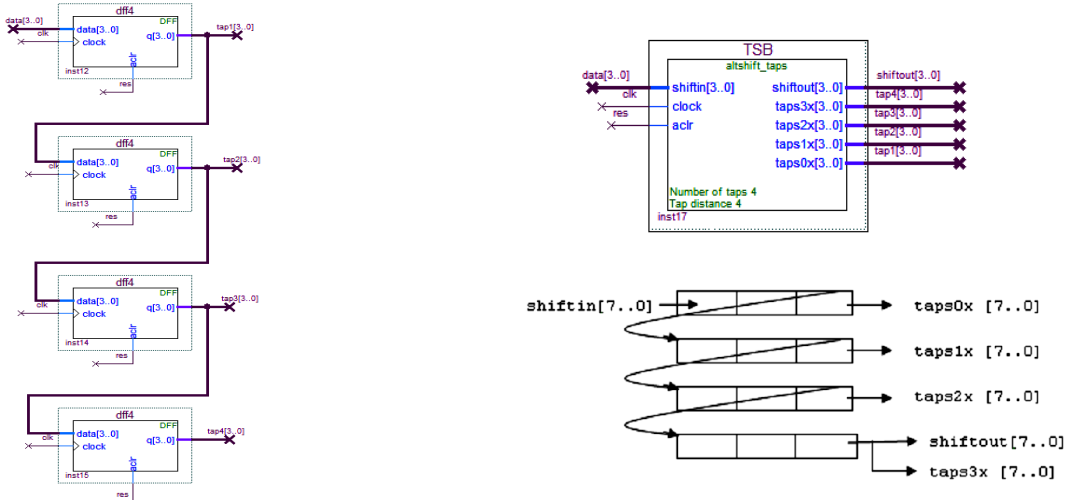
В тестировании принимают участие следующие структуры фильтров: классическая параллельная с использованием аппаратных ЦОС-боков (собран на мегафункции ALTMULT_ADD); систолическая структура; поведенческое описание на языке VHDL с использованием оператора цикла без привязки к какой-либо структуре, с использованием 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 позволяют для ПЛИС серии Cyclone III получить рекордное быстродействие за счет использования “безумножительных” схем умножения (табл. 3.4), не снижаемое 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).

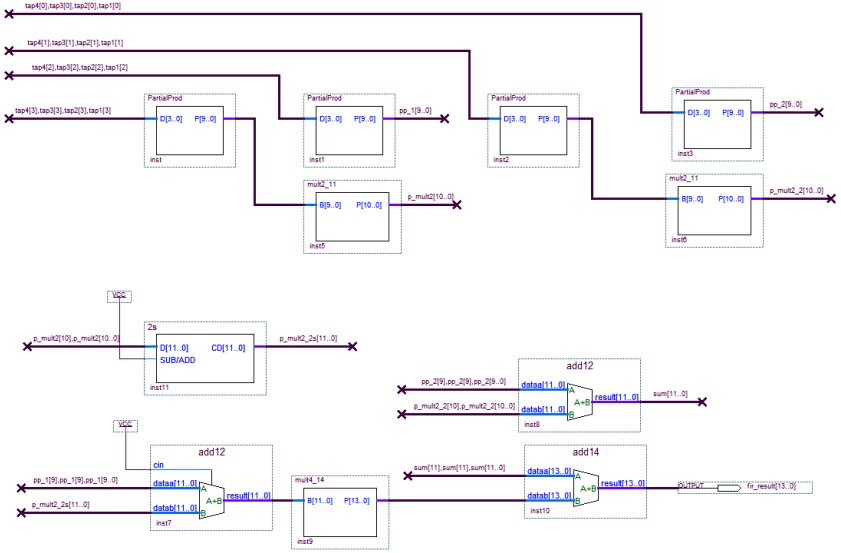
Использование кода языка VHDL и мегаядра FIR Compiler (структура фильтра задается пользователем опционально) в отличие от фильтров, структура которых разрабатывается “вручную”, приводит к пониженному быстродействию, однако при этом необходимо учитывать, что рассматриваемые примеры сильно упрощены. Для 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. 3.29. КИХ-фильтр на четыре отвода четыре 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 иерархического дерево многоразрядных сумматоров



в)

Рис. 3.29. КИХ-фильтр на четыре отвода четыре 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 иерархического дерево многоразрядных сумматоров (продолжение)

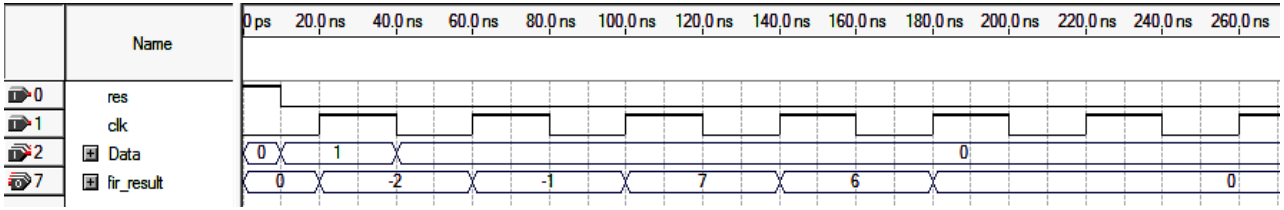


Рис. 3.30. Прохождение единичного импульса по структуре КИХ-фильтра в случае использования линии задержки на регистрах

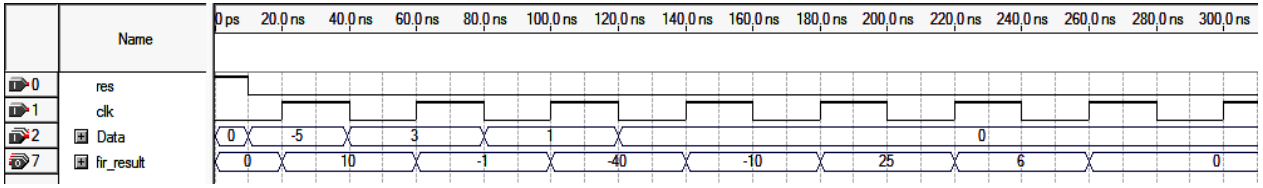


Рис. 3.31. Временные диаграммы работы фильтра в случае использования линии задержки на регистрах. На вход КИХ-фильтра поступают входные отсчеты -5, 3, 1 и 0. Правильные значения на выходе фильтра: 10, -1, -40, -10, 26, 6 и т.д.

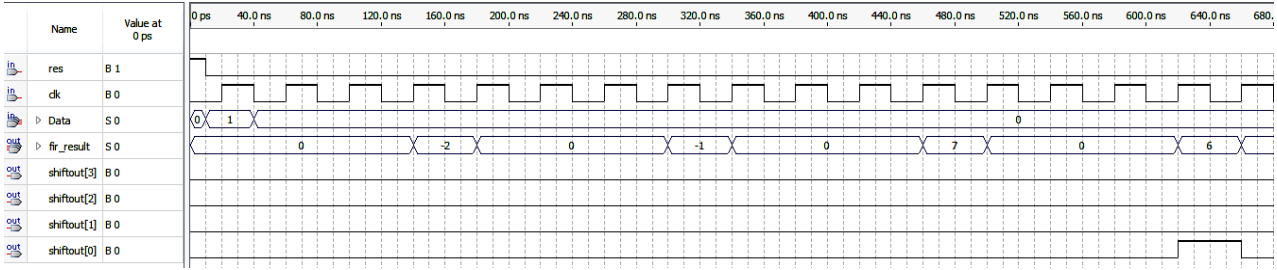


Рис. 3.32. Прохождение единичного импульса по структуре КИХ-фильтра в случае использования линии задержки на базе встроенных блоков ОЗУ

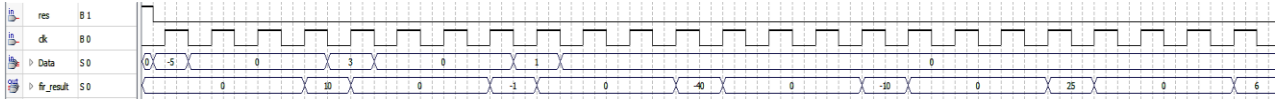


Рис. 3.33. Временные диаграммы работы фильтра в случае использования линии задержки на базе встроенных блоков ОЗУ

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

В данном разделе предлагается рассмотреть вопрос проектирования КИХ-фильтров на распределенной арифметике с использованием генератора параметризированных ядер XLogiCORE IP FIR Compiler Compiler v5.0. 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.3.34, *а* показана структурная схема КИХ-фильтра на четыре 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. 3.34, *б*) заменив их единственной таблицей перекодировок. На практике, для реализации адресуемых массивов комбинаций весовых коэффициентов фильтра используют таблицы перекодировок (LUT) ПЛИС (рис. 3.34, *в*).

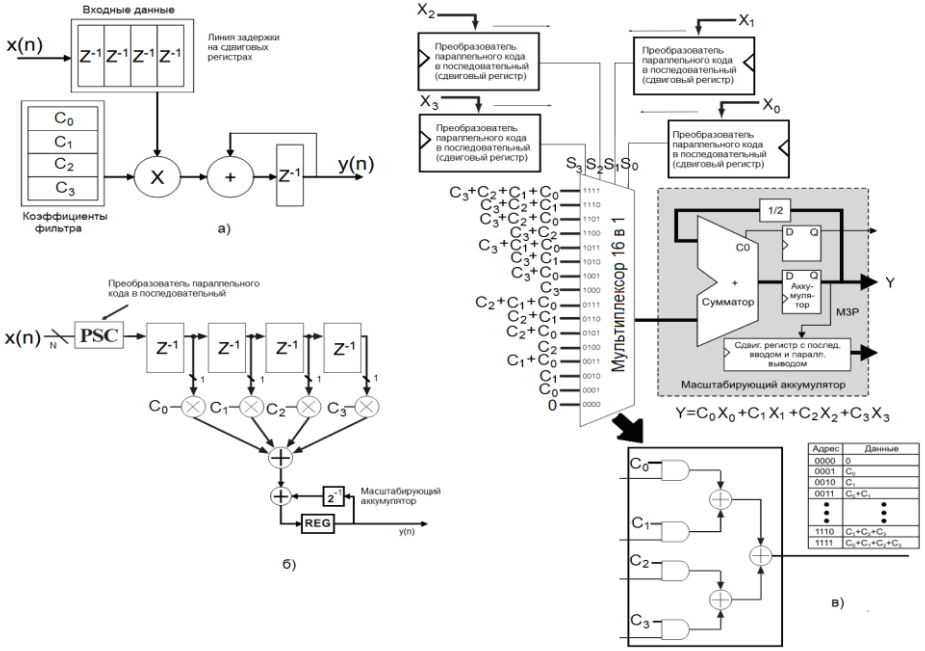


Рис. 3.34. Структуры КИХ-фильтров: а) – один MAC-фильтр; б) – последовательный КИХ-фильтр на четыре отвода; в) – упрощенное представление структуры КИХ-фильтра на четыре отвода с использованием последовательной распределенной арифметики

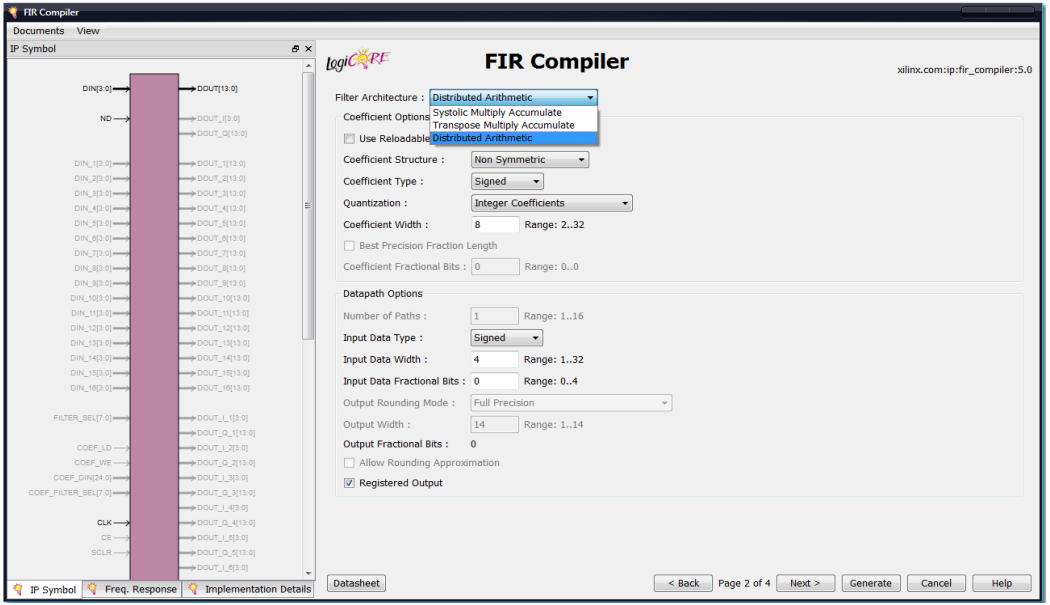


Рис. 3.35. Опции генератора параметризированных ядер XLogiCORE IP FIR Compiler Compiler v5.0. Выбирается структура фильтра на распределенной арифметике. Коэффициенты фильтра несимметричные, со знаком, целые, представляются с 8-битной точностью. Входные значения – целые со знаком, представляются с 4-битной точностью. Выходные значения представляются с 14-битной точностью

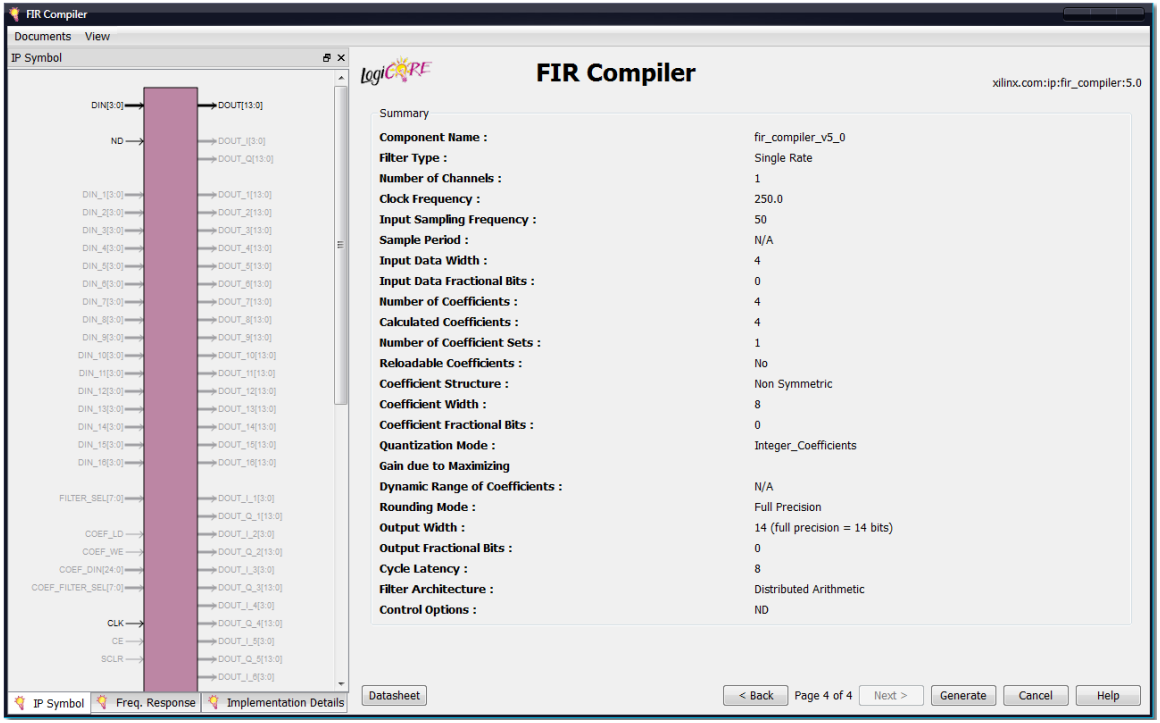


Рис. 3.36. Отчет о характеристиках КИХ-фильтра на четыре отвода

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 – восемь бит. 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 (B) – четыре бита, профильтрованного сигнала – четырнадцать бит (рис.3.35). 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 требуется 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. 3.36 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 – четыре бита. Структура коэффициентов фильтра – не симметричная, разрядность представления коэффициентов – восемь бит. Задана полная точность вычисления выходных

значений профильтрованного сигнала. Латентность фильтра восемь тактов синхрочастоты.

На рис. 3.37 представлен проект КИХ-фильтра на четыре 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 четыре отвода показывает пример 4. Моделирование прохождения сигнала по структуре КИХ-фильтра демонстрирует рис. 3.38. На вход фильтра подаются значения -5, 3, 1.

Разрешение на прием фильтром новых значений определяется высоким уровнем сигнала nd. Готовность фильтром 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 четыре такта синхрочастоты (рис. 3.38).

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;
```

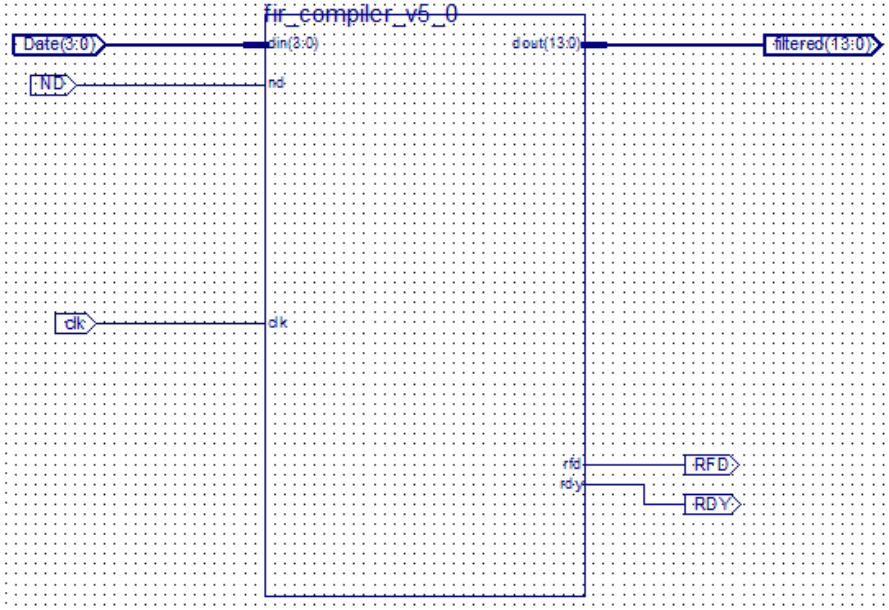


Рис. 3.37. Проект КИХ-фильтра на четыре отвода в САПР ПЛИС Xilinx ISE 14.2 с использованием генератора параметризированных ядер XLogiCORE IP FIR Compiler Compiler v5.0

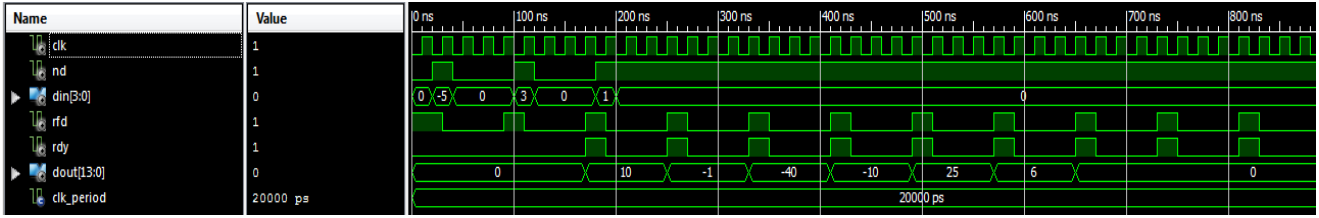


Рис. 3.38. Моделирование прохождения сигнала по структуре КИХ-фильтра на последовательной распределенной арифметике. На вход фильтра подаются значения -5, 3, 1. Правильные значения на выходе 10, -1, -40, -10, 25, 6. Латентность фильтра 8 тактов синхрочастоты

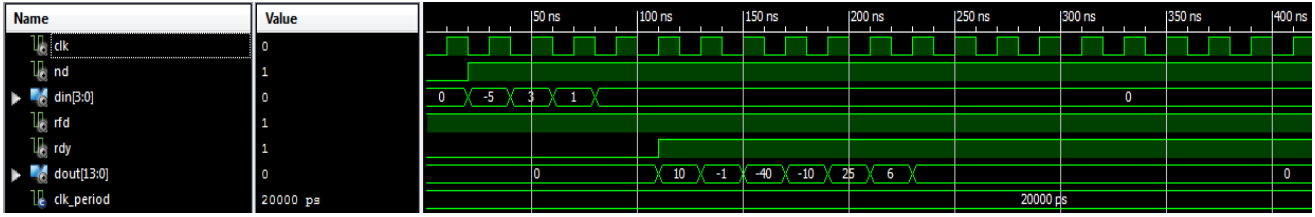


Рис. 3.39. Моделирование прохождения сигнала по структуре КИХ-фильтра на параллельной распределенной арифметике. На вход фильтра подаются значения -5, 3, 1. Правильные значения на выходе 10, -1, -40, -10, 25, 6. Латентность фильтра 5 тактов синхрочастоты

```
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. 3.39 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. 4.2 приведен анализ задействованных ресурсов ПЛИС XC6SLX4 при реализации КИХ-фильтров на четыре 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.

Таблица 3.4

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	Параллельная
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

Рассмотрим размещение КИХ-фильтра в ресурсы ПЛИС Xilinx XC6SLX4 (рис. 3.40). Во всех случаях установлена целевая задача проектирования – сбалансированная. 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. 3.40, *а*) и фильтров с

использованием распределенной арифметики (рис. 3.40, *а* и рис. 3.41, *б*) 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 тоже время ведущие разработчики САПР Xilinx и Altera сохранили возможность использования распределенной арифметики из за ряда преимуществ. Например, структуры КИХ-фильтров на основе распределенной арифметике 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 так же реализовать полифазный банк фильтров и параллельную потоковую обработку информации.

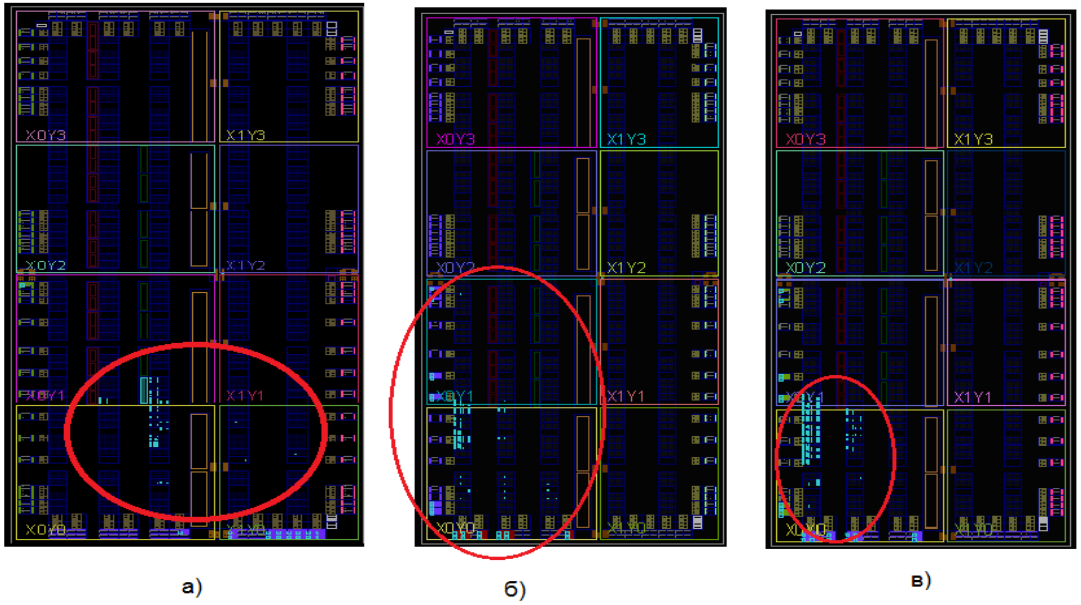


Рис. 3.40. Размещение КИХ-фильтра на четыре отвода в ресурсы ПЛИС XC6SLX4: а) систолическая структура с использованием ЦОС-блока; б) последовательная арифметика; в) параллельная арифметика

4. СИСТОЛИЧЕСКИЕ КИХ-ФИЛЬТРЫ В БАЗИСЕ ПЛИС

4.1. 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 версии 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. 4.1, *а*). 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. 4.1, *б*).

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 четыре отвода (рис. 4.2). В основе структуры лежит

ритмическое прохождение двух потоков данных x_{in} и y_{in} навстречу друг другу.

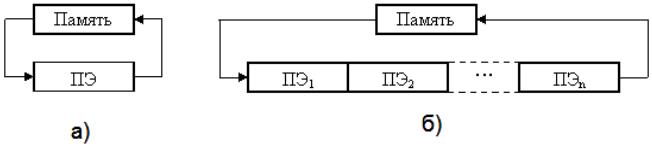


Рис. 4.1. Обработка данных в вычислительных системах:
а) фон-неймановского типа; б) – систолической структуры

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. 4.2). 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. 4.2 показана структура систолического КИХ-фильтра на четыре отвода. Потоки сигналов y_{in}, представляющего накопленный результат вычисления скалярного произведения и входного x_{in} распространяются слева на право. 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.

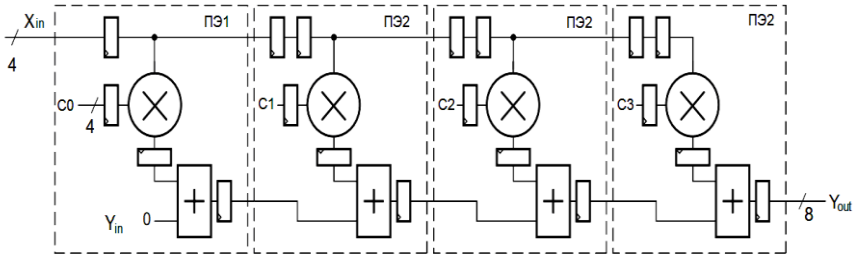


Рис. 4.2. Систолический КИХ-фильтр на четыре отвода составленный из четырех секций DSP48 ПЛИС Virtex-4 фирмы Xilinx

Используя представленные выше соображения разработаем модель систолического фильтра на четыре 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 перемножителей используем мегафункцию LPM_MULT. Регистры выполнены на мегафункциях LPM_FF. Систолический КИХ-фильтр на четыре отвода с процессорными элементами ПЭ1 и ПЭ2 в САПР ПЛИС Quartus II показан на рис. 4.3. На рис. 4.4 и рис. 4.5 показаны схемы процессорных элементов ПЭ1 и ПЭ2, а на рис. 4.6 - временные диаграммы работы систолического КИХ-фильтра на четыре отвода.

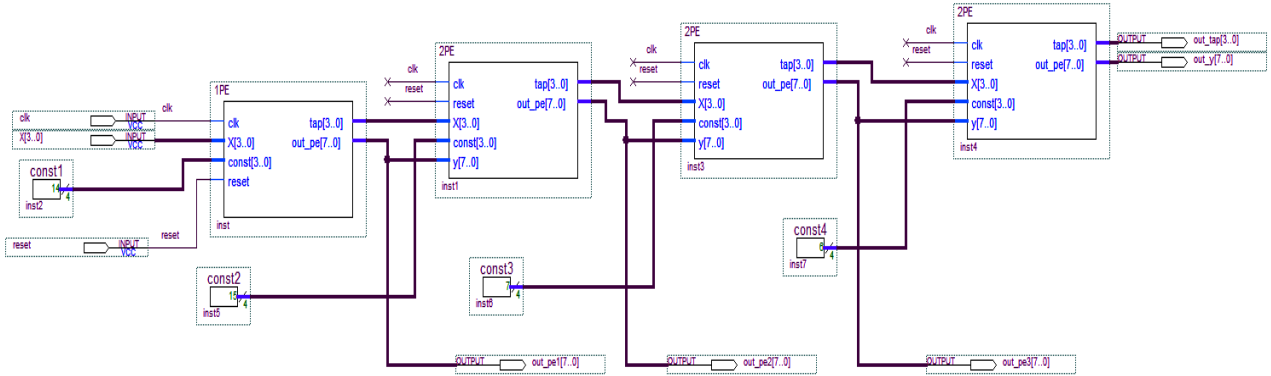


Рис. 4.3. Систолический КИХ-фильтр на четыре отвода в САПР ПЛИС Quartus II с процессорными элементами ПЭ1 и ПЭ2

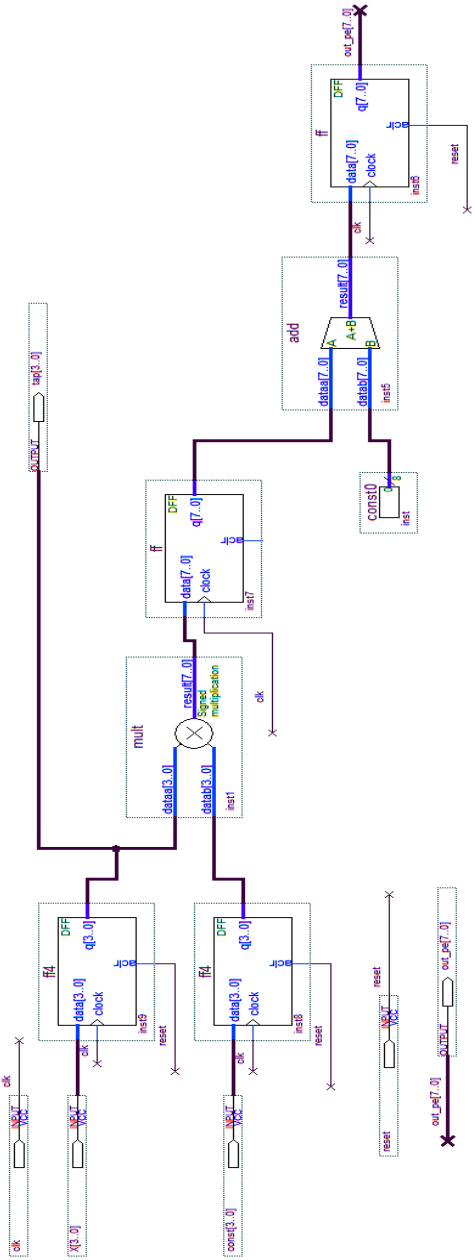


Рис. 4.4. Первый процессорный элемент ПЭ1

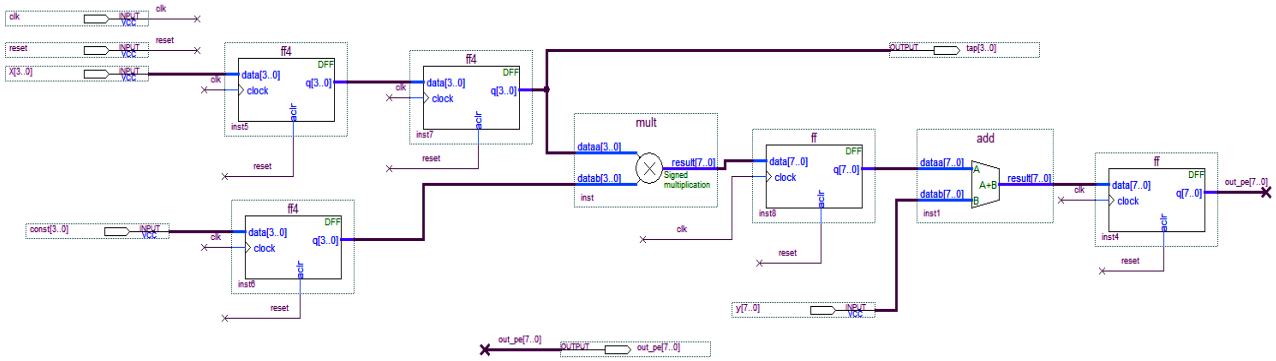


Рис. 4.5. Второй процессорный элемент ПЭ2

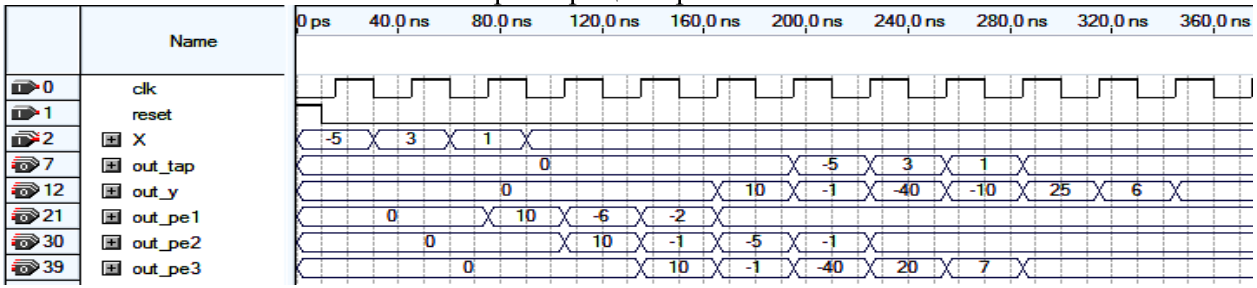


Рис. 4.6. Временные диаграммы работы систолического КИХ-фильтра на четыре отвода с процессорными элементами ПЭ1 и ПЭ2

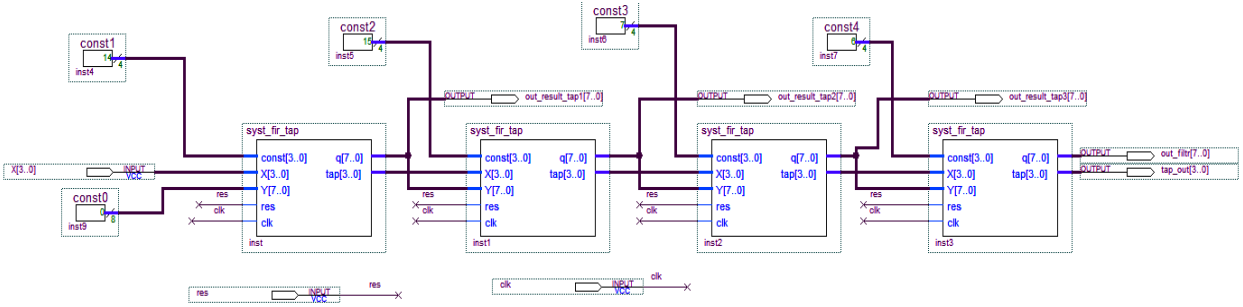


Рис. 4.7. Систолический КИХ-фильтр на четыре отвода в САПР ПЛИС Quartus II с однотипными процессорными элементами

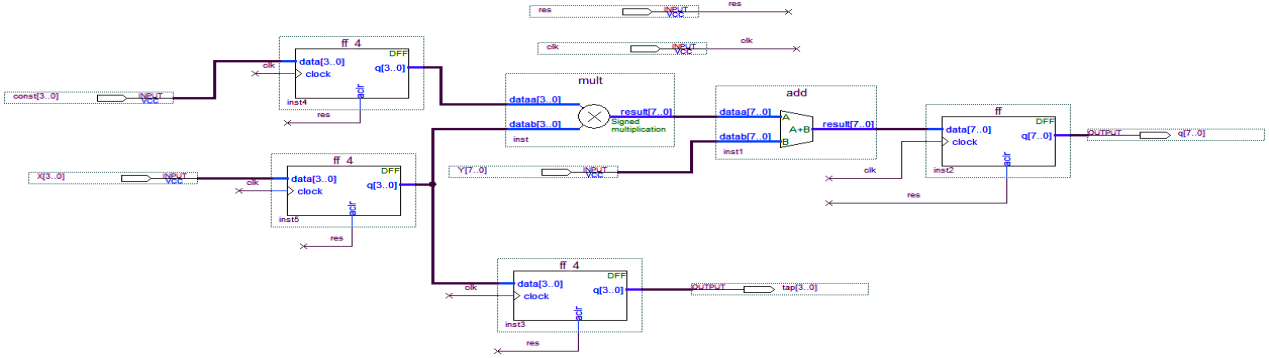


Рис. 4.8. Однотипный процессорный элемент

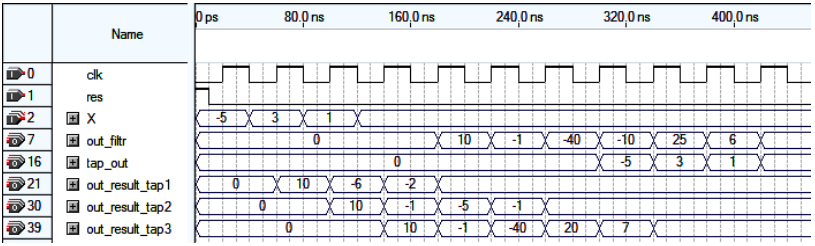


Рис. 4.9. Временные диаграммы работы систолического КИХ-фильтра на четыре отвода с однотипными процессорными элементами

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. 4.7 и рис. 4.8). Правильность функционирования такого решения подтверждает диаграмма на рис. 4.9.

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. 4.10. Сравнивая временные диаграммы на рис. 4.6, 4.9 и рис. 4.10, 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

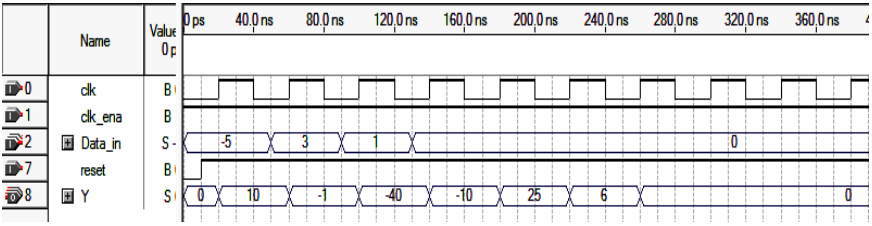


Рис. 4.10. Временные диаграммы работы КИХ-фильтра на четыре отвода по коду языка VHDL

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. 4.11 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.

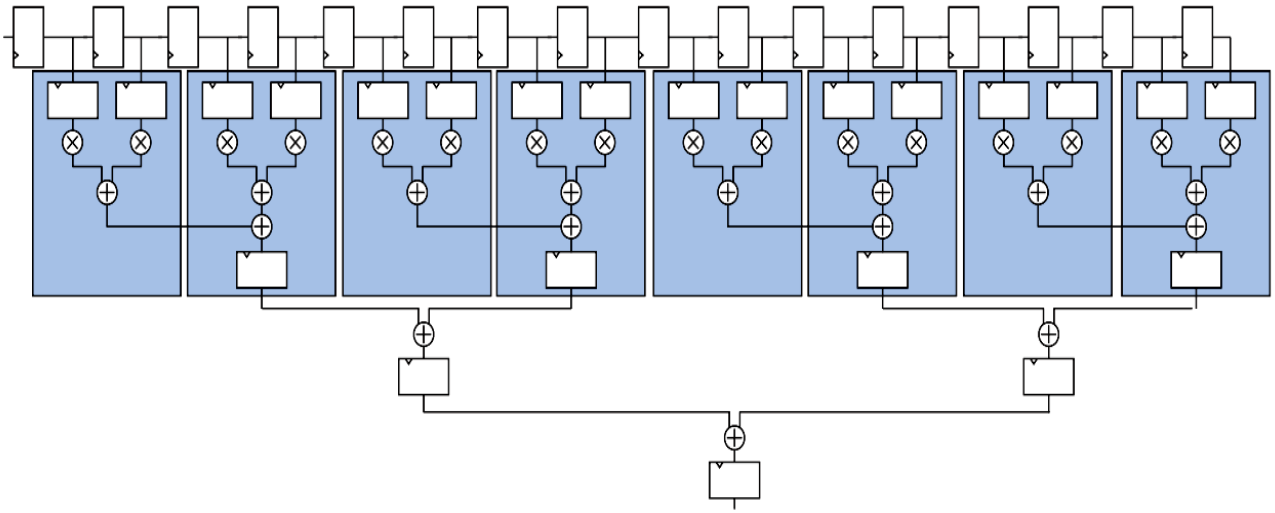
На рис. 4.12 показана прямая форма КИХ-фильтра на 8 отводов с несимметричными коэффициентами с

использованием ЦОС-блоков 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 ресурсах и, как следствие, снижается быстродействие фильтра за счет увеличения задержек в трассировочных каналах самой ПЛИС.

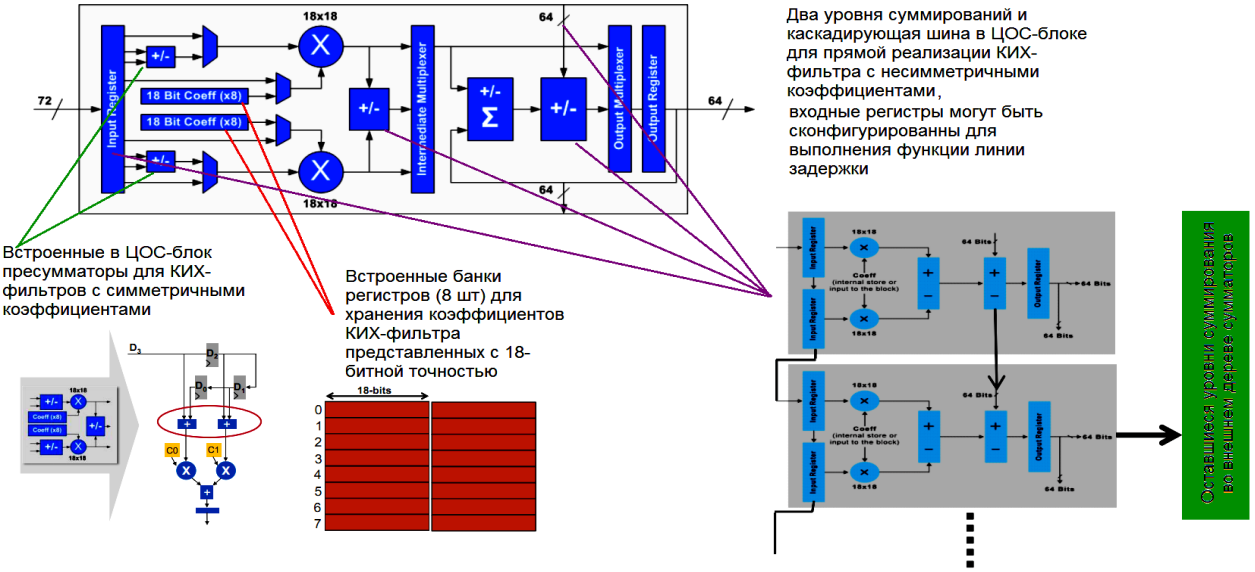
Существенно уменьшить число используемых ресурсов позволяет систолический КИХ-фильтр, в ЦОС-блоках которого реализуются операции умножения и сложения с конвеиеризацией, т.е. отпадает необходимость в дереве многоразрядных сумматоров, однако он может иметь большую латентность по сравнению с прямой реализацией фильтра (рис. 4.13). 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. 4.14 представлена структура ЦОС-блока с переменной точностью ПЛИС серии Cyclon V, на которой, выделен процессорный элемент, а на рис. 4.15 показано включение систолического регистра в мегафункции ALTMULT_ADD для ПЛИС серии Cyclon V.



а)

Рис. 4.11. 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. 4.11. 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)

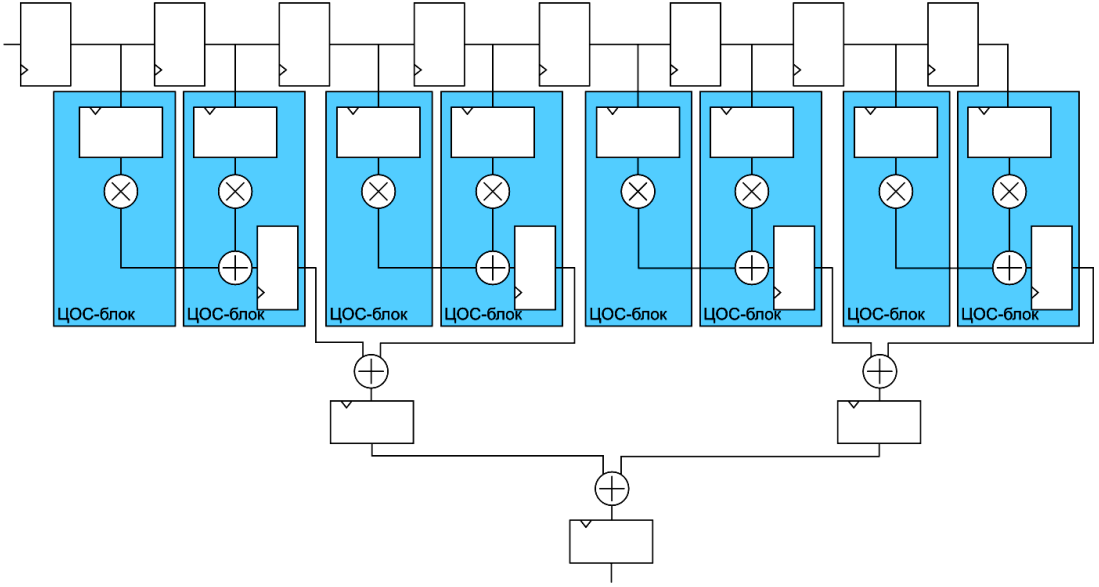


Рис. 4.12. Прямая реализация КИХ-фильтра с несимметричными коэффициентами на восемь отводов, коэффициенты представлены с 27-битной точностью (высокоточный режим) с использованием ЦОС-блоков ПЛИС серии Stratix V. В ЦОС-блоках используется один уровень суммирования по отношению к внешнему дереву многоразрядных сумматоров

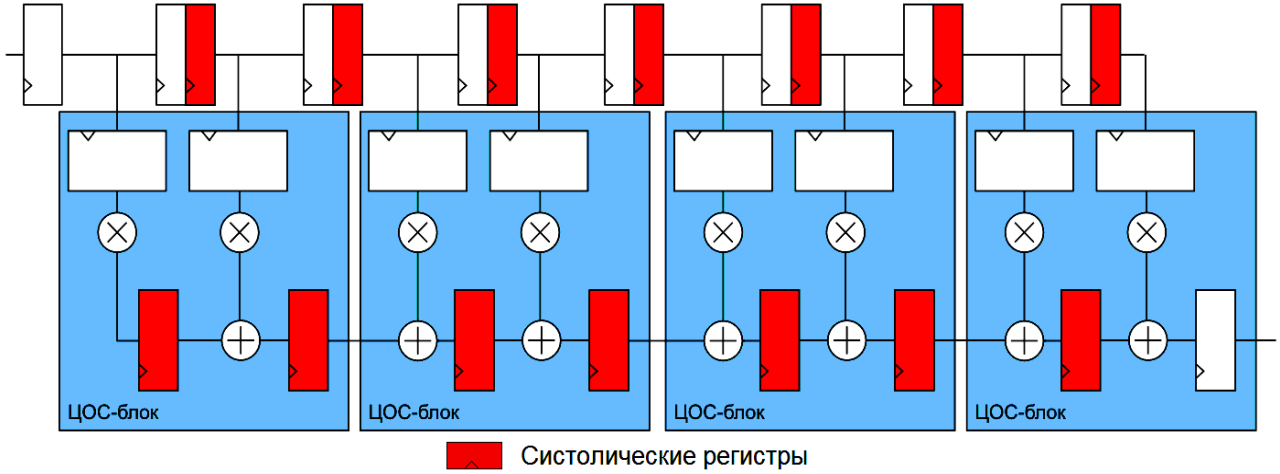


Рис. 4.13. Систолический КИХ-фильтр на восемь отводов с 18-битной точностью представления коэффициентов с использованием ЦОС-блоков ПЛИС серии Stratix V

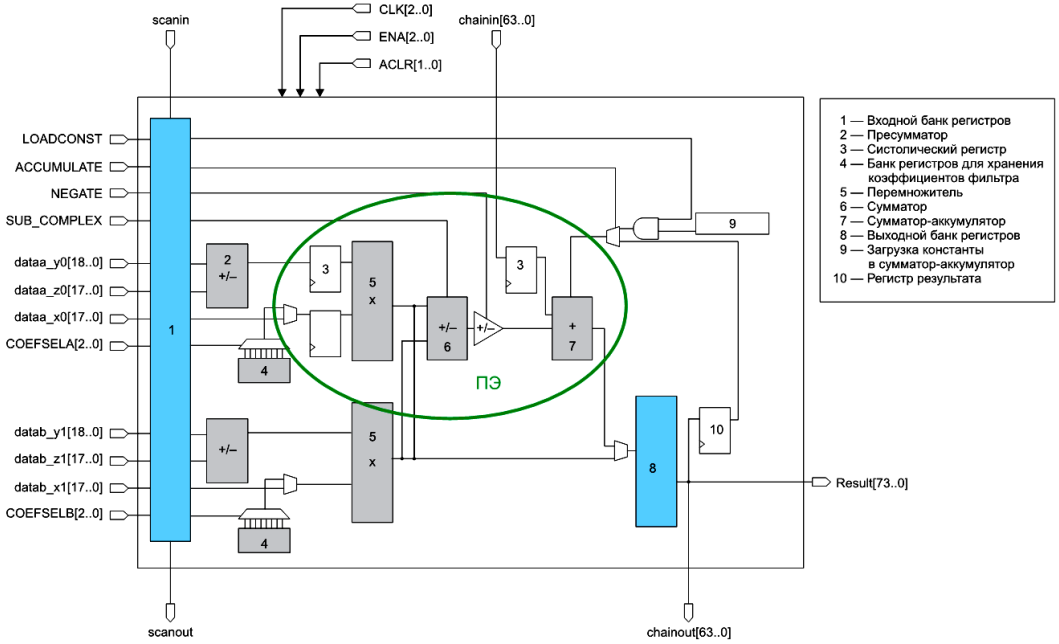


Рис. 4.14. Систолические регистры, подключаемые опционально с помощью мегафункции ALTMULT_ADD в ЦОС-блоке с переменной точность ПЛИС серии Cyclon V

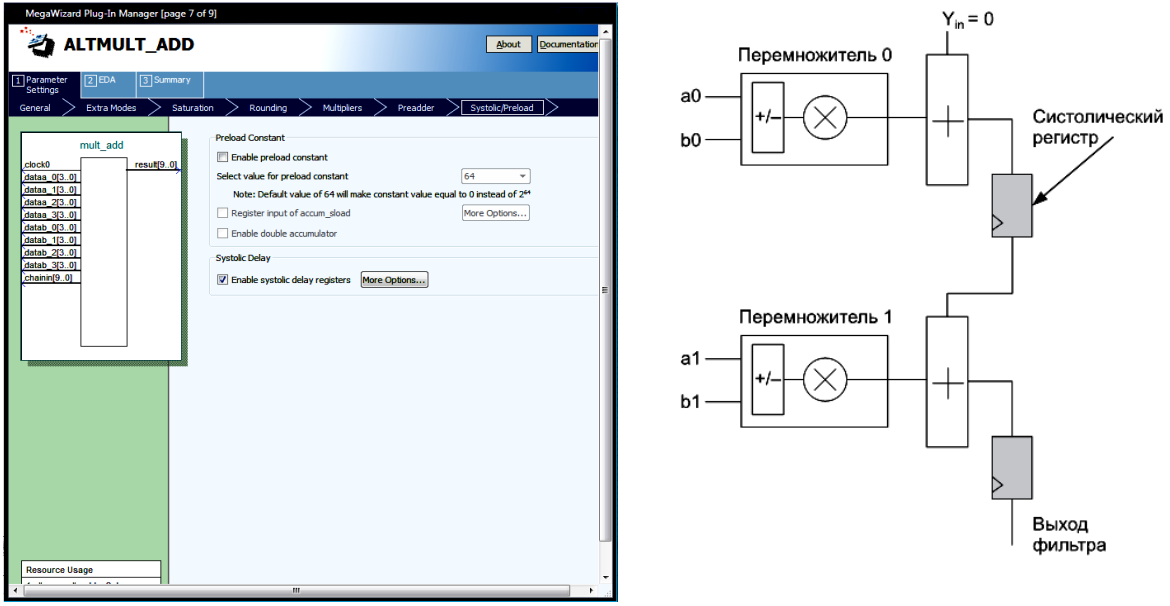


Рис. 4.15. Включение систолического регистра в последовательную цепь сумматоров в мегафункции ALTMULT_ADD для ПЛИС серии Cyclon V

Реализация параллельных КИХ-фильтров на четыре 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, 18x18 бит	-	16	4

Реализации параллельных КИХ-фильтров на четыре отвода в базис ПЛИС серии Stratix III представлены в таблице. Все рассмотренные варианты фильтров реализованы на мегафункциях. Первый вариант реализован без использования мегафункциями ЦОС-блоков (рис. 4.7), второй вариант задействует лишь по одному умножителю в блоке из четырех возможных мегафункции ALTMULT_ACCUM (рис. 2.13), а третий – четыре умножителя из четырех возможных в блоке мегафункции ALTMULT_ADD (рис.2.14). 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.

4.2. 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:

$$y = \sum_{k=0}^{K-1} c_k \cdot x_k ,$$

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 четыре 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. 4.16 и рис. 4.17). 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. 4.18 показаны временные диаграммы работы систолического КИХ-фильтра реализованного в базисе ПЛИС Stratix III на четыре отвода с однотипными процессорными элементами в САПР ПЛИС Quartus II версии 9.1.

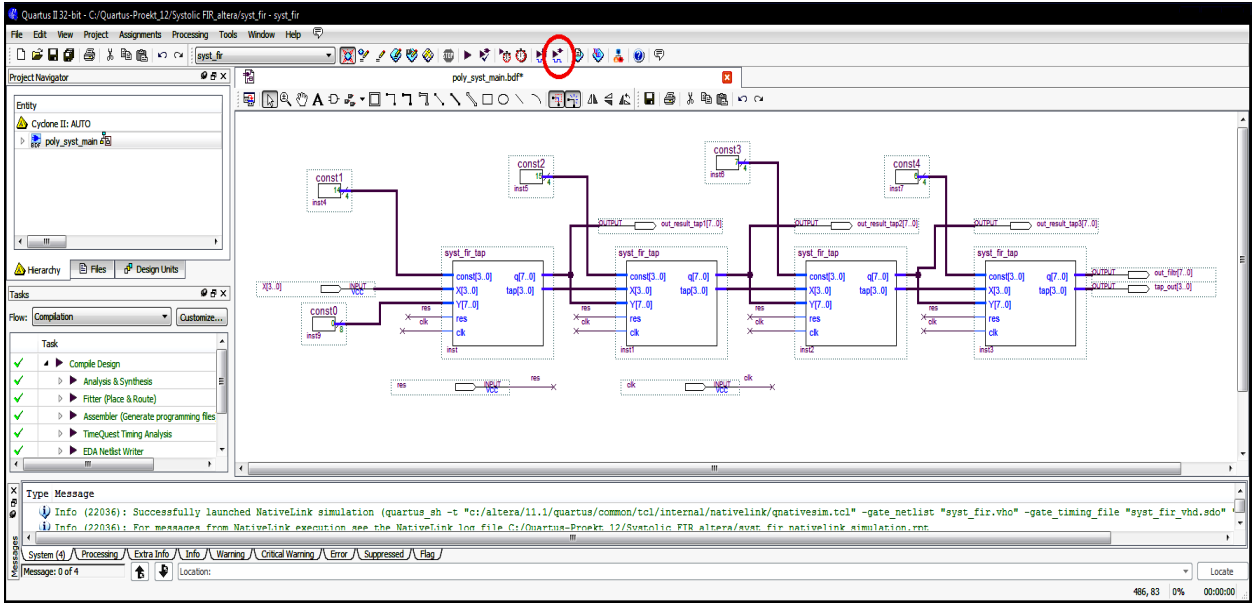


Рис. 4.16. Систолический КИХ-фильтр на четыре отвода в САПР ПЛИС Quartus II версии 11.1 с однотипными процессорными элементами

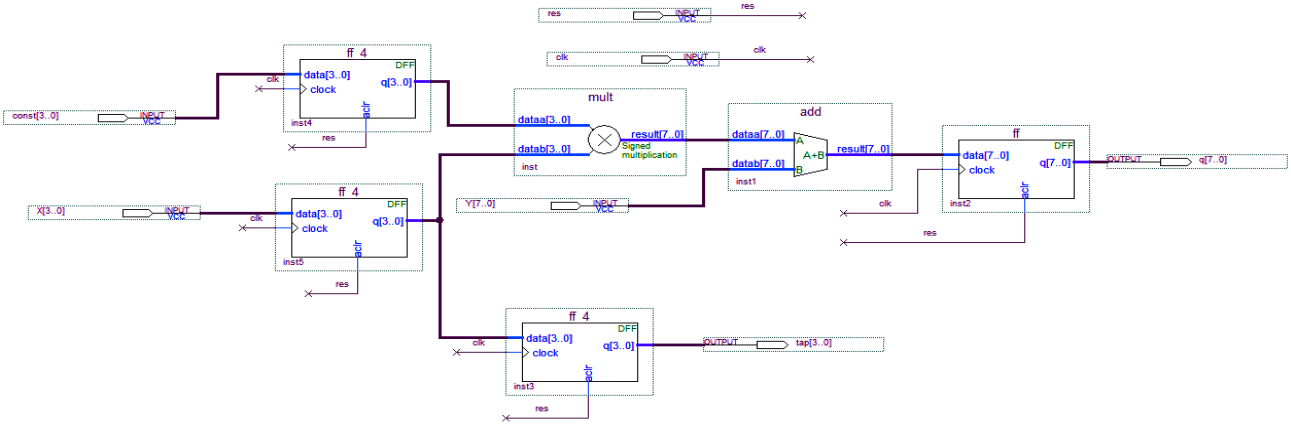


Рис. 4.17. Однотипный процессорный элемент

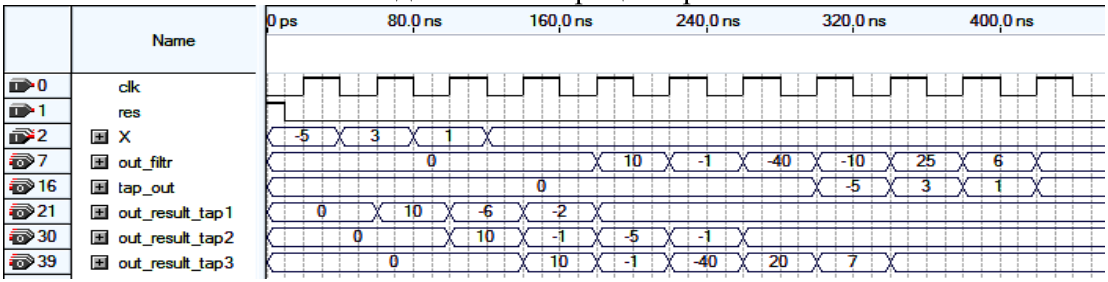


Рис. 4.18. Временные диаграммы работы систолического КИХ-фильтра на четыре отвода с однотипными процессорными элементами в САПР ПЛИС Quartus II версии 9.1

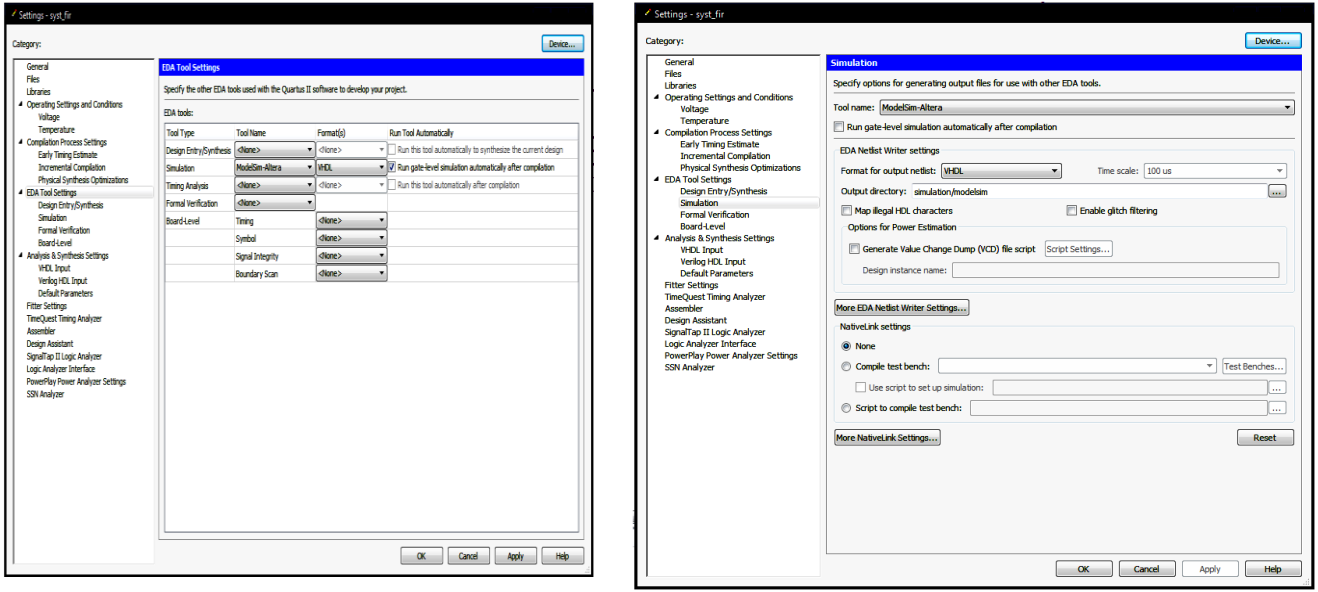


Рис. 4.19. Настройки закладки EDA Tool Settings САПР ПЛИС Quartus II версии 11.1

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 “нетлиста” – язык VHDL (рис. 4.19).

Предварительно необходимо создать текстовый сценарий функционирования систолического КИХ-фильтра с использованием структурного стиля языка VHDL (“тестбенч”). 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. 4.16 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. 4.20). Двойным щелчком мыши по объекту test_poly_syst_main автоматически запускаются различные вспомогательные окна (рис. 4.21).

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 ns, получим временные диаграммы работы КИХ-фильтра (рис. 4.21). Сравниваем полученные результаты с временными диаграммами на рис. 4.18, убеждаемся в правильности работы систолического КИХ-фильтра на четыре отвода в базисе ПЛИС Cyclone II.

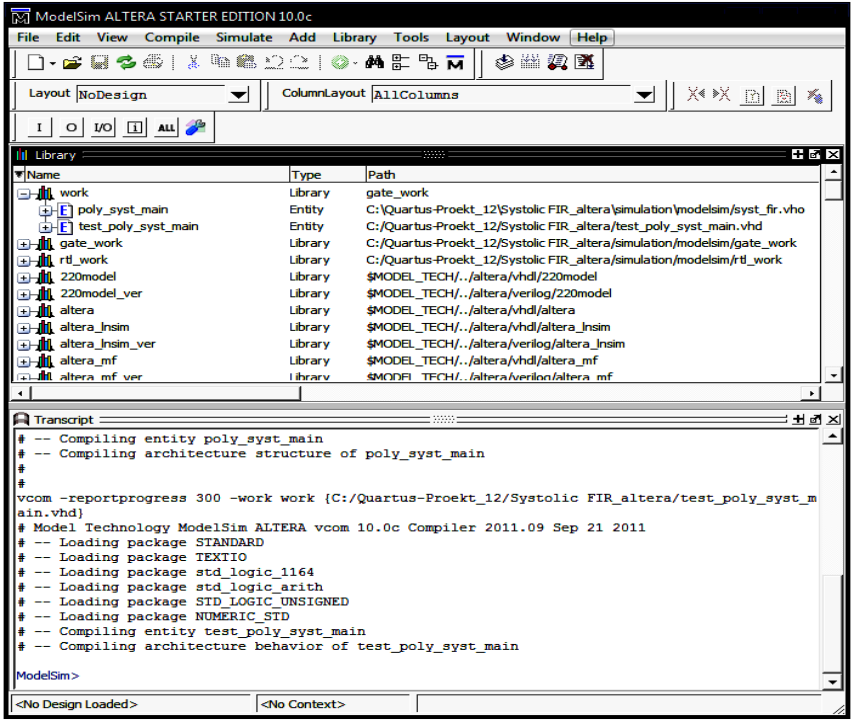


Рис. 4.20. Создается рабочая библиотека work, в которую помещаются два объекта poly_syst_main и test_poly_syst_main

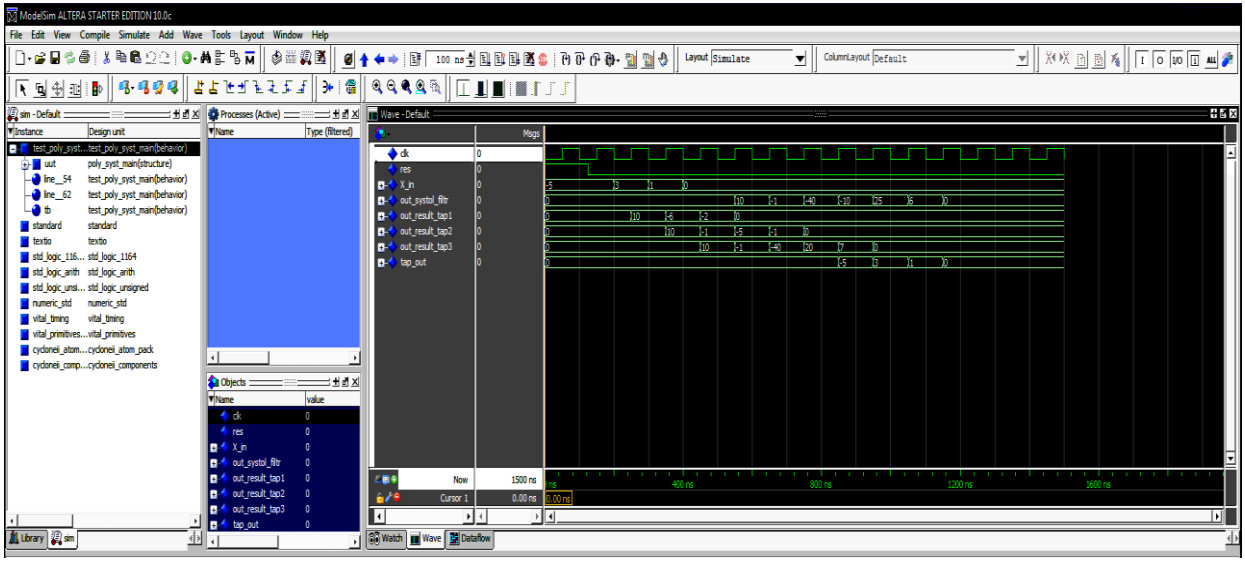


Рис. 4.21. Временные диаграммы работы систолического КИХ-фильтра на четыре отвода с однотипными процессорными элементами в симуляторе ModelSim-Altera версии 10.c

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 ns, коэффициент заполнения - 50, уровни логической единицы и нуля, радиокнопкой выбираем активным передний фронт синхросигнала (рис. 4.22).

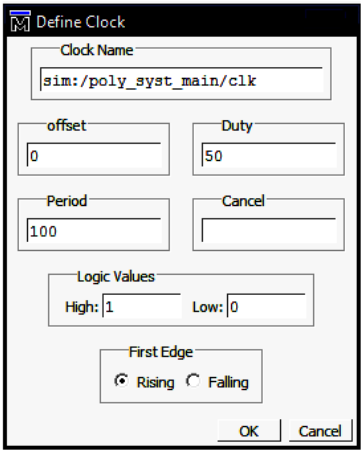


Рис. 4.22. Окно настройки тактового синхросигнала clk

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. 4.23).

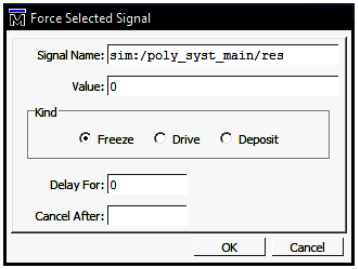


Рис. 4.23. Настройка сигнала сброса res

Далее выбираем сигнал X и с помощью меню Force задаем десятичное число -5 (рис. 4.24). 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. 4.25). Задание для моделирования (пример 4) можно использовать, повторно сохранив в текстовый файл.

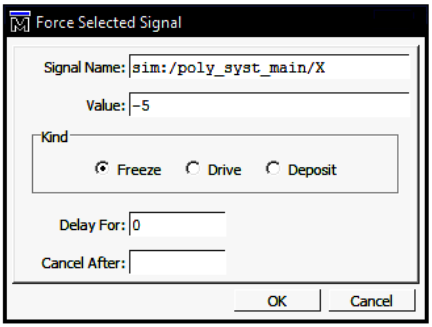


Рис. 4.24. Настройка сигнала X

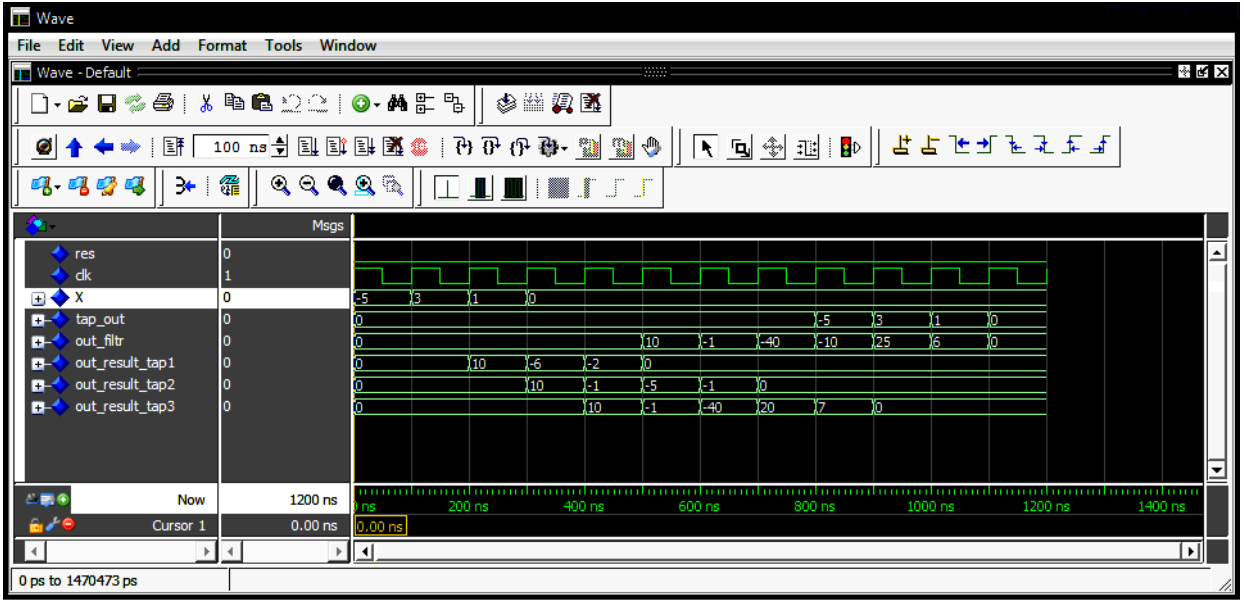


Рис. 4.25. Окно Wave с результатами моделирования КИХ-фильтра

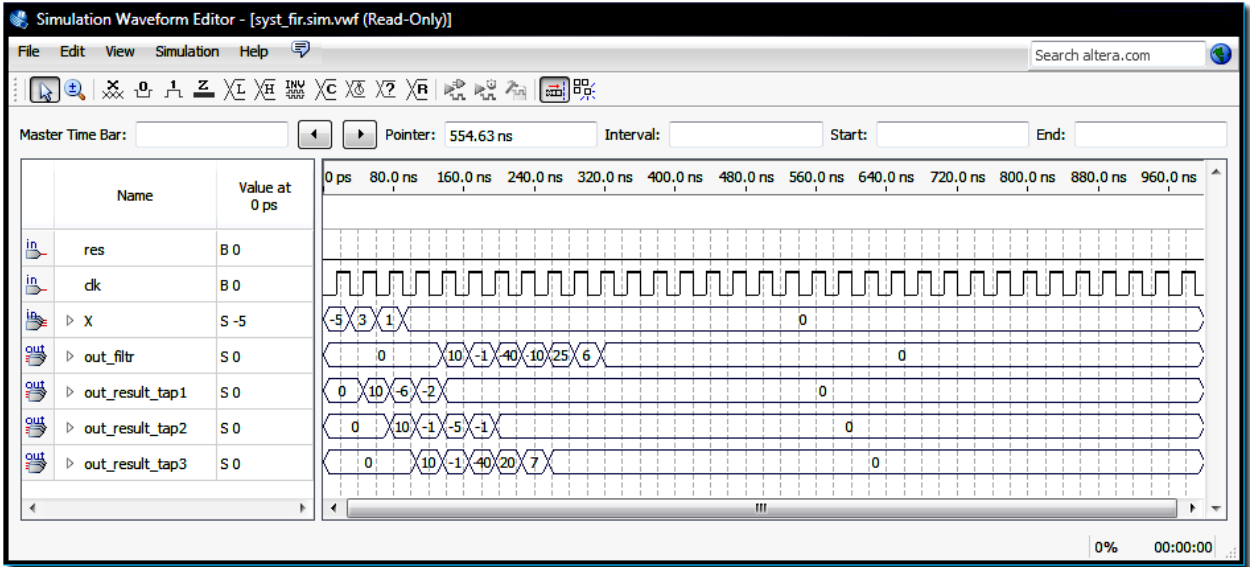


Рис. 4.26. Временные диаграммы работы систолического КИХ-фильтра на четыре отвода с однотипными процессорными элементами в САПР ПЛИС Quartus II версии 13.0

```
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. 4.26 показаны временные диаграммы работы систолического КИХ-фильтра на четыре 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.

4.3. 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. 4.27 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. 4.28).

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. 4.28 показан учет эффектов квантования. Коэффициенты фильтра не симметричные, целочисленные со знаком $C_0 = -2$, $C_1 = -1$, $C_2 = 7$ и $C_3 = 6$, разрядность представления коэффициентов – четыре 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 – восемь бит. Под дробную часть числа в обоих случаях отводим 0 бит (Input Data Fractional Bits и Output Fractional Bits) (рис. 4.29).

На рис. 4.30 показан выбор структуры фильтра. Используем прямую форму систолического фильтра, в котором операции умножения и сложения выполняются параллельно с конвейеризацией (рис. 4.31). В каждой секции доступно 4 ЦОС-блока располагающиеся 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.

Так же 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.

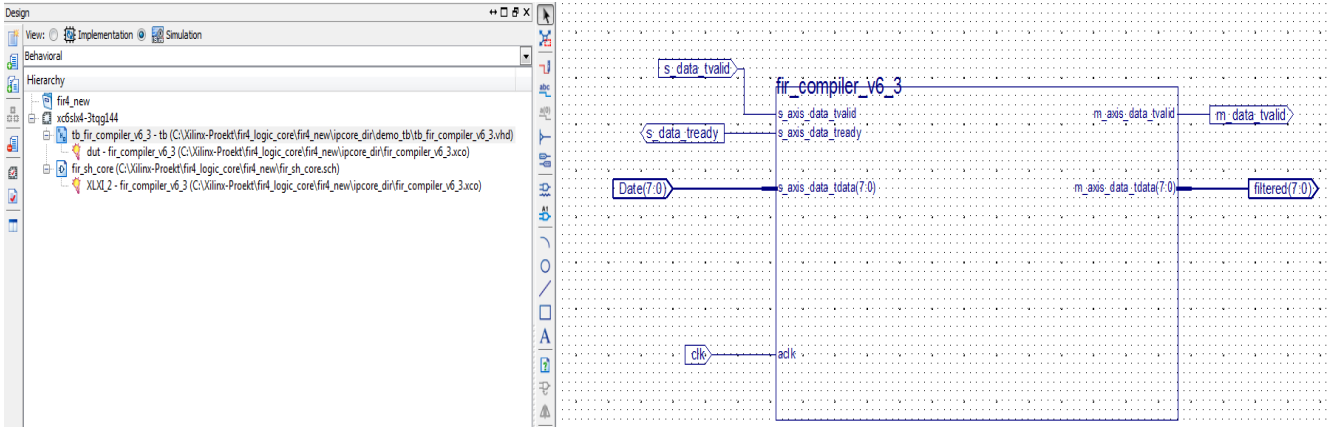


Рис. 4.27. Проект КИХ-фильтра в САПР ПЛИС Xilinx ISE 14.2 с использованием генератора параметризированных ядер XLogiCORE IP FIR Compiler v6.3. Верхний уровень иерархии - схемный файл (sch-файл)

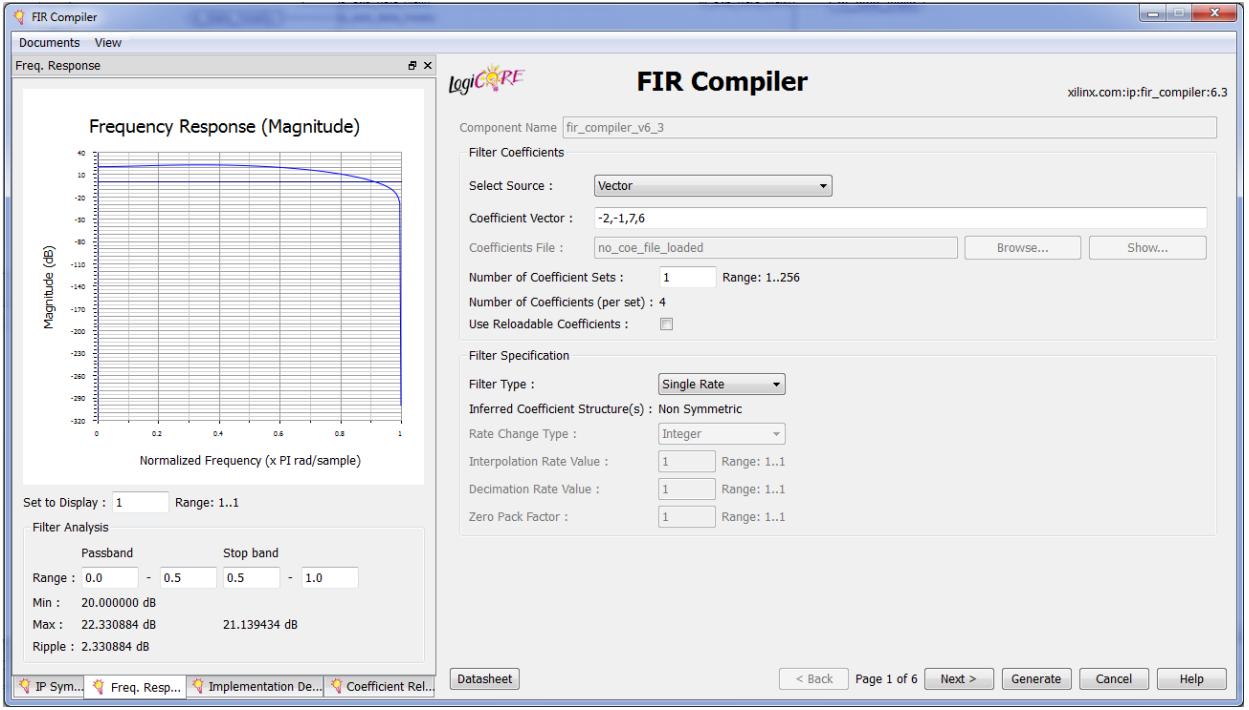


Рис. 4.28. Настройки функции FIR Compiler

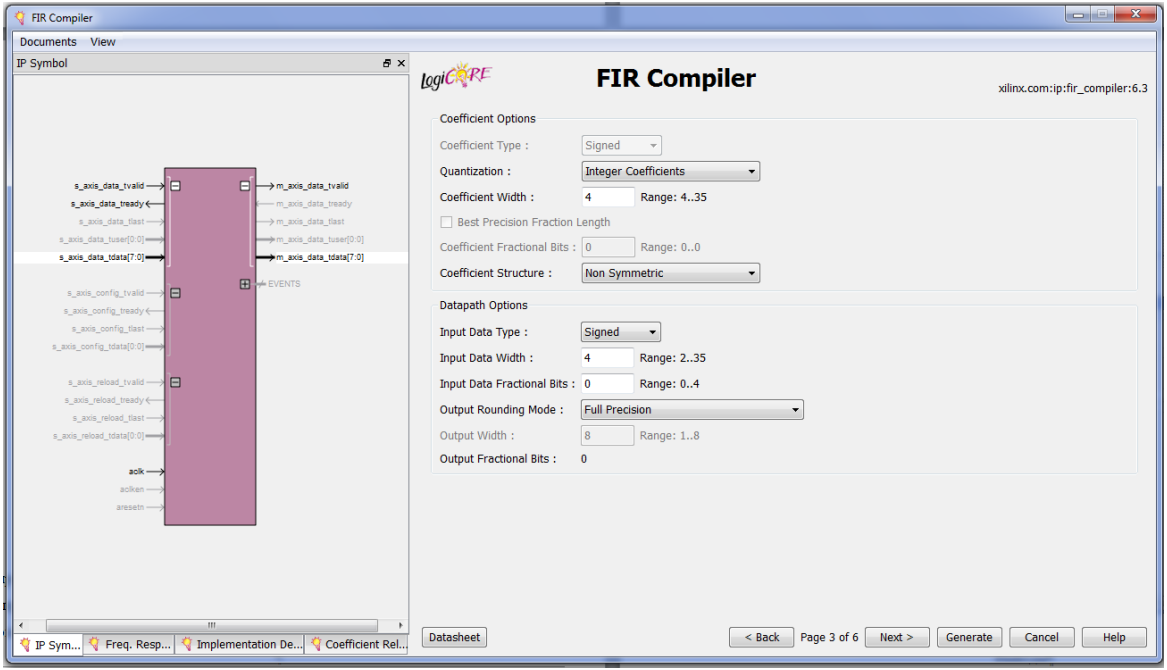
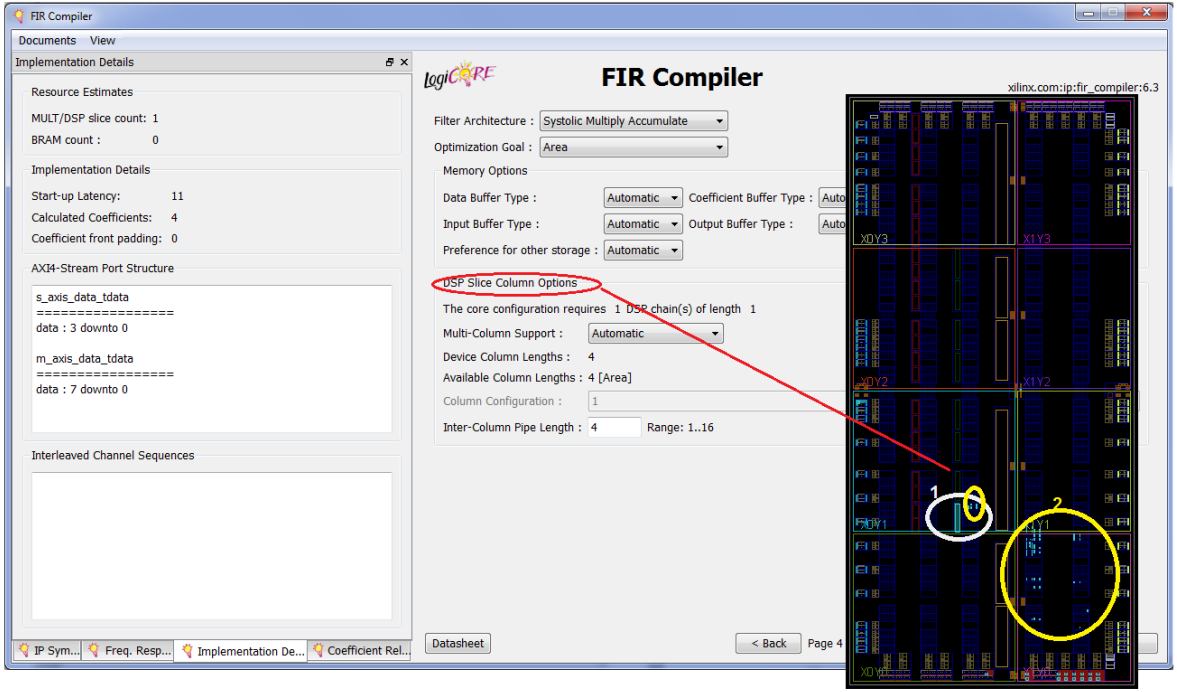


Рис. 4.29. Учет эффектов квантования коэффициентов КИХ-фильтра, точности представления входных и выходных значений (сигналов). Опции позволяющие настроить форматы представления коэффициентов и входных/выходных значений фильтра



1 ЦОС-блок DSP48A1;
2 Логические ресурсы

Рис. 4.30. 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. 4.32 показан симулятор ISim САПР ПЛИС Xilinx ISE 14.2. Демонстрируется прохождение единичного импульса по структуре фильтра (импульсная характеристика КИХ-фильтра на четыре отвода полученная с использованием тестбенча сгенерированного в автоматическом режиме). Латентность фильтра 11 тактов синхрочастоты. Для размещения проекта в базис ПЛИС XC6SLX4 требуется 48 триггеров тактируемых фронтом синхросигнала из общих логических ресурсов ПЛИС и один ЦОС-блок DSP48A1.

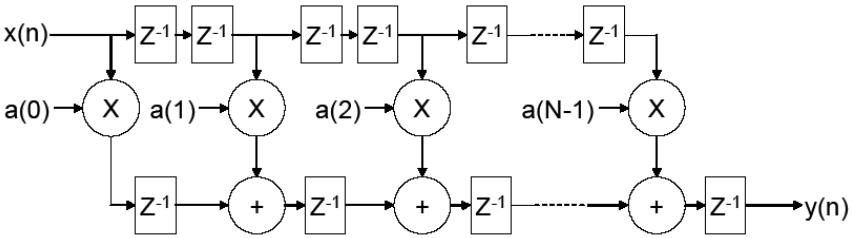


Рис. 4.31. Прямая форма систолического КИХ-фильтра с конвериезацией

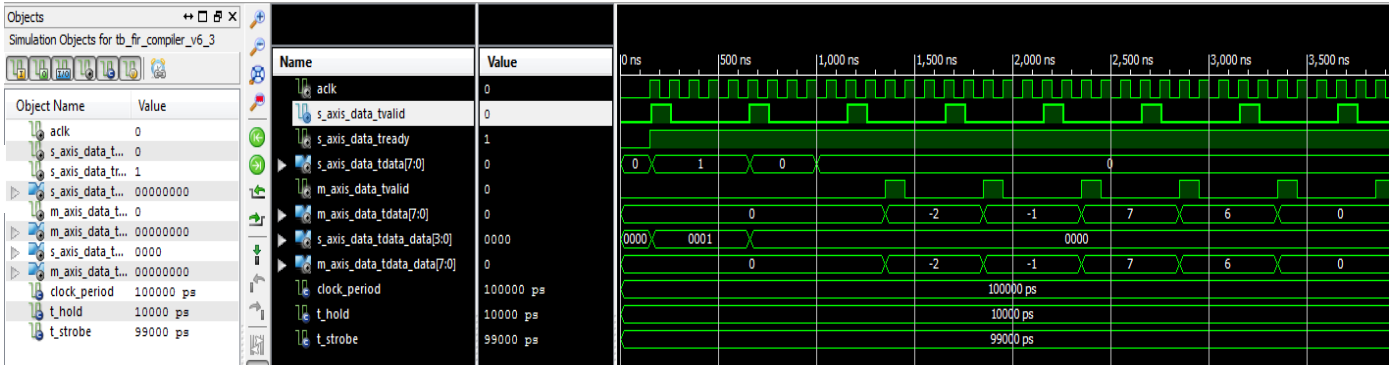


Рис. 4.32. Симулятор ISim САПР ПЛИС Xilinx ISE 14.2. Импульсная характеристика КИХ-фильтра на четыре отвода полученная с использованием тестбенча сгенерированного в автоматическом режиме

5. ИСПОЛЬЗОВАНИЕ СИСТЕМЫ ВИЗУАЛЬНО-ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ MATLAB/SIMULINK ДЛЯ ПРОЕКТИРОВАНИЯ КИХ-ФИЛЬТРОВ В САПР ISE DESIGN SUITE

5.1. 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 Matlab/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 Matlab/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 Matlab/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 являющимся аналогом функции FIR Compiler v5.0 САПР

Xilinx ISE получаемой с помощью генератора параметризированных ядер XLogiCORE IP (рис. 5.1).

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. 5.2). На рис. 5.2 показаны настройки блока. Согласно настройкам блока реализуется КИХ-фильтр на параллельной распределенной арифметике. Результаты имитационного моделирования представлены на рис. 5.3.

Для представления чисел со знаком в формате с фиксированной запятой Xilinx System Generator использует нотацию FIX, а для без знаковых - 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. 5.1). 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. 5.4 и рис. 5.5 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. 5.6 показана имитационная модель КИХ-фильтра на 4 отвода с перегружаемыми коэффициентами с использованием блока FIR Compiler v5.0, а на рис. 5.7 результаты имитационного моделирования.

Рассмотрим разработку имитационной модели систолического КИХ-фильтра (тип фильтра - Single-Rate FIR) с использованием блока FIR Compiler v6.3 являющимся аналогом функции FIR Compiler v6.3 САПР Xilinx ISE получаемой с помощью генератора параметризированных ядер XLogiCORE IP (рис. 5.8). На рис. 5.9 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. 5.10 показано имитационное моделирование.

С помощью блока System Generator создадим в автоматическом режиме проект фильтра и испытательный стенд. На рис. 5.11 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:

$$y/256 = C_0(x_0 * 256) + C_1(x_1 * 256) + C_2(x_2 * 256) + C_3(x_3 * 256).$$

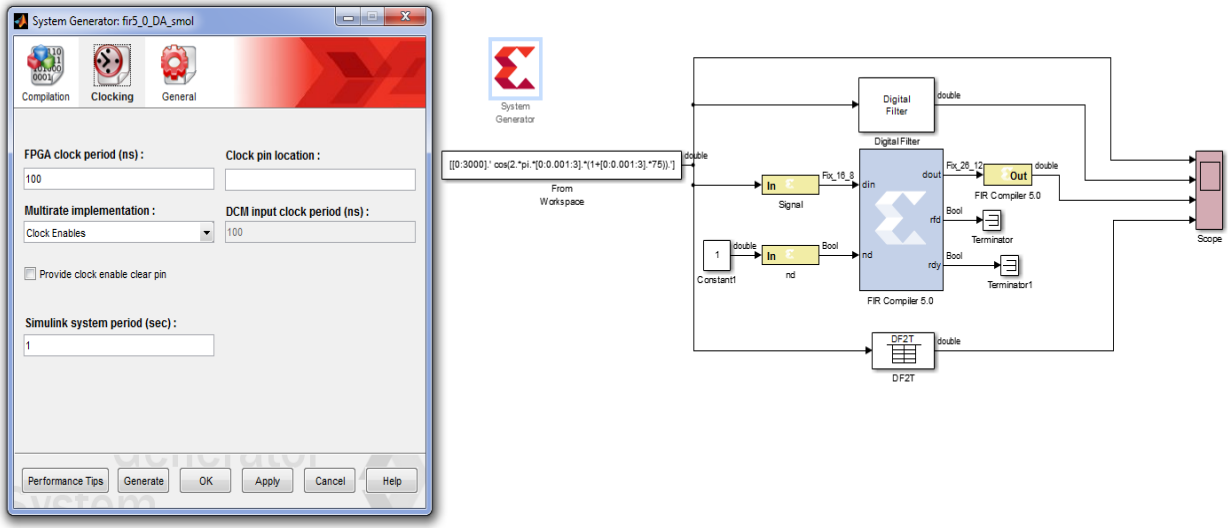
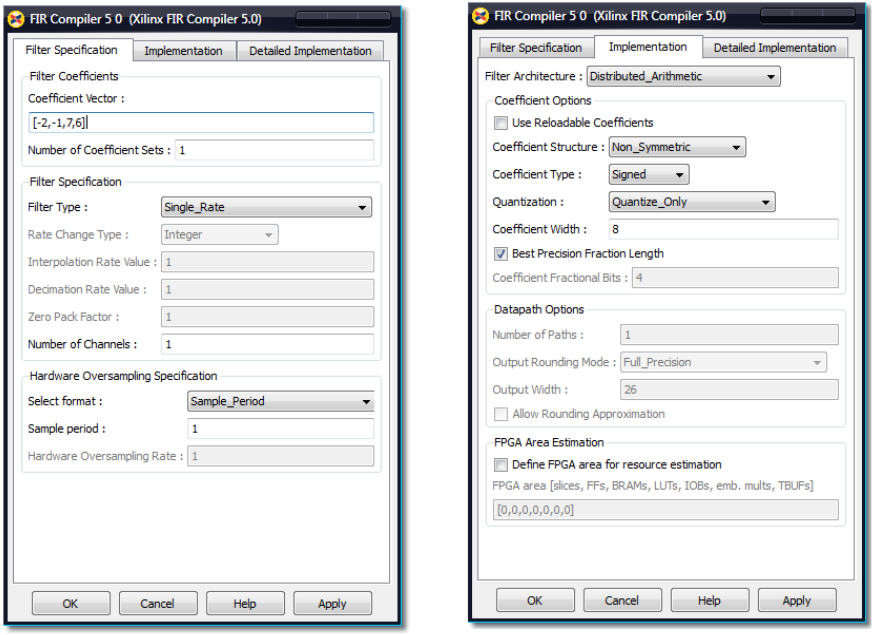


Рис. 5.1. (а) Задание частоты тактирования системы (фильтра) в САПР ISE (100 ns) и периода симуляции в Simulink (1 с); б) - модель КИХ-фильтра на четыре отвода с использованием блока FIR Compiler v5.0



а) б)

Рис. 5.2. 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

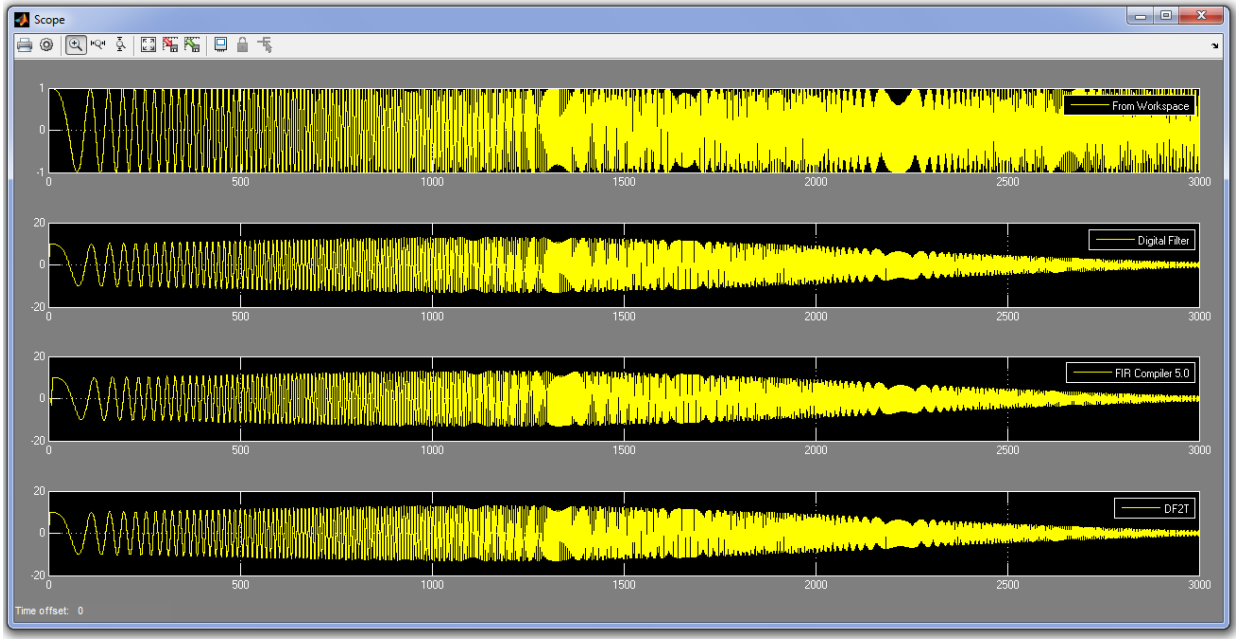


Рис. 5.3. Имитационное моделирование в системе Matlab/Simulink КИХ-фильтра на четыре отвода созданного с помощью блоков Digital Filter, FIR Compiler 5.0 и DF2T

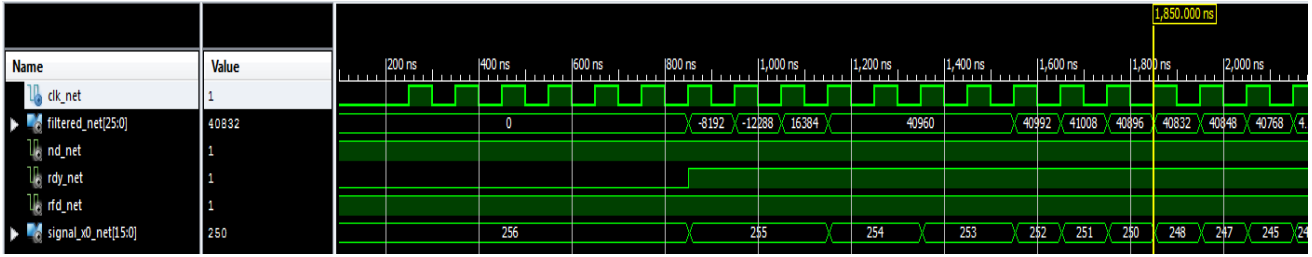


Рис. 5.4. Входной сигнал умножается на масштабный множитель 256, коэффициенты фильтра на 16. Период синхросигнала 100 нс

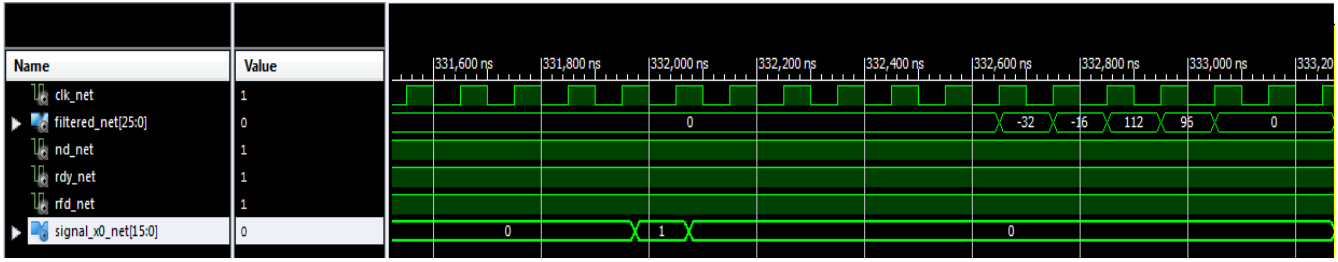


Рис. 5.5. Импульсная характеристика фильтра. Коэффициенты умножаются на 16

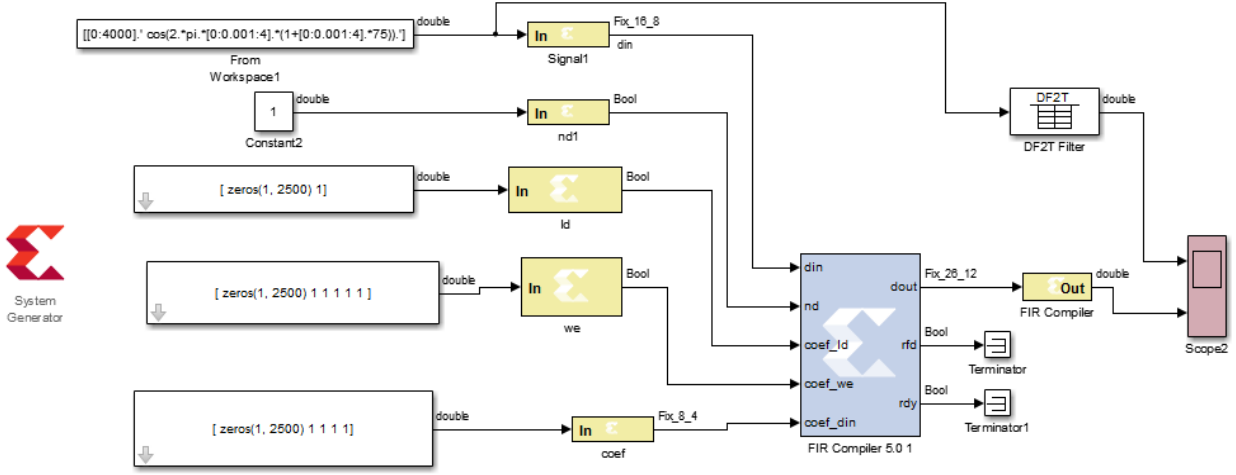


Рис. 5.6. Модель КИХ-фильтра на 4 отвода с перегружаемыми коэффициентами с использованием блока FIR Compiler v5.0

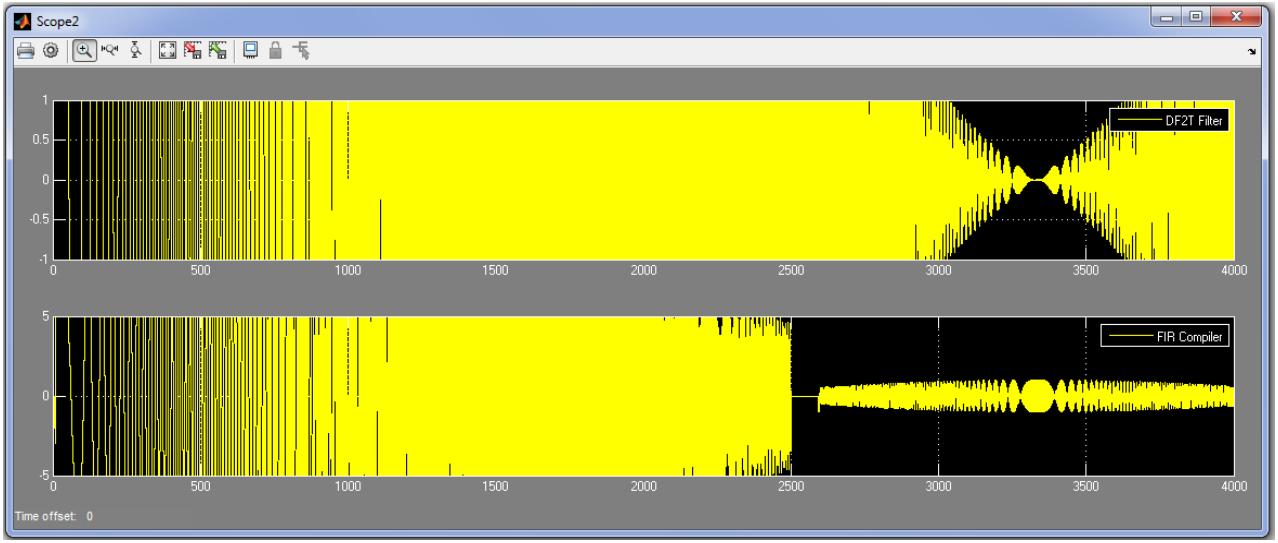


Рис. 5.7. При достижении 2500 отсчета происходит загрузка вектора значений коэффициентов [1 1 1 1] вместо [-2 -1 7 6]

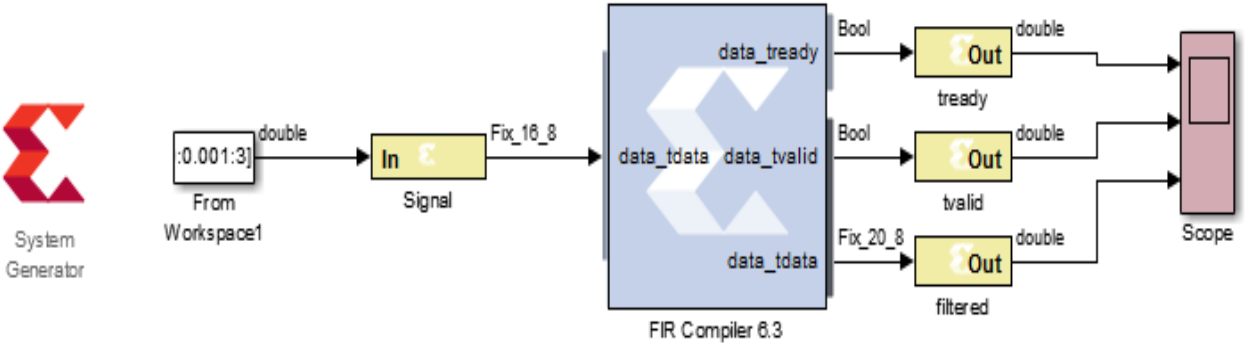
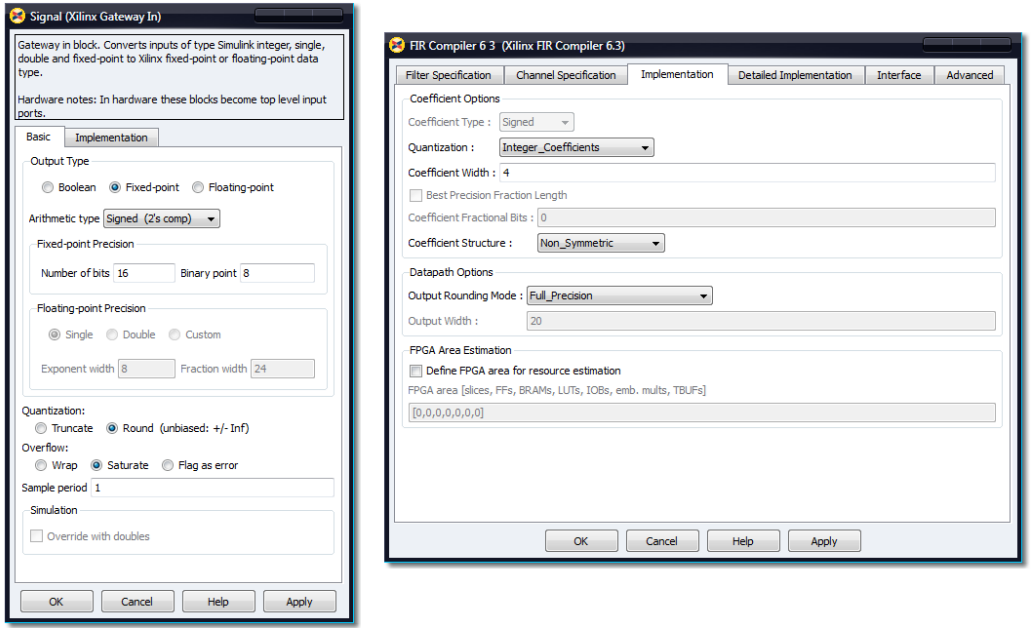


Рис. 5.8. Систолический КИХ-фильтр на четыре отвода с использованием блока FIR Compiler v6.3



а) б)

Рис. 5.9. 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

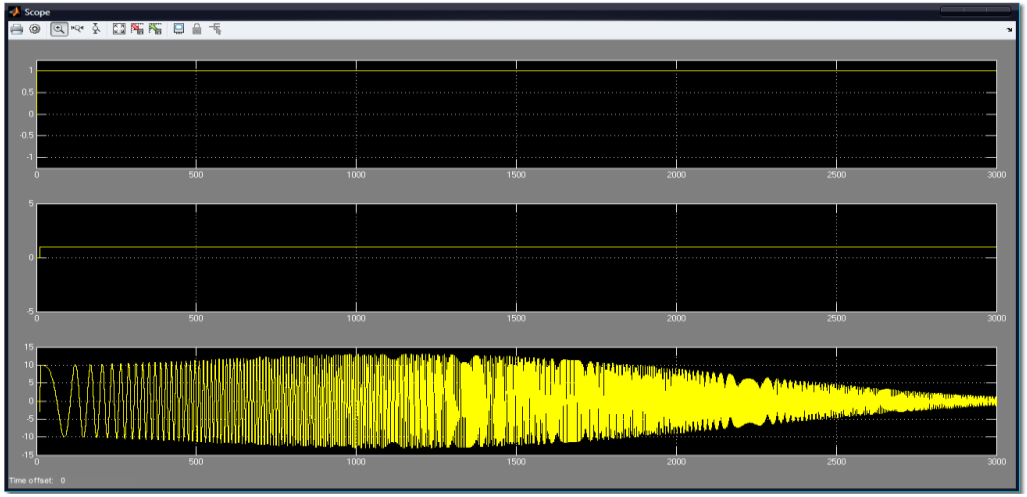


Рис. 5.10. Имитационное моделирование фильтра на четыре отвода созданного с помощью блока FIR Compiler 6.3

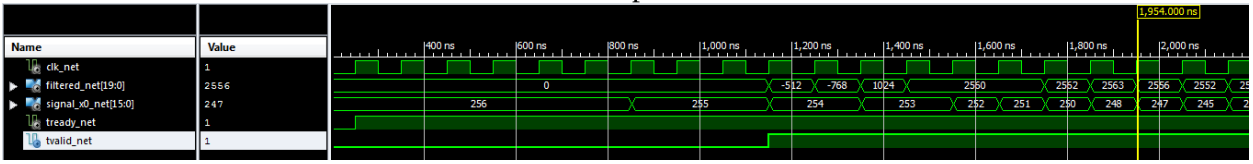


Рис. 5.11. Функциональное моделирование с использованием моделирующей программы на языке VHDL сгенерированной в автоматическом режиме. Входной сигнал умножается на 256, а коэффициенты фильтра не масштабируются

5.2. Проектирование последовательных КИХ-фильтров со структурой MAC-блоков в системе Xilinx System Generator САПР ISE Design Suite

Рассмотрим проектирование КИХ-фильтра (рис. 5.12) 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 (двух портовое 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. 5.13 показаны настройки функционального блока источника случайного сигнала. В поле Sample Time необходимо указать период дискретизации сигнала $1/F_S = 1/44100$.

Библиотека DSP Xilinx blockset содержит встроенную графическую среду для синтеза и анализа фильтров FDATool представленную в виде одноименного блока помеченного маркером X (рис. 5.14, *а*). Использование специальной функции xlfda_numerator(‘FDATool’) позволяет импортировать рассчитанные коэффициенты фильтра (числитель передаточной функции) непосредственно в блок MAC Based FIR. Коэффициенты можно так же посмотреть в системе Matlab с помощью команды xlfda_numerator('FDATool').

Блок MAC Based FIR является параметризованным (шесть переменных). Двойным кликом по блоку можно задать значения параметров в определенных полях (рис. 5.14, *б*). 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. 5.15.

Максимальное значение коэффициента составляет 0.3022 (определяется командой: 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] (рис. 5.16).

На рис. 5.17, *а* показана структурная схема КИХ-фильтра на 43 отвода с использованием одного MAC-блока и блочной двух портовой 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.

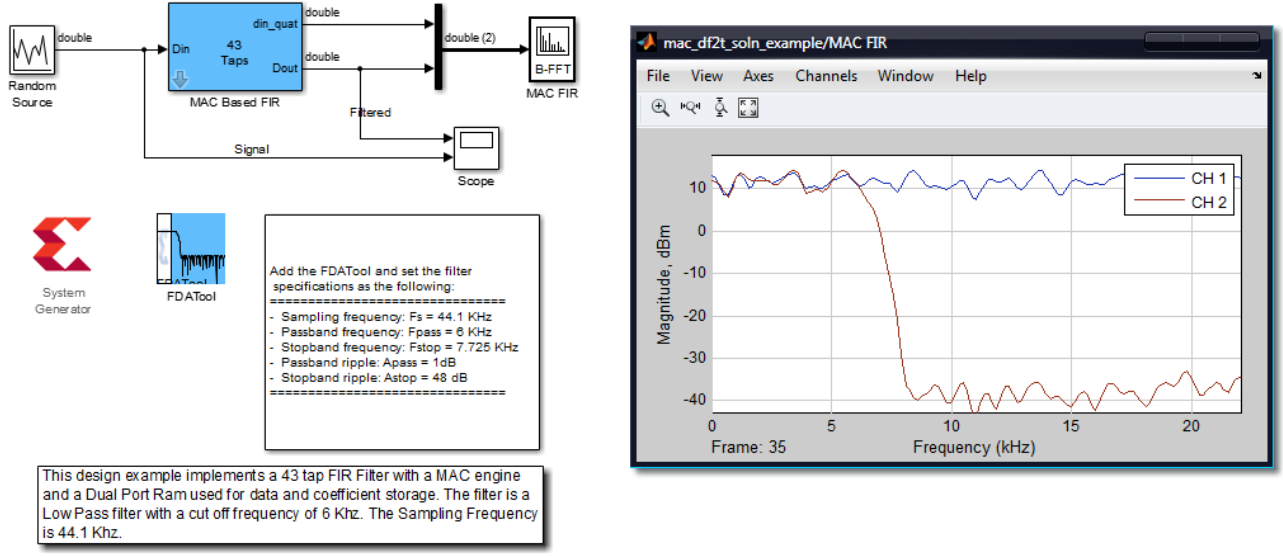


Рис. 5.12. Имитационная модель КИХ-фильтра на 43 отвода с возможностью вычисления коэффициентов по заданной спецификации с помощью графической среды FDATool и АЧХ. На вход фильтра подключен источник случайного сигнала

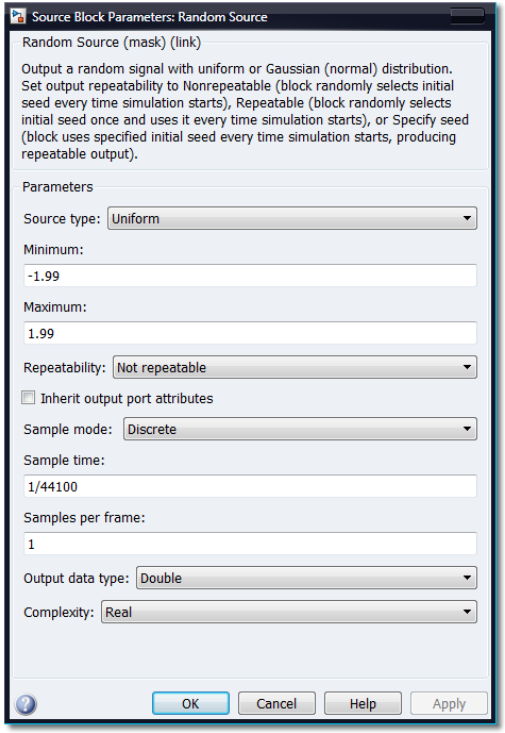
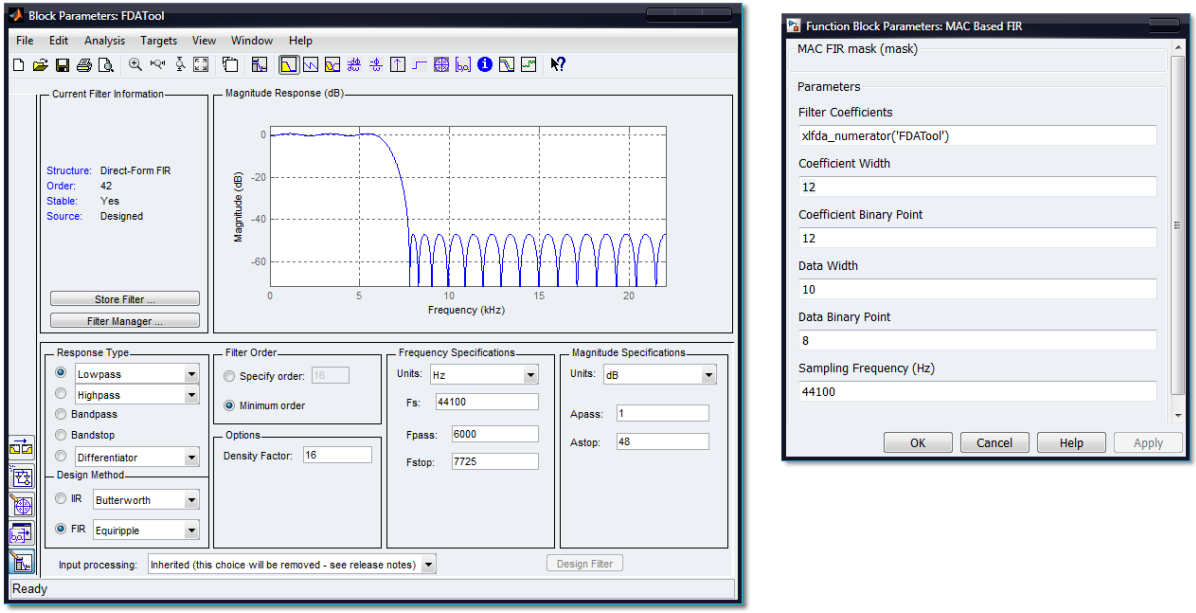


Рис. 5.13. Источник случайного сигнала с периодом дискретизации 1/44100



а) б)

Рис. 5.14. Блок FDATool (а) и маска с параметрами блока MAC Based FIR (б). Значения коэффициентов фильтра вычисляются с помощью команды xlfda_numerator(‘FDATool’)

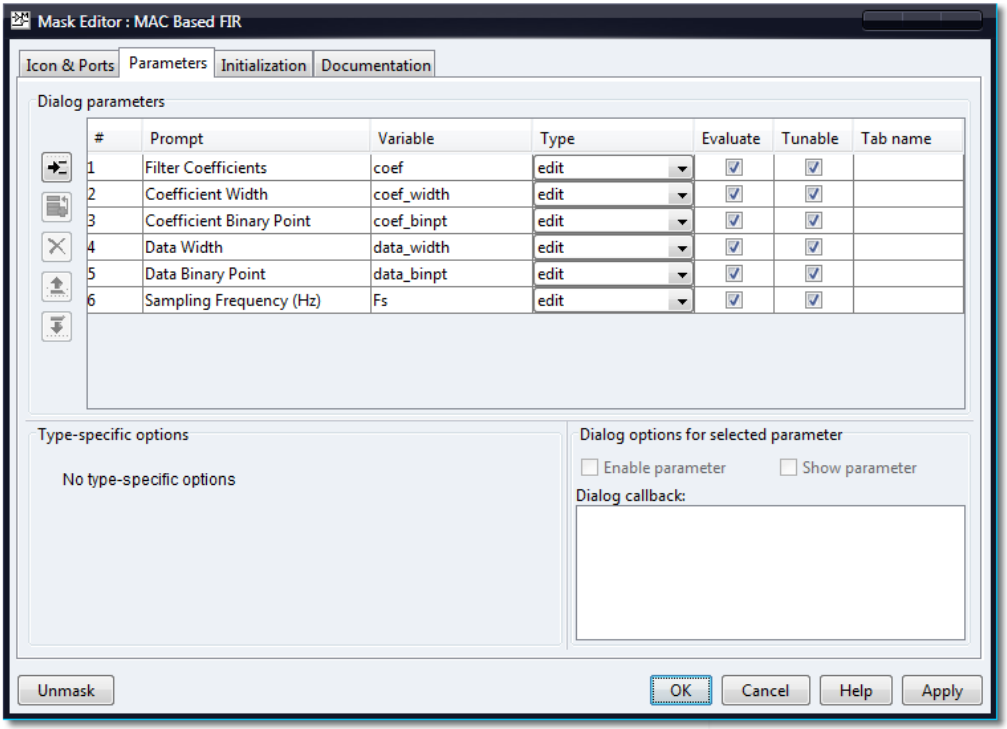


Рис. 5.15. Маска с параметрами функционального блока MAC Based FIR

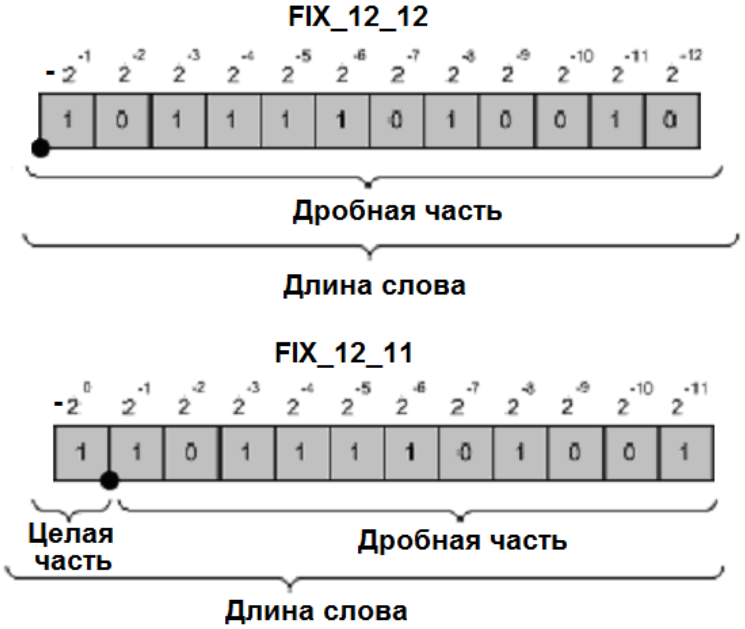


Рис. 5.16. Представление чисел в формате с фиксированной запятой в System Generator

Емкость первого банка памяти предназначенного для хранения отсчетов составляет 43 строки (адреса 0 - 42) на 10 столбцов (FIX_10_8) или 43 10-разрядных слов. Циклический буфер имитирует работу линии задержки фильтра (рис. 5.17, *а*). Второй банк предназначен для хранения коэффициентов фильтра емкостью 43 строки (адреса 43 - 85) на 12 столбцов (FIX_12_12) или 43 12-разрядных слов.

На рис. 5.18 показана структурная схема MAC-фильтра с применением библиотек System Generator. Схема предназначена для размещения в логические ресурсы ПЛИС. Основные функциональные блоки: Memory (двух портовая 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 в и считываются из порта A (ОЗУ), коэффициенты считываются из порта B (рис. 5.19). Процессом записи и считывания управляет автомат (блок Address Control) (рис. 5.20).

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.

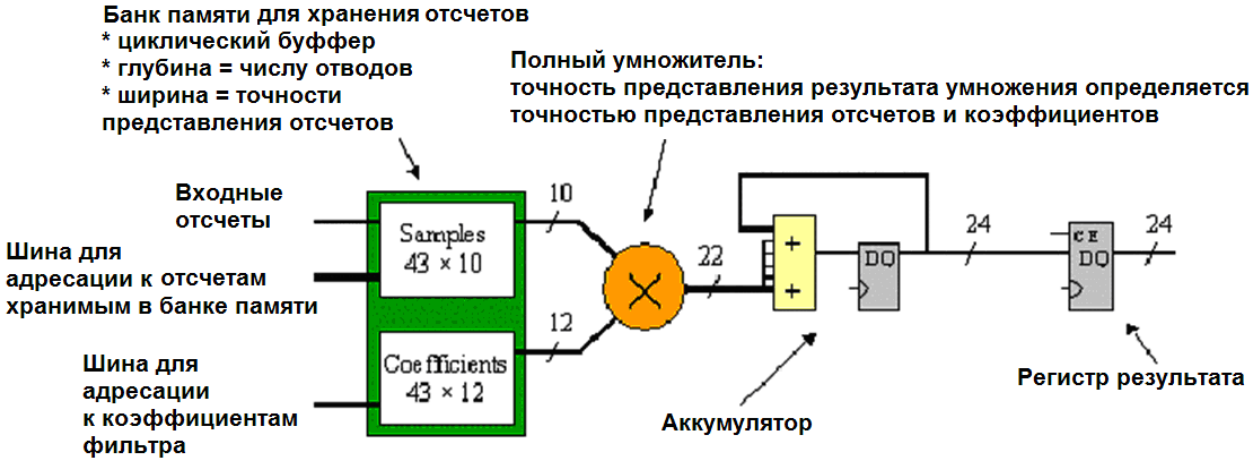


Рис. 5.17. Структурная схема КИХ-фильтра на 43 отвода с использованием одного MAC-блока и блочной памяти разбитой на два банка, один из которых используется в виде циклического буфера для хранения и считывания отсчетов, а другой – для хранения коэффициентов фильтра

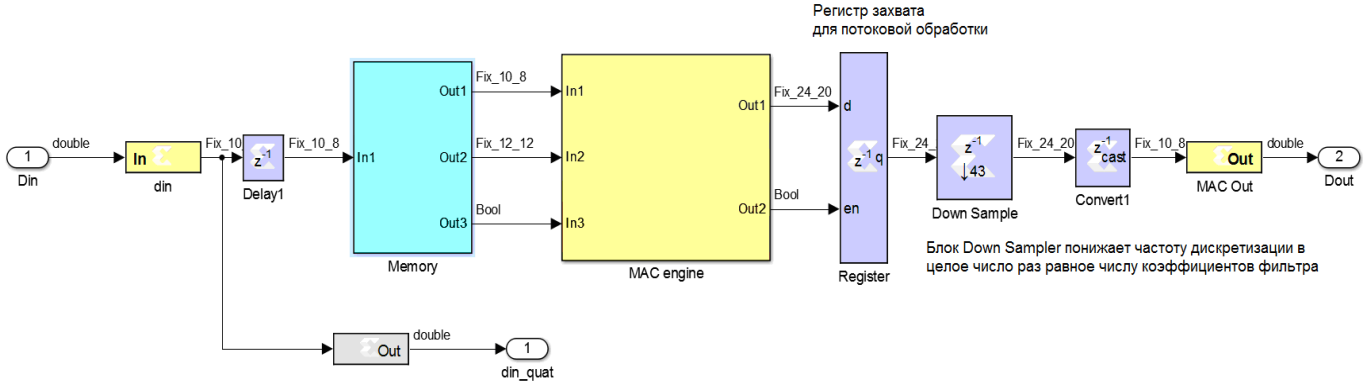


Рис. 5.18. Структурная схема MAC-фильтра с применением библиотек Xilinx System Generator

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. 5.21) инициализируется вектором значений с применением следующей команды [zeros(1, length(coef)) (coef)] (рис.5.22). Блок памяти имеет латентность - единица.

На рис. 5.21 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) выполняющего 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. 5.23 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.

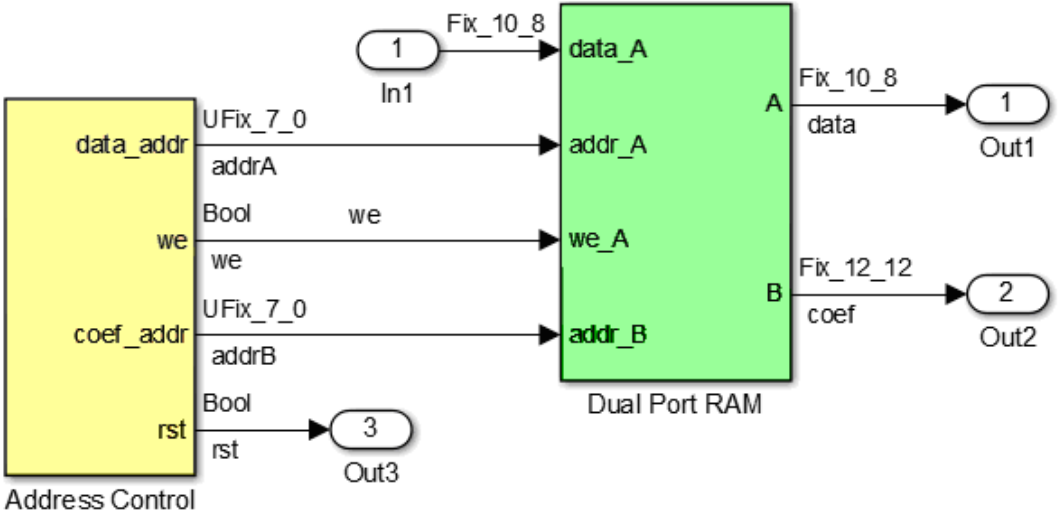


Рис. 5.19. Двух портовая память с управляющим автоматом

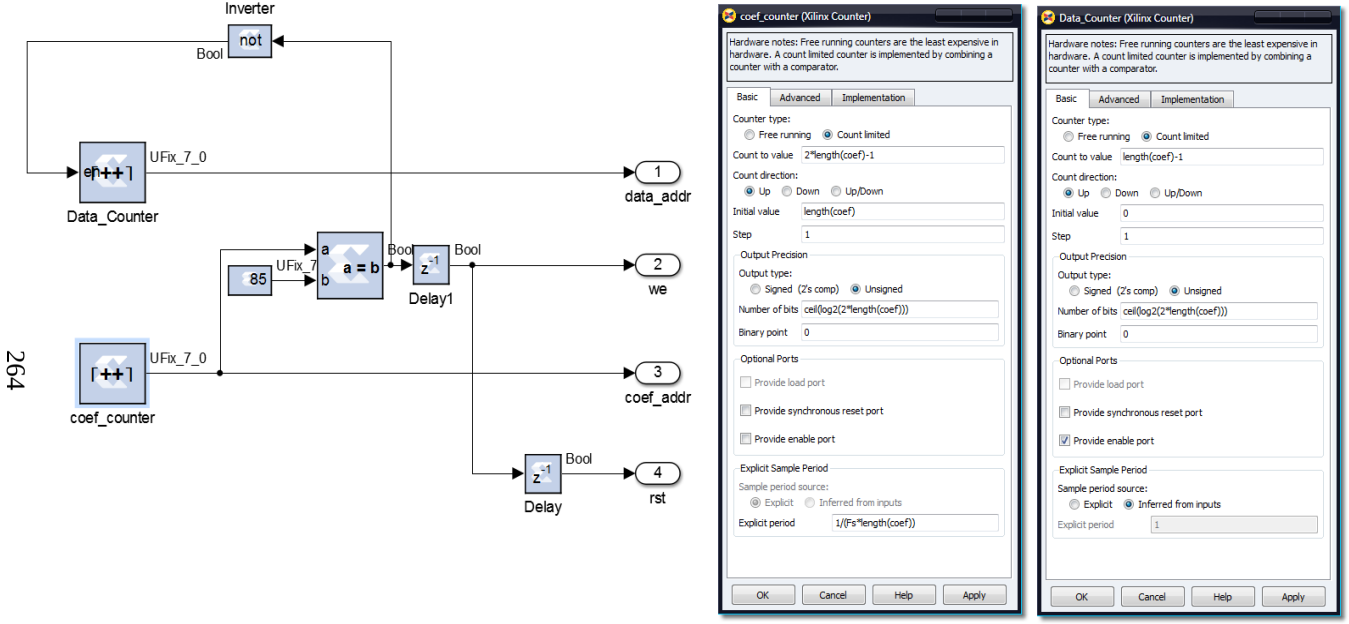


Рис. 5.20. Управляющий автомат двух портовой памяти и настройки блоков счетчиков coef_counter и Data_Counter

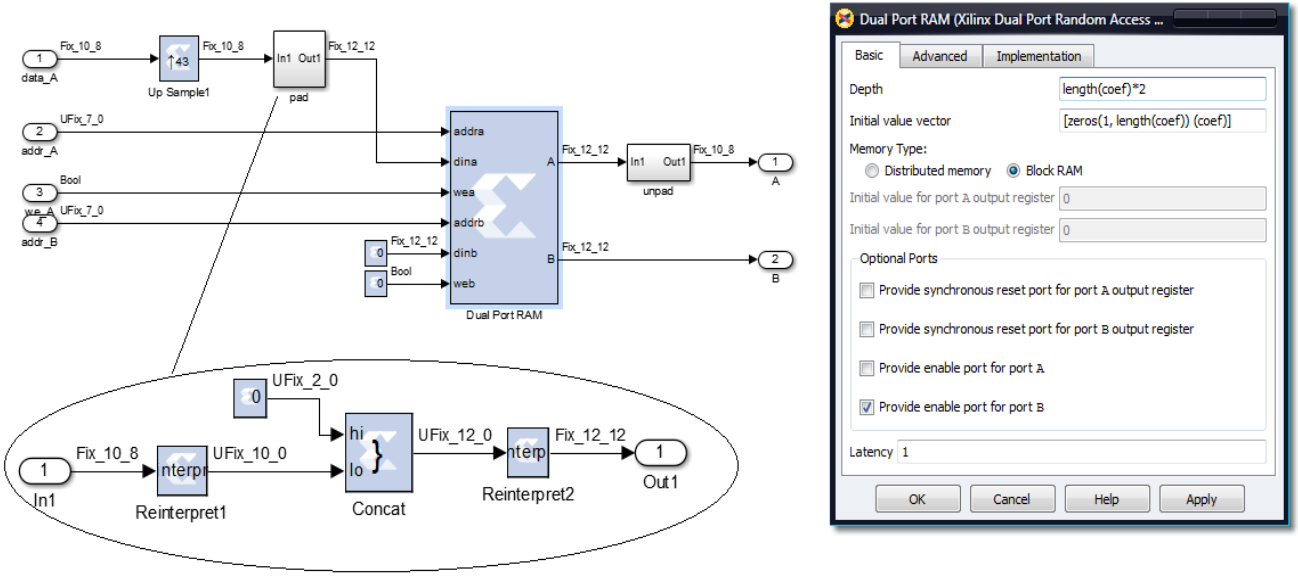


Рис. 5.21. Двух портовая память на основе блочной памяти ПЛИС и настройки блока Xilinx Dual Random Access

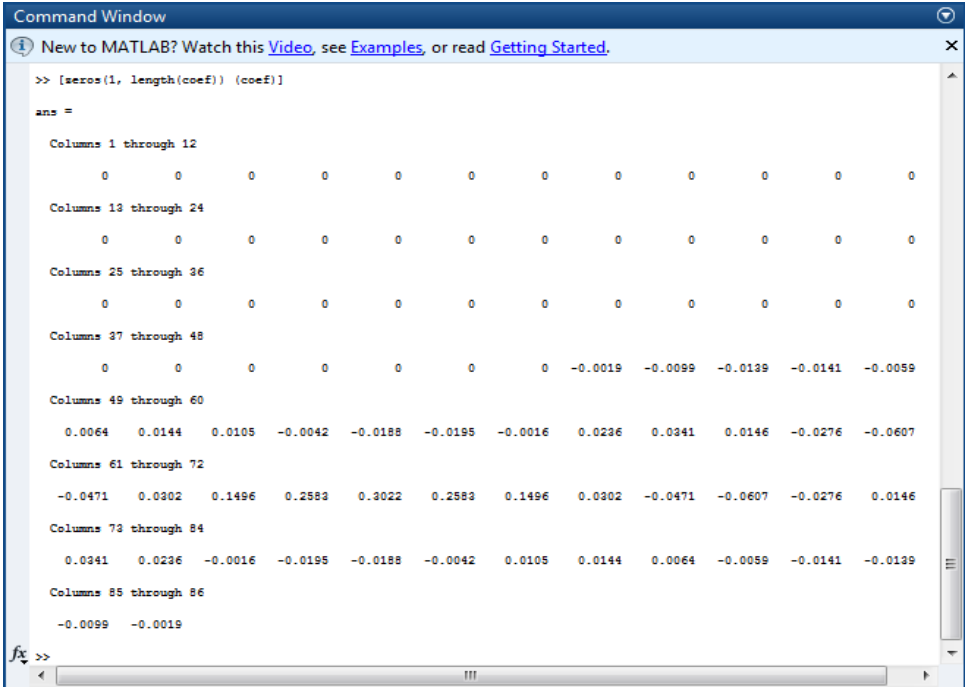


Рис. 5.22. Вектор инициализации блочной памяти ПЛИС (коэффициенты КИХ-фильтра симметричны)

Умножитель имеет латентность три для согласования с реальной латентностью равной трем характерной для встроенных умножителей в ПЛИС Xilinx.

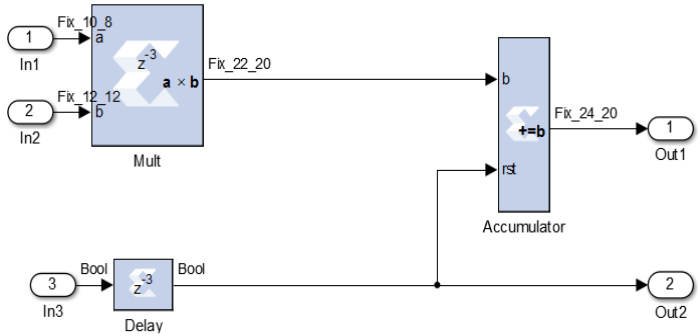


Рис. 5.23. Умножитель и аккумулятор

На рис. 5.24 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. 5.25 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 показано на рис. 5.26. 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. 5.27). 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.5.1.

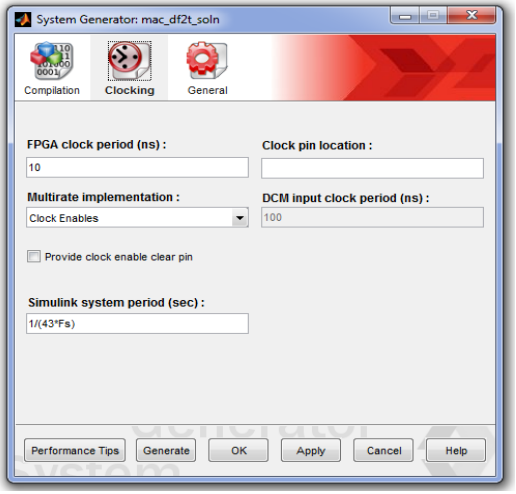


Рис. 5.24. Настройка периода симуляции в Simulink

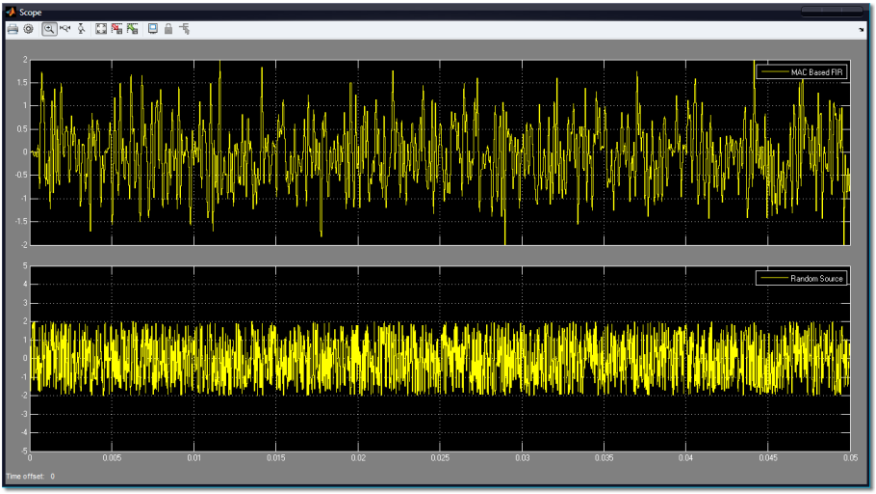
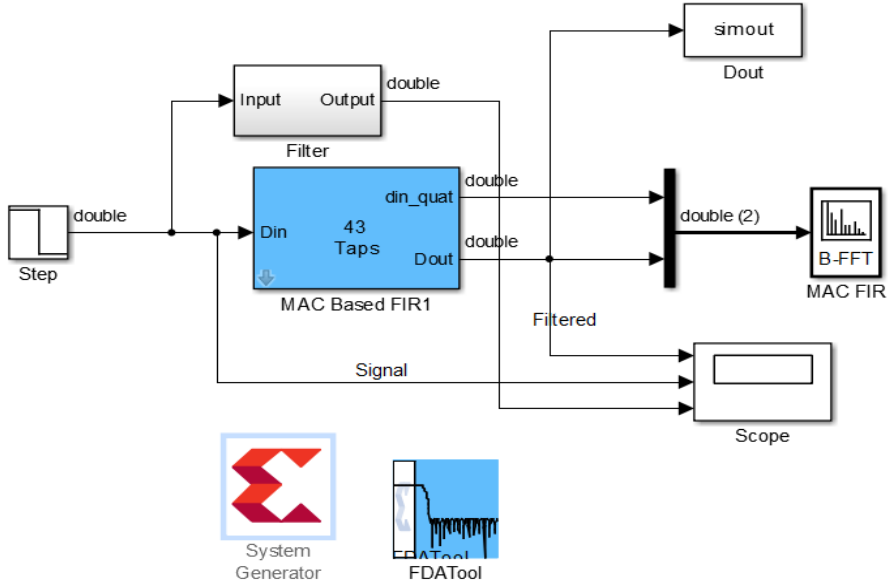
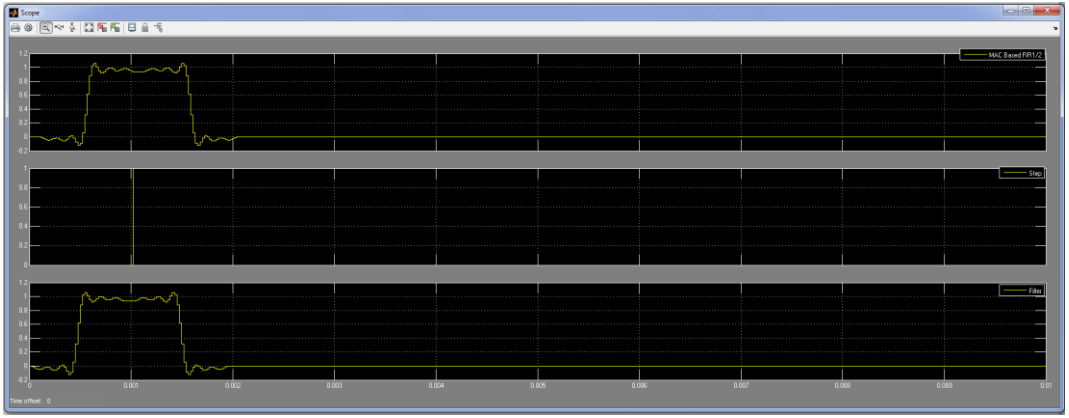


Рис. 5.25. Имитационное моделирование в системе Matlab/Simulink КИХ-фильтра на 34 отвода с использованием одного MAC-блока и блочной памяти



а)

Рис. 5.26. а) Имитационная модель КИХ-фильтра (на вход подается единичный импульс) и моделирование отклика КИХ-фильтра (б)



б)

Рис. 5.26. а) Имитационная модель КИХ-фильтра (на вход подается единичный импульс) и моделирование отклика КИХ-фильтра (б) (продолжение)

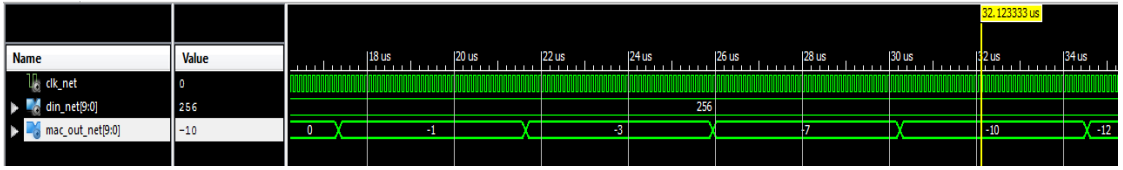


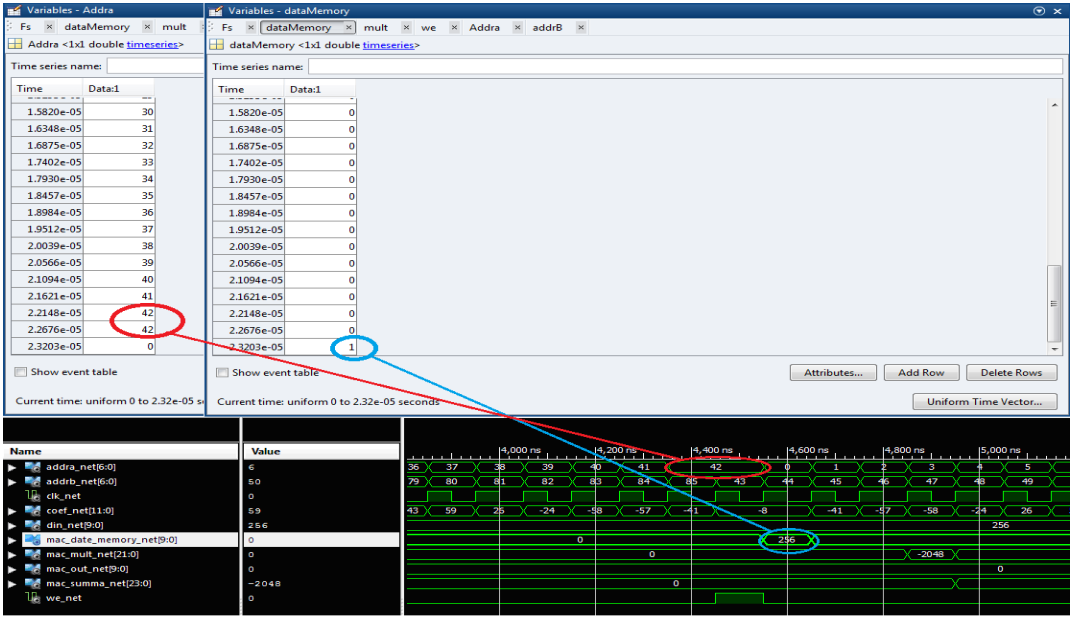
Рис. 5.27. Функциональное моделирование с использованием моделирующей программы на языке VHDL сгенерированной в автоматическом режиме. Входной сигнал (единичный импульс) и выходной сигнал умножаются на 256

Таблица 5.1

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

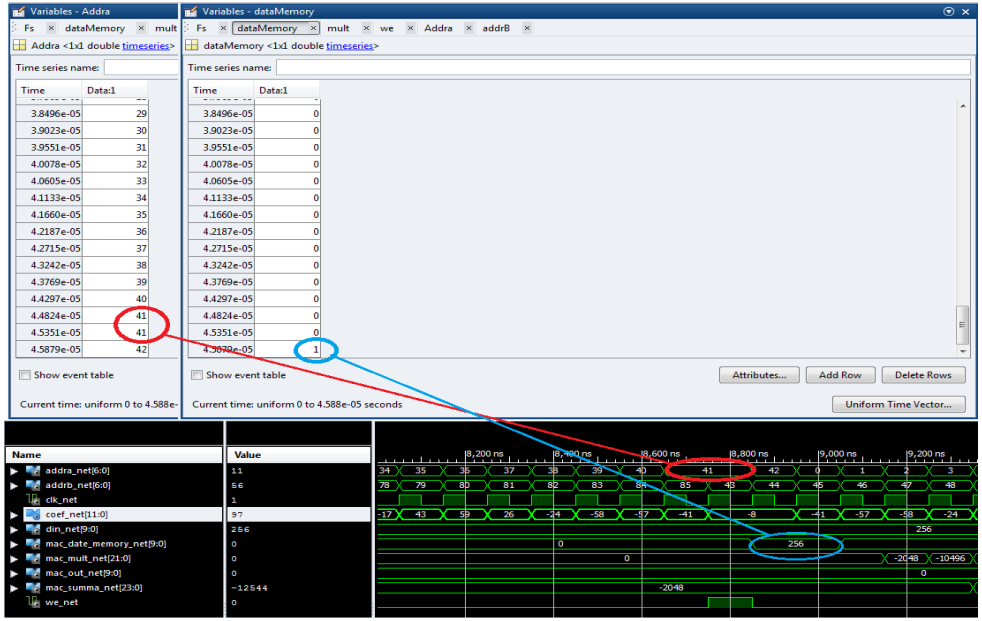
Значения отклика в формате double	Значения отклика в формате FIX_10_8 (значения отклика * на масштабный множитель 256)	Коэффициенты фильтра, в формате double	Коэффициенты фильтра в формате FIX_12_12 (коэф. фильтра * на масштабный множитель 4096)
-0,00195;	-1	-0.0019	-8
-0,01196;	-3	-0.099	-41
-0,02588;	-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. 5.28), тем самым происходит двойная адресация к строке 42 (два такта синхроимпульса). Это обеспечивает запись десятичного числа один 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. 5.28, *а*). Через последующие 43 такта десятичная единица будет записана в строку с адресом 42 (рис. 5.28, *б*). Эта единица (256) выше описанным 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. 5.26, блок Step). Как только импульс упадет в ноль, им последовательно будет заполнен первый банк ОЗУ.



а)

Рис. 5.28. Результаты расчетов в пошаговом режиме в системе Matlab и временные диаграммы в ISE Design Suite поясняющие принцип работы циклического буфера



б)

Рис. 5.28. Результаты расчетов в пошаговом режиме в системе Matlab и временные диаграммы в ISE Design Suite поясняющие принцип работы циклического буфера (продолжение)

5.3. Проектирование последовательных КИХ-фильтров в системе Xilinx System Generator с применением библиотеки Reference BlockSet/DSP

Рассмотрим проектирование двух последовательных КИХ-фильтров на четыре 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 двух портовой 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 16x1. 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 Matlab 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: $y = C_0x_0 + C_1x_1 + C_2x_2 + C_3x_3$ 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 и 0 т.д.

Правильные значения на выходе фильтра: 10, -1, -40, -10, 26, 6 и 0 т.д.

Воспользуемся готовым блоком n-tap MAC FIR filter (рис. 5.29). Проект разместим в базис ПЛИС серии Spartan-6 xa6slx4-3tqg144. Настроим маску с параметрами блока n-tap MAC FIR filter (рис. 5.30, *а*). Коэффициенты фильтра и входной сигнал, подлежащий фильтрации представляются в формате с фиксированной запятой с 4-битной точностью. Число разрядов дробной части – ноль.

На рис. 5.30, *б* показано задание частоты тактирования фильтра в САПР ISE (100 ns) и периода симуляции в Simulink (1/4 с). На рис. 5.30, *в* и рис. 5.31 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. Рис. 5.32 показывает имитационное моделирование в системе Matlab/Simulink КИХ-фильтра на четыре 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. 5.33. Разработаем имитационную модель управления адресуемым сдвиговым регистром с помощью 2-разрядного суммирующего счетчика (рис. 5.34). Результаты имитационного моделирования показывают, что латентность блока ASR составляет четыре такта синхрочастоты (рис. 5.35). Латентность блока может быть задана в ручную или вычислена исходя из максимальной разрядности адресного порта. Подключение задержки на один такт на вход ASR-блока приводит к увеличению латентности на восемь тактов синхрочастоты (рис. 5.36 и рис. 5.37). А подключение задержки на один такт на его вход и задержки на два такта на выходе составит для такой конструкции десять тактов синхрочастоты (рис. 5.38 и рис. 5.39).

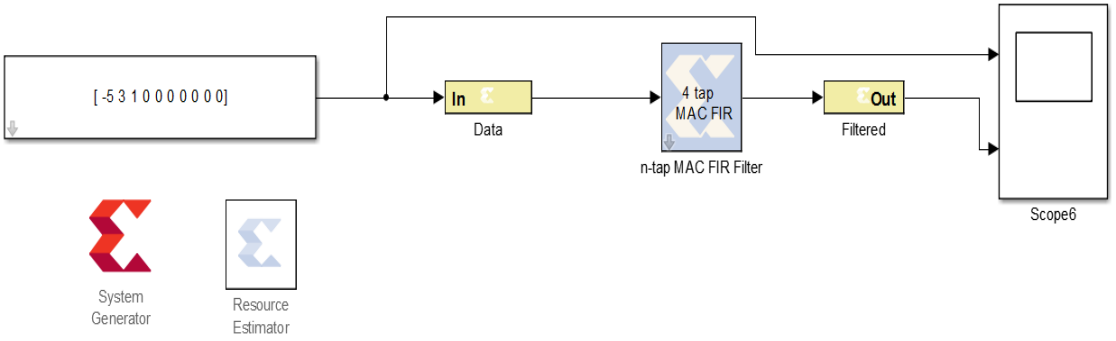
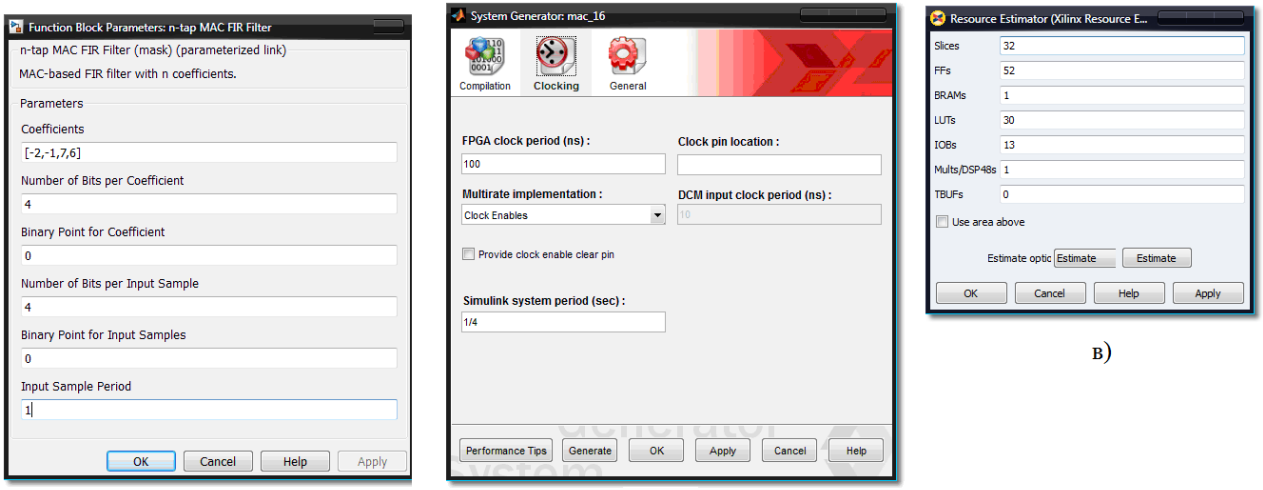


Рис. 5.29. Имитационная модель КИХ-фильтра на четыре отвода для реализации в базисе ПЛИС серии Spartan-6 xa6slx4-3tqg144 на основе n-tap MAC FIR filter



а) б) в)

Рис. 5.30. 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. 5.31. Оценка ресурсов проекта в САПР ISE по коду языка VHDL извлеченному в автоматическом режиме с помощью маркера System Generator

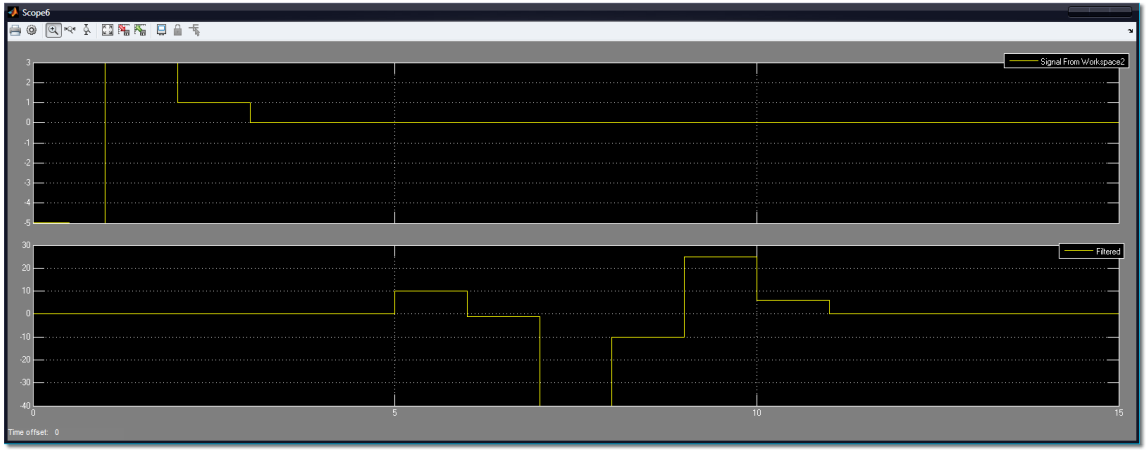


Рис. 5.32. Имитационное моделирование в системе Matlab/Simulink КИХ-фильтра на четыре отвода

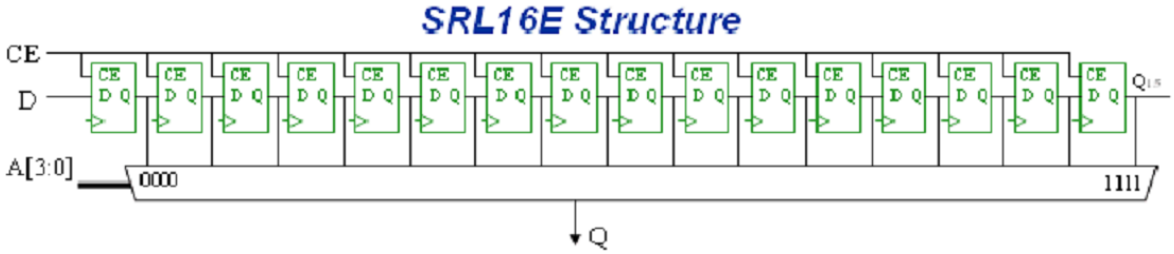


Рис. 5.33. Структурная схема адресуемого сдвигового регистра на примитиве SRL16E

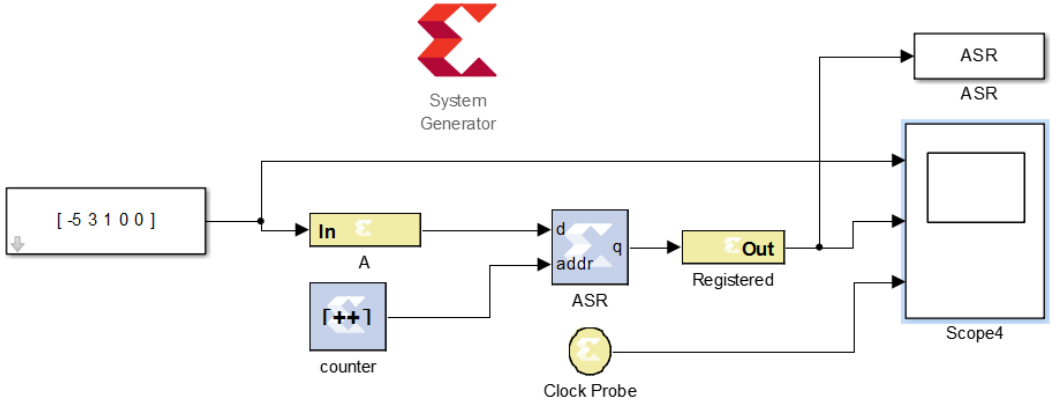


Рис. 5.34. Управление адресуемым сдвиговым регистром (функциональный блок ASR) с помощью счетчика

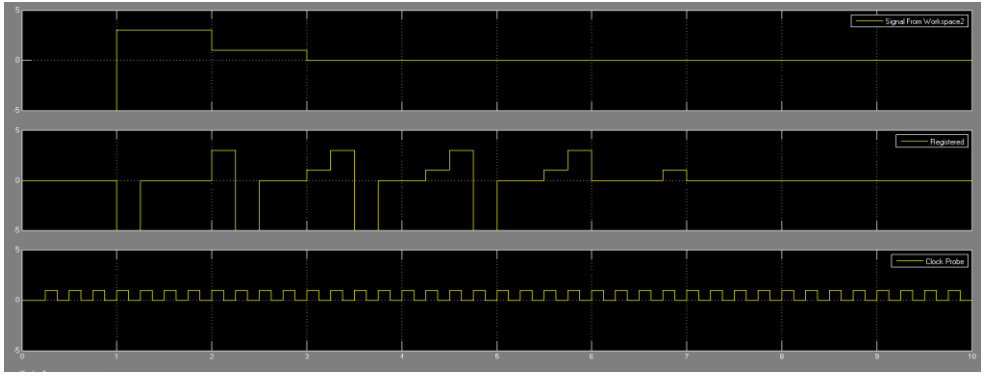


Рис. 5.35. Латентность ASR-блока 4 такта синхрочастоты

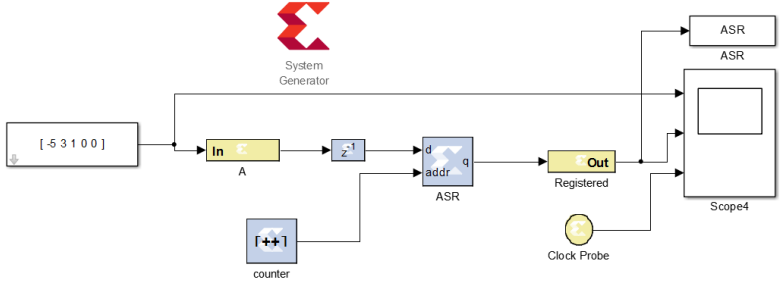


Рис. 5.36. Исследование латентности ASR-блока при подключении задержки на один такт на его вход

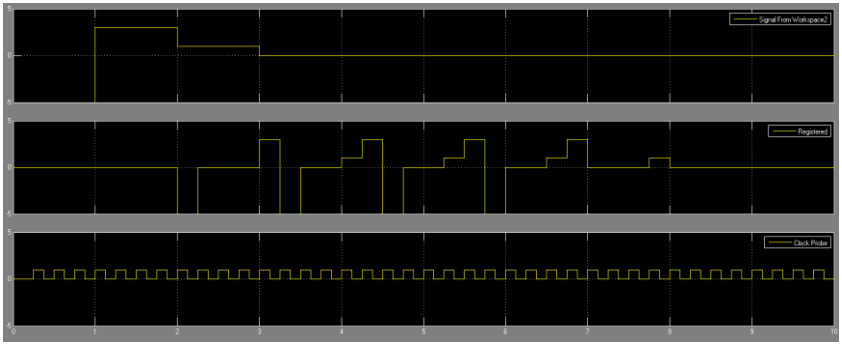


Рис. 5.37. Латентность ASR-блока при подключении задержки на его вход составляет восемь тактов синхрочастоты

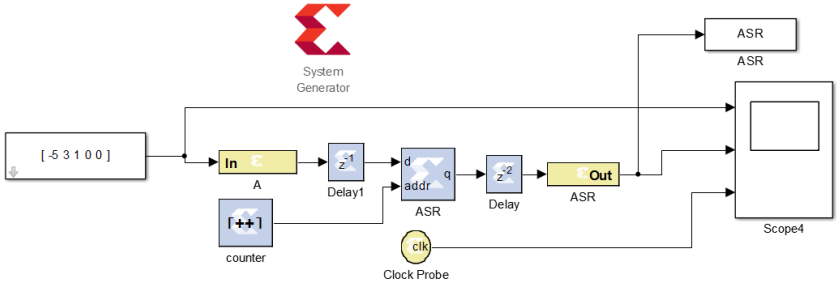


Рис. 5.38. Исследование латентности ASR-блока при подключении задержки на один такт на его вход и задержки на два такта на выход

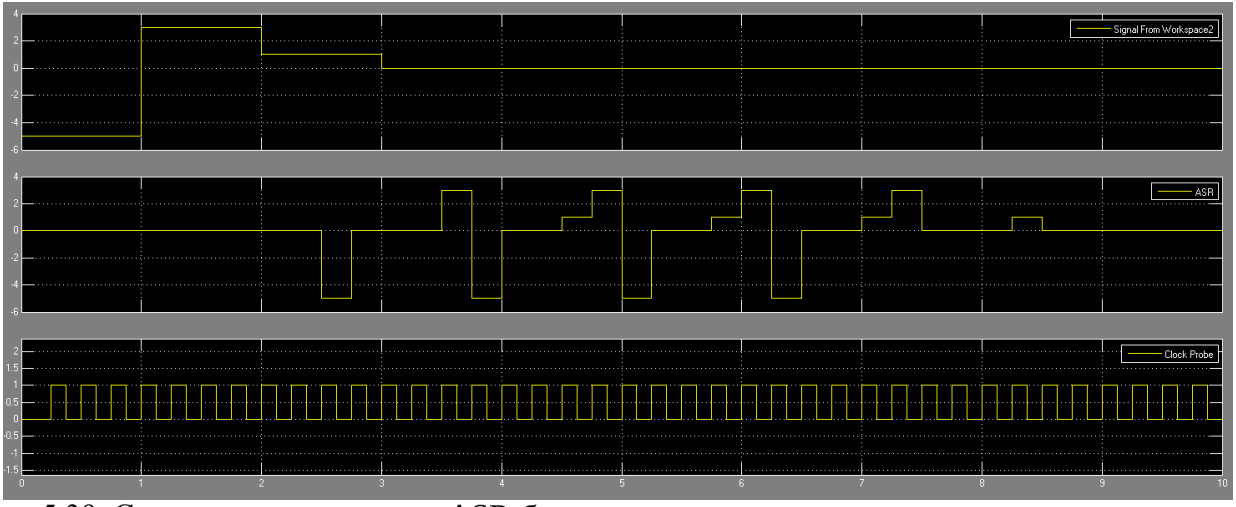


Рис. 5.39. Суммарная латентность ASR-блока при подключении задержки на один такт на его вход и задержки на два такта на выход составит десять тактов синхрочастоты

Рассмотрим структуру блоков входящих в состав функционального блока n-tap MAC FIR filter. Структурная схема управляющего автомата (функциональный блок Address Control) показана на рис. 5.40, а на рис. 5.41 представлена структурная схема функционального блока Memory. Выход суммирующего 2-разрядного счетчика подключен к адресным шинам блоков ASR и ROM (ПЗУ). Емкость ПЗУ составляет четыре строки по четыре бита, инициализируется вектором значений [-2 -1 7 6]. На рис. 5.42 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 пять 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. 5.43). 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. 5.44 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. 5.45 показаны значения сигналов на выходах блоков. Сигнал A (Data_RAM) на выходе линии задержки имеет латентность десять тактов синхрочастоты (рис. 5.45, *а*). Коэффициенты фильтра, извлекаемые из ПЗУ имеют латентность два такта синхрочастоты (сигнал B, он же Coef_ram) (рис. 5.45, *б*). Сигнал res_mac имеет латентность – пять (рис. 5.45, *в*).

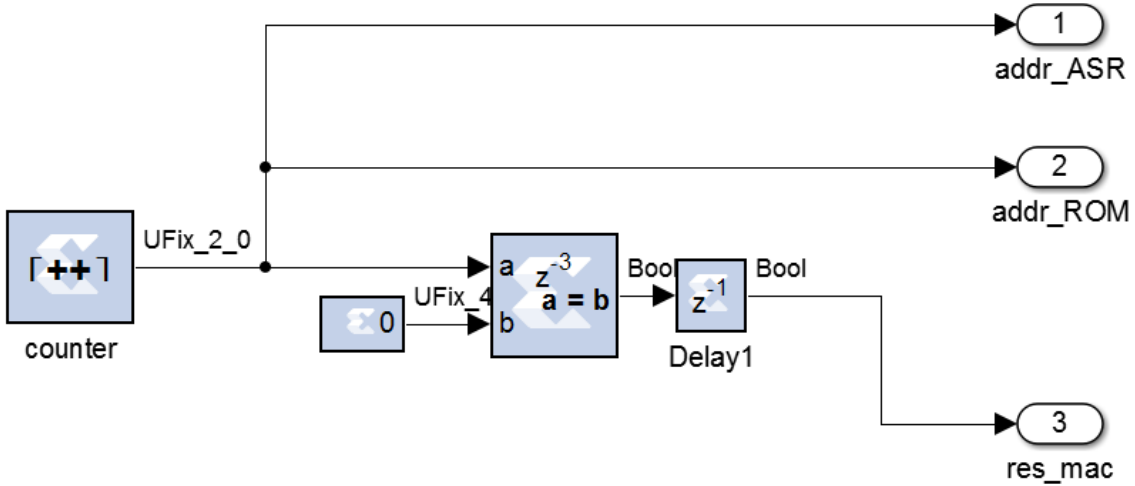


Рис. 5.40. Структурная схема управляющего автомата (функциональный блок Address Control)

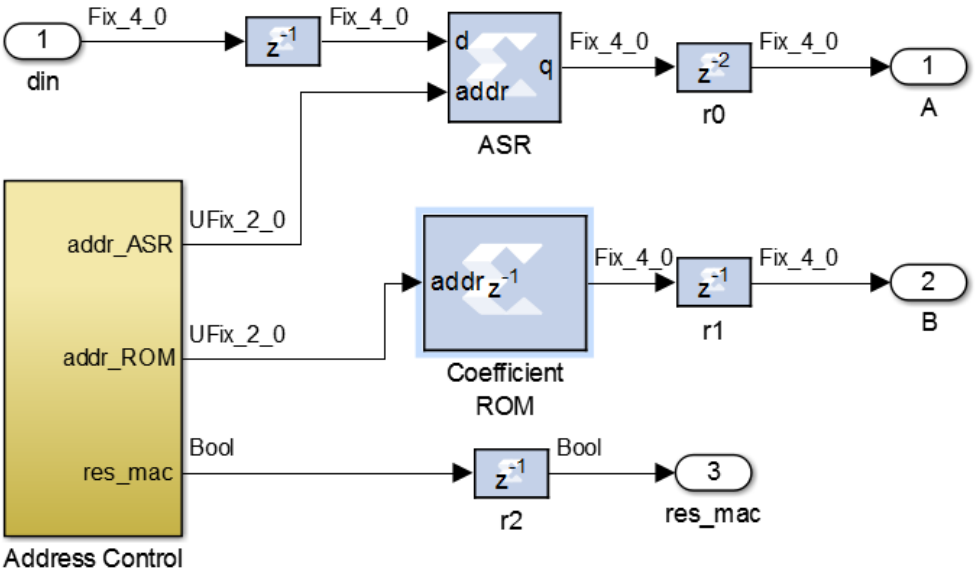


Рис. 5.41. Структурная схема функционального блока Memory

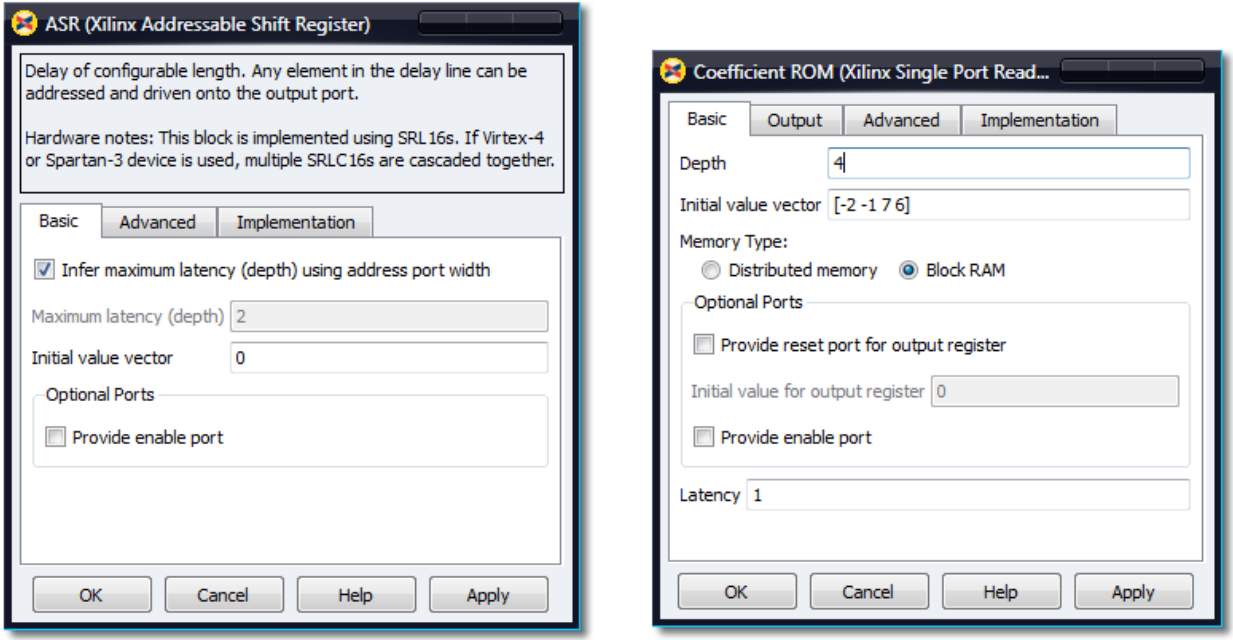


Рис. 5.42. Настройки ASR-блока и ROM-блока

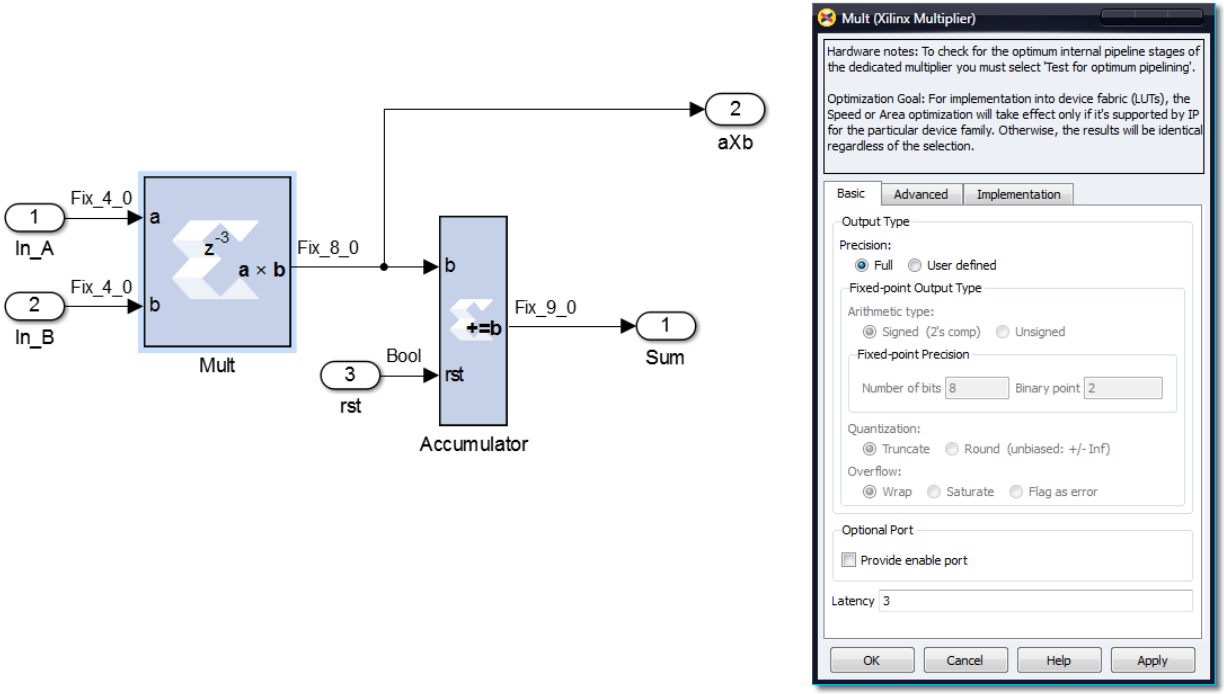


Рис. 5.43. Блок умножения с накоплением (функциональный блок MAC engine) и настройки умножителя (Mult-блок)

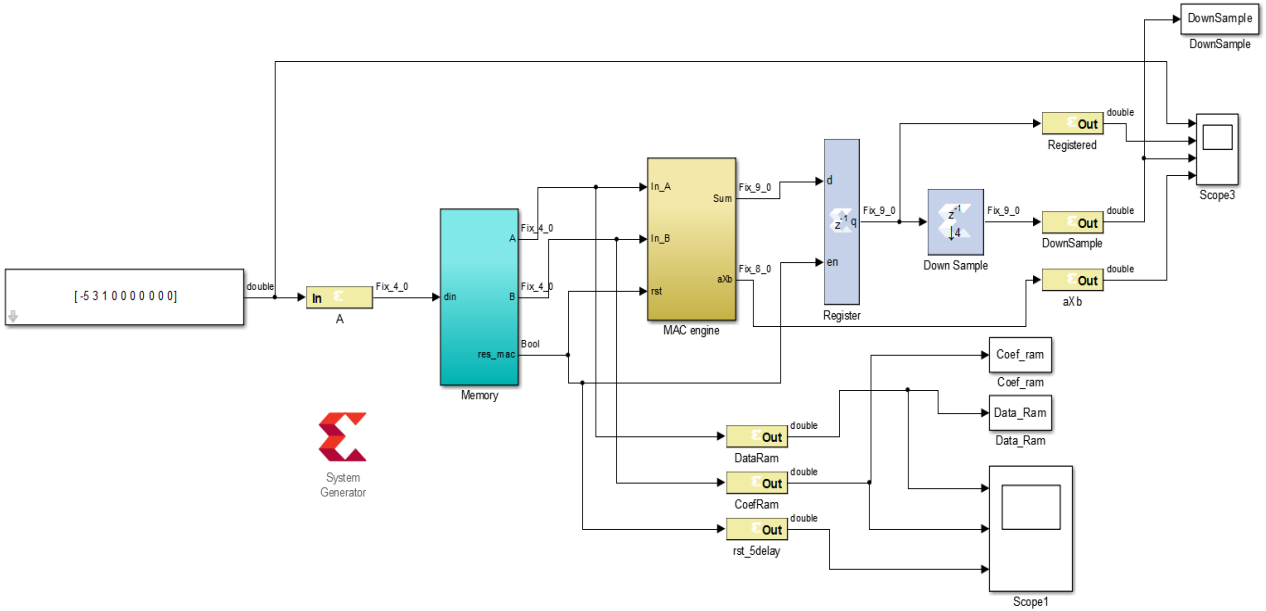


Рис. 5.44. Имитационная модель КИХ-фильтра на четыре отвода на основе функционального блока n-tap MAC FIR filter

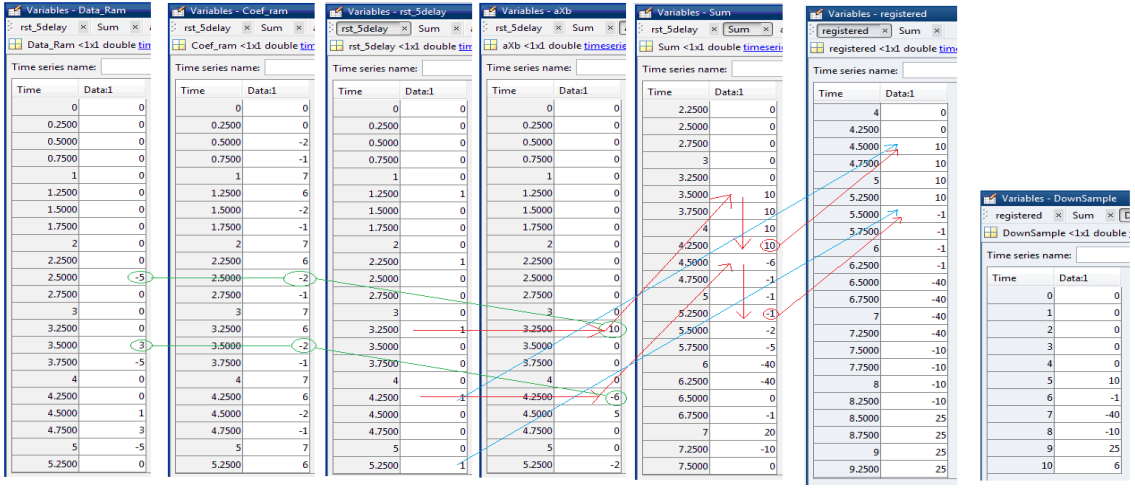


Рис. 5.45. Представление значений сигналов в главном окне системы Matlab в дискретные моменты времени: а) - сигнал A; б) коэффициенты фильтра (сигнал B, он же Coef_ram); в) сигнал res_mac, обеспечивающий правильность загрузки новых значений произведения aXb в аккумулятор MAC-блока и последующих накоплений 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)

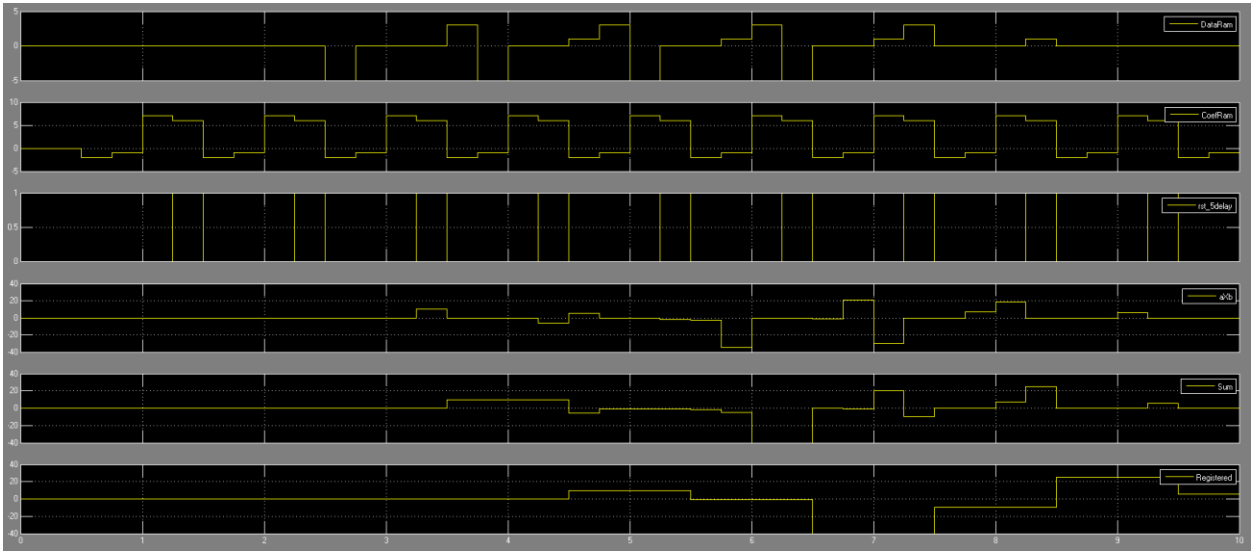


Рис. 5.46. Имитационное моделирование в системе Matlab/Simulink КИХ-фильтра на четыре отвода созданного на основе функционального блока n-tap MAC FIR filter. Сигналы расположенные согласно позициям а), б), в), г), д) и е) представленным на рис. 5.45

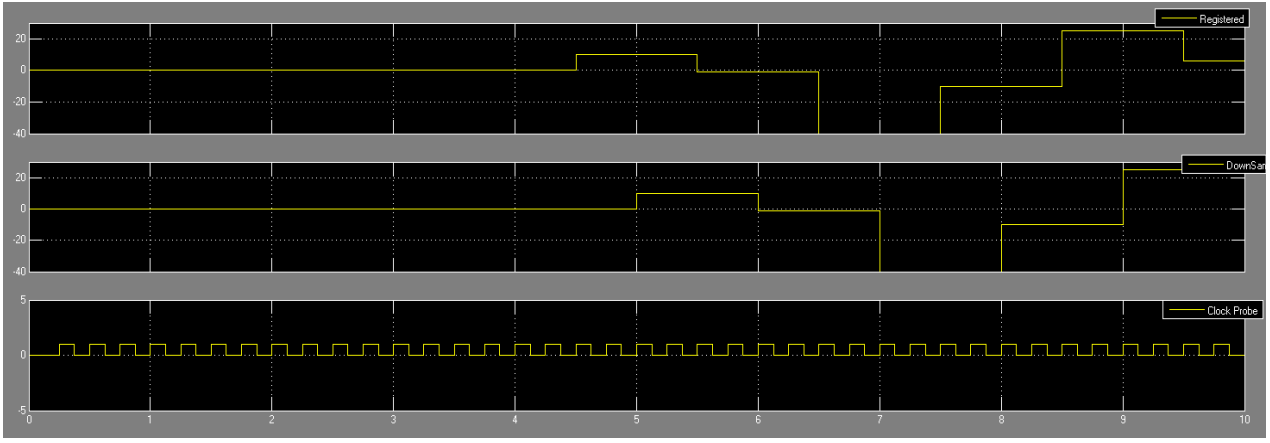


Рис. 5.47. Сигнал до и после операции децимации (позиции *е* и *к* на рис. 5.45)

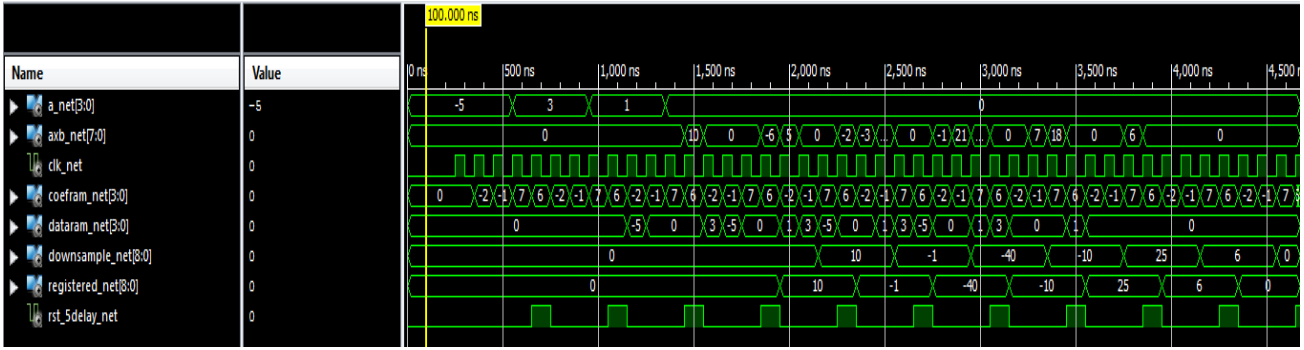


Рис. 5.48. Функциональное моделирование КИХ-фильтра на четыре отвода созданного на основе функционального блока n-tap MAC FIR filter с использованием моделирующей программы на языке VHDL сгенерированной в автоматическом режиме.
Обозначение сигналов согласно рис. 5.44

Сигнал aXb является текущим значением произведения сигналов A и B (рис. 5.45, *г*), а сигнал Sum представляет сумму значений произведений сигналов A и B накопленных в аккумуляторе (рис. 5.45, *д*). Сигналы aXb и Sum формируются в блоке MAC engine.

На рис.5.46 показаны сигналы расположенные согласно позициям а), б), в), г), д) и е) соответствующие рис.5.45, а на рис.5.47 представлен сигнал до и после операции децимации (позиции *е* и *к* на рис. 5.45). Зелеными (формирование произведения), красными (формирование суммы произведений) и синими (значения суммы произведений в регистре захвата) стрелками на рис. 5.45 обозначены информационные потоки при вычислениях. Правильные значения на выходе фильтра: 10, -1, -40, -10, 26, 6 и все последующие нулевые значения показаны на рис. 5.45, *г*.

Результаты функционального моделирования в САПР Xilinx ISE с использованием испытательного стенда, код которого получен в автоматическом режиме, показаны на рис. 5.48. Сравнивая результаты имитационного моделирования на рис. 5.46 и функционального представленного на рис. 5.48 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 четыре отвода для реализации в базисе ПЛИС серии Spartan-6 xa6slx4-3tqg144 адаптированная к нашей задаче показана на рис. 5.49.

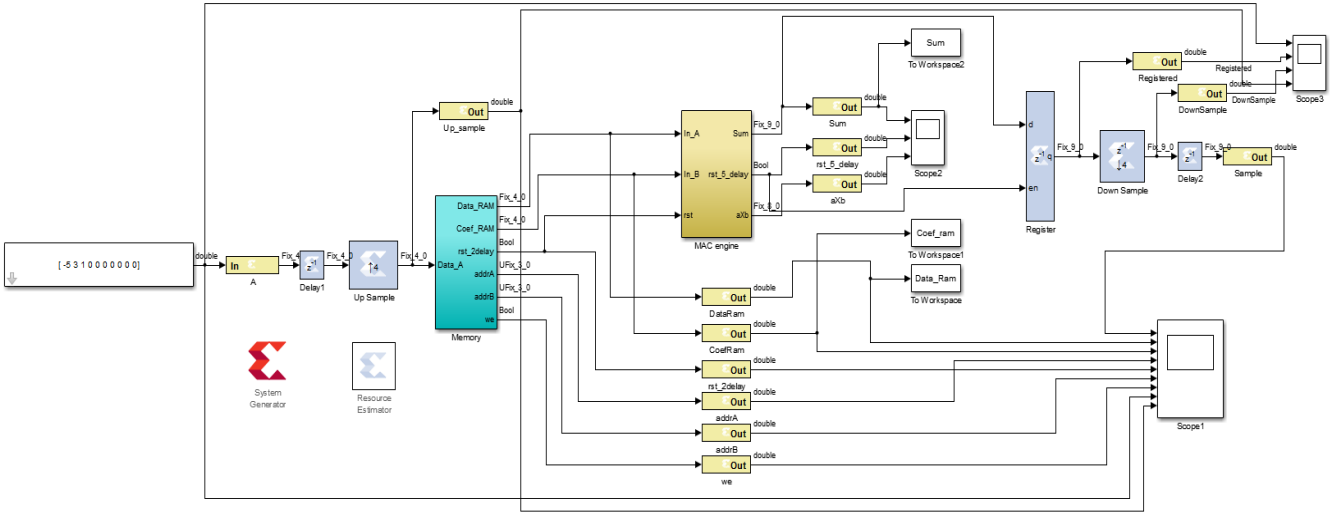


Рис. 5.49. Имитационная модель КИХ-фильтра на четыре отвода для реализации в базисе ПЛИС серии Spartan-6 xa6slx4-3tqg144 на основе n-tap Dual Port Memory MAC FIR Filter

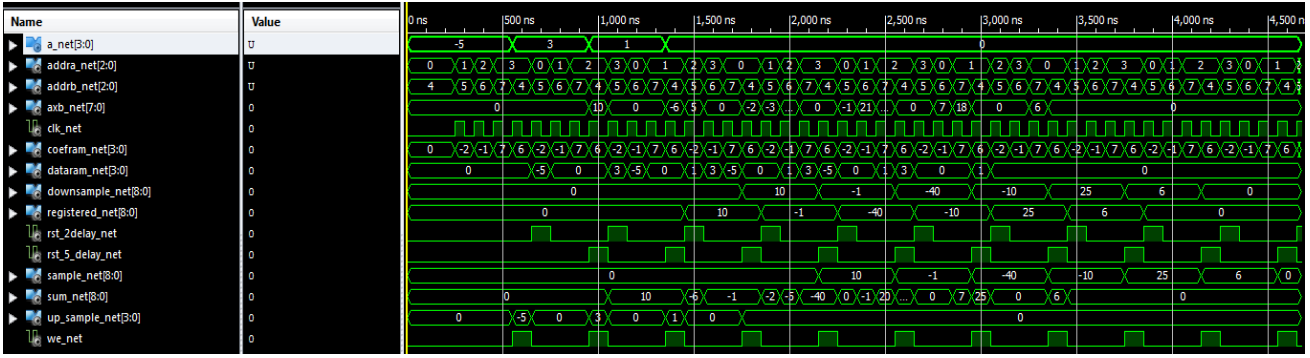


Рис. 5.50. Функциональное моделирование КИХ-фильтра на четыре отвода созданного на основе функционального блока n-tap Dual Port Memory MAC FIR Filter с использованием моделирующей программы на языке VHDL сгенерированной в автоматическом режиме

Таблица 5.2

Оценка ресурсов ПЛИС серии Spartan-6 xa6slx4-3tqg144 при реализации КИХ-фильтров на четыре 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 А настроен на режим чтение затем запись. Порт В – чтение.

На рис. 5.50 показано функциональное моделирование. Максимальная частота проекта по коду языка VHDL 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 таблице.

В Xilinx System Generator рассмотрено проектирование последовательных КИХ-фильтров на четыре 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. 5.2).

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. 5.2, 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 арифметике 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. ИСПОЛЬЗОВАНИЕ СИСТЕМЫ ВИЗУАЛЬНО-ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ MATLAB/SIMULINK ДЛЯ ПРОЕКТИРОВАНИЯ КИХ-ФИЛЬТРОВ В САПР QUARTUS II

6.1. Проектирование последовательных КИХ-фильтров в системе визуально-имитационного моделирования Matlab/Simulink с использованием Altera DSP Builder

Пакет Altera DSP Builder ver. 12.1 работает в связке с САПР Quartus II ver.12.1 (сборка 177) 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 Matlab/Simulink обеспечивают высокоуровневое оптимизированное VHDL-представление проектов с автоматическим компилированием в ПЛИС Xilinx и Altera с последующим созданием испытательных стендов.

Рассмотрим разработку КИХ-фильтр на четыре отвода с использованием одного блока умножения и накопления (1 MAC-блок) в Altera DSP Builder: $y = C_0x_0 + C_1x_1 + C_2x_2 + C_3x_3$. Предположим что, коэффициенты фильтра известны $C_0 = -2$, $C_1 = -1$, $C_2 = 7$ и $C_3 = 6$. Такие фильтры еще называют MAC-фильтры. За основу проектируемого КИХ-фильтра возьмем имитационную модель с именем FIR_MAC32.mdl КИХ-фильтра на 32 отвода из справочной системы САПР Quartus II по адресу:

quartus\dsp_builder\DesignExamples\Demos\Filters\Mac32

и адаптируем ее под свои задачи.

Рассмотрим кратко основные структуры КИХ-фильтров с использованием MAC-блоков в Xilinx System Generator. В главе 5 с использованием параметризованных функциональных блоков n-tap MAC FIR filter и n-tap Dual Port Memory MAC FIR Filter показано проектирование

односкоростных последовательных КИХ-фильтров на четыре отвода.

Функциональный блок n-tap MAC FIR filter реализуется на базе адресуемого сдвигового регистра на ячейках конфигурационной памяти ПЛИС выполняющего роль линии задержки и блочной памяти в режиме ПЗУ для хранения коэффициентов фильтра. Такая опция доступна только для ПЛИС фирмы Xilinx.

Функциональный блок n-tap Dual Port Memory MAC FIR Filter реализуется на базе блочной памяти в режиме ОЗУ. Двух портовая память логически разбивается 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 А настроен на режим чтение затем запись. Порт В – чтение.

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. КИХ-фильтры на основе функциональных блоков n-tap- и n-tap Dual Port Memory MAC FIR из за того что используют различные типы памяти отличаются друг от друга управляющими автоматами (рис. 6.1 и рис.6.2). Функциональное моделирование КИХ-фильтра на четыре отвода созданного на основе функционального блока n-tap MAC FIR filter показано на рис. 6.3. На рис. 6.4 представлено функциональное моделирование КИХ-фильтра на четыре отвода созданного на основе функционального блока n-tap Dual Port Memory MAC FIR Filter. Профильтрованные данные на выходе фильтров обновляются через четыре такта синхроимпульса.

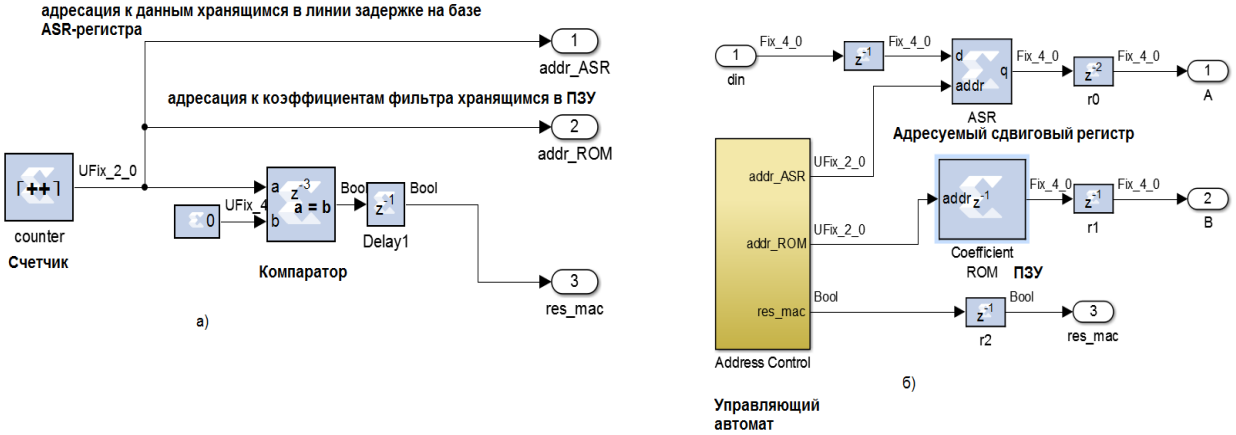


Рис. 6.1. Структурная схема управляющего автомата КИХ-фильтра (а) и его подключение к адресуемому сдвиговому регистру (ASR) и блочной памяти в режиме ПЗУ (функциональный блок n-tap MAC FIR filter) (б)

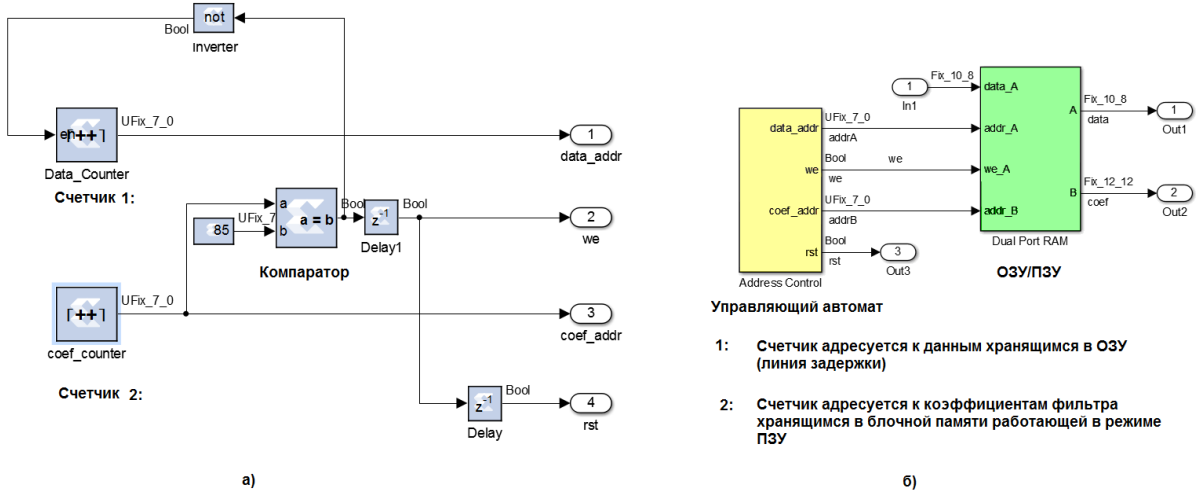


Рис. 6.2. Структурная схема управляющего автомата КИХ-фильтра (а) и его подключение к блочной памяти в режиме ОЗУ (функциональный блок n-tap Dual Port Memory MAC FIR Filter)

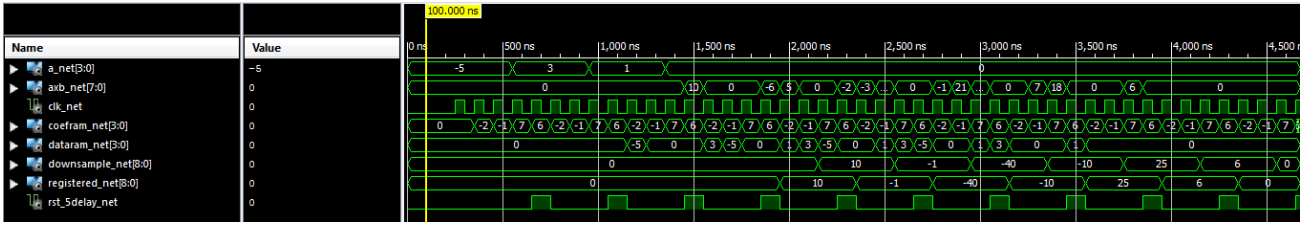


Рис. 6.3. Функциональное моделирование КИХ-фильтра на четыре отвода созданного на основе функционального блока n-tap MAC FIR filter

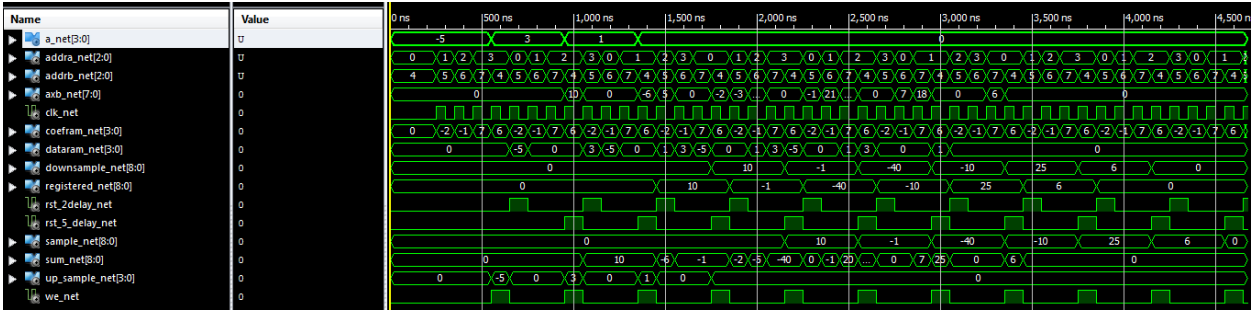


Рис. 6.4. Функциональное моделирование КИХ-фильтра на четыре отвода созданного на основе функционального блока n-tap Dual Port Memory MAC FIR Filter

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. На рис. 6.5 представлена имитационная модель последовательного КИХ-фильтра на четыре 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: счетчик (управляющий автомат); линия задержки на базе двух портовой памяти (ОЗУ) и вспомогательного мультиплексора; LUT (блок памяти в режиме ПЗУ) для хранения коэффициентов фильтра; MAC-блок и регистр для хранения результата; компилятор сигналов; генератор испытательных стендов; формирование 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 Matlab/Simulink 0.25 с. Шаг симуляции выбирается исходя из следующих соображений, что профильтрованные данные на выходе фильтра должны обновляться через четыре такта синхроимпульса (рис. 6.6).

Функциональный блок Increment Decrement исполняет роль 2-разрядного суммирующего счетчика (увеличивает свое содержимое на единицу) (рис. 6.7). Играет роль управляющего автомата. Счетчик управляет работой линией задержки на четыре отвода через функциональный блок “дешифратор” (рис. 6.8). “Дешифратор” фактически представляет собой компаратор, который сравнивает входной сигнал с заданной

декодируемой величиной. Дешифратор вырабатывает активный сигнал (десятичная единица) через каждые три последующие временных шага моделирования (на четвертый шаг), который подключен ко 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 эмулируется работа линия задержки на базе сдвигового регистра. И можно говорить, что ОЗУ работает в режиме циклического буфера. Счетчик так же 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 двух портовой памяти и мультиплексора показана на рис. 6.9. Двух портовая 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. 6.10 показаны настройки функционального блока Dual-Port RAM.

Настройки функционального блока LUT показаны на рис. 6.11. Выходные шины данных функциональных блоков Dual-Port RAM и LUT должны быть регистерные (рис. 6.10, *б* и рис. 6.11, *б*). Это обеспечивает правильность функционирования блока MultiplyAccumulate. Настройки функционального блока MultiplyAccumulate выполняющего роль операции умножения с накоплением показаны на рис. 6.12. Информационные потоки вычислений в структуре КИХ-фильтра показаны на рис. 6.13.

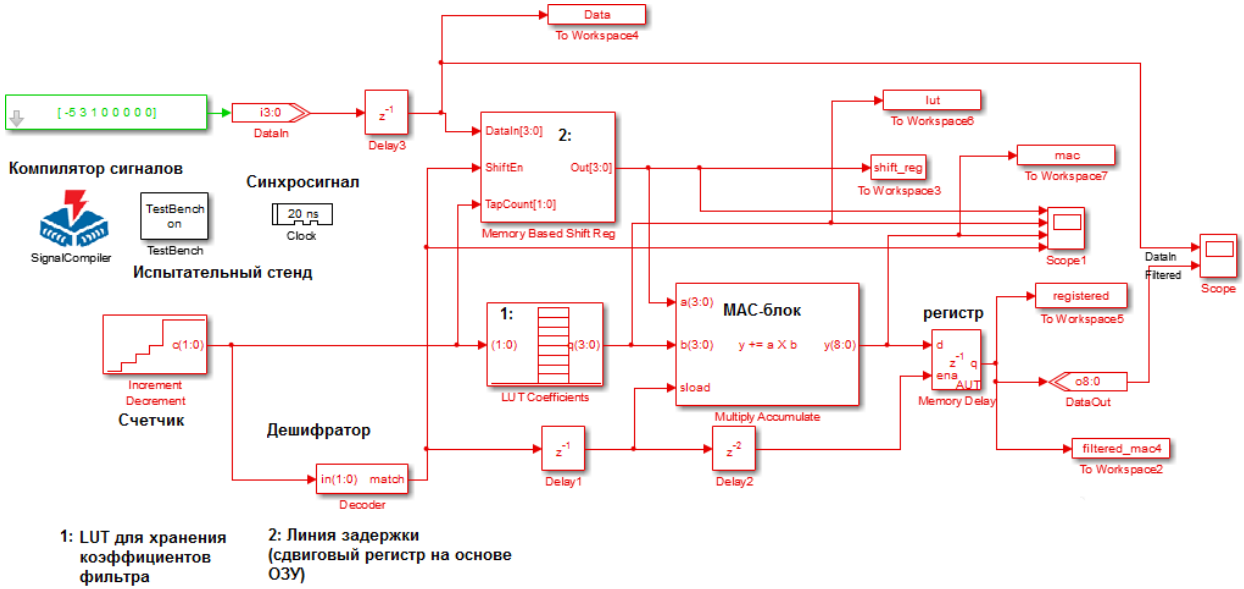


Рис. 6.5 Имитационная модель последовательного КИХ-фильтра на четыре отвода для реализации в базисе ПЛИС Altera серии Cyclone на основе одного МАС-блока

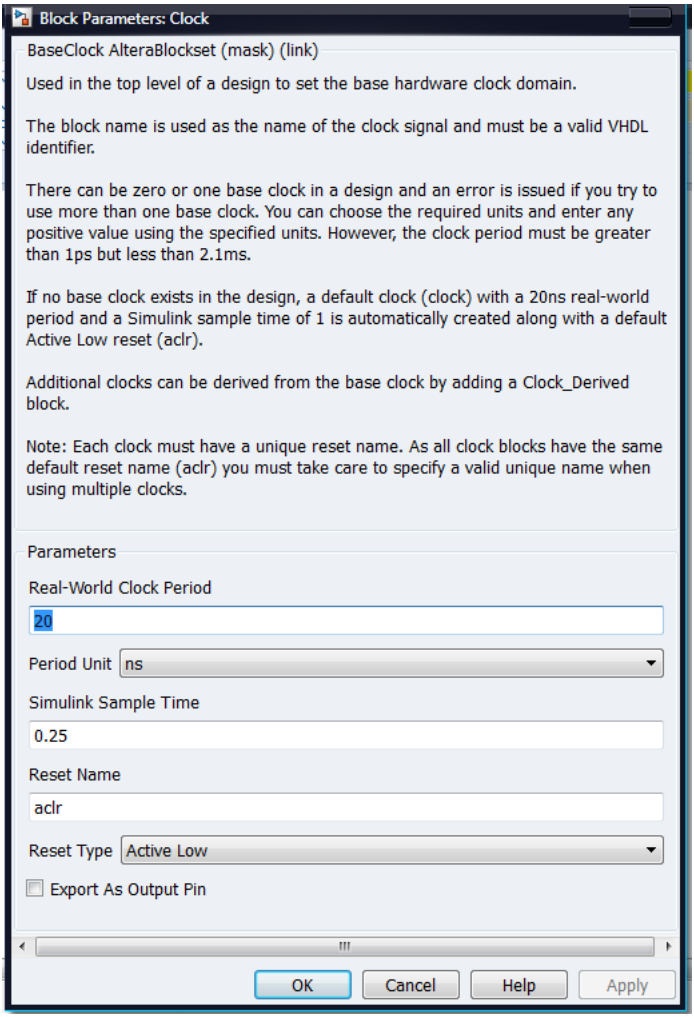


Рис. 6.6. Настройки функционального блока BaseClock. Задается период синхросигнала 20 нс для функционального моделирования в САПР Quartus II ver 12.1 и временной шаг симуляции в Matlab/Simulink 0.25 с

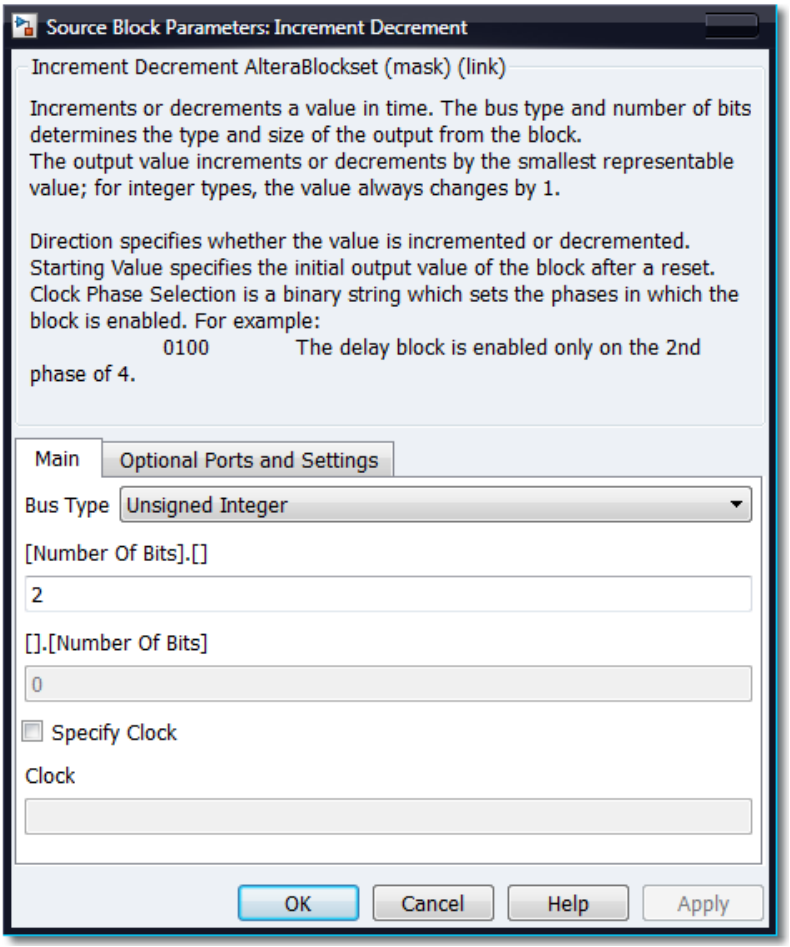


Рис. 6.7. Настройки функционального блока Increment Decrement

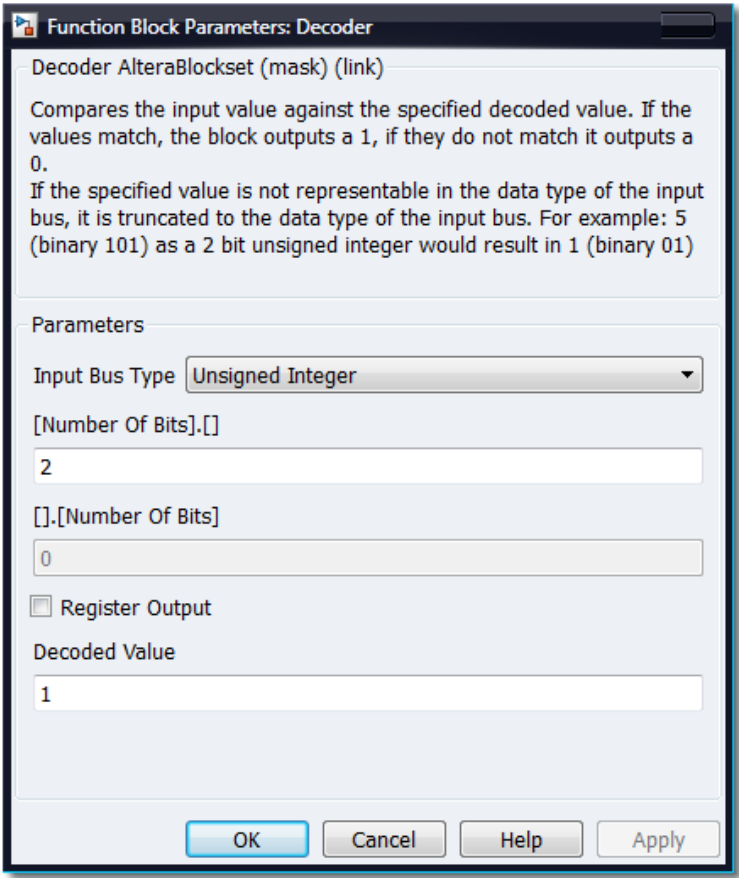


Рис. 6.8. Функциональный блок Decoder. Задается 2-разрядная входная шина типа Unsigned Integer и декодируемая величина – десятичная единица

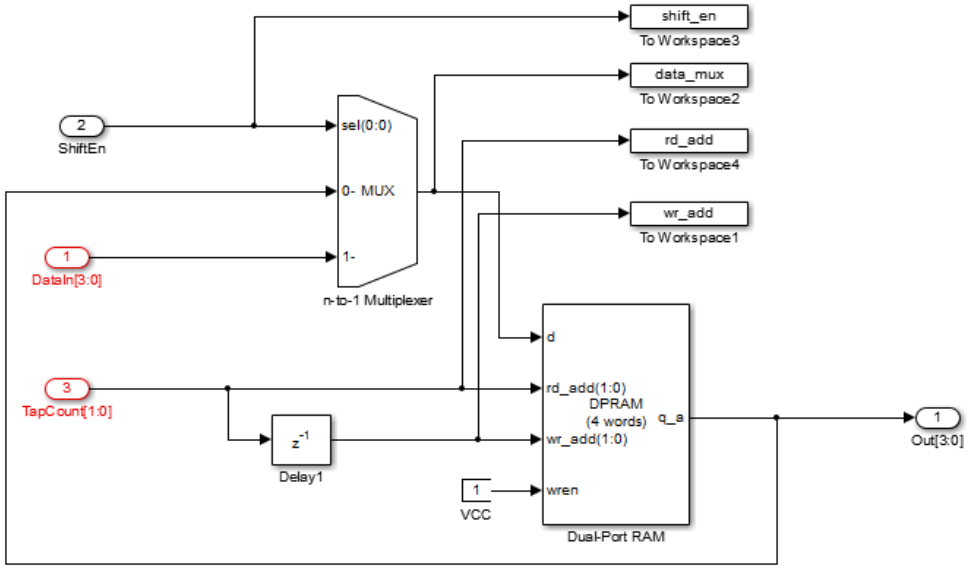
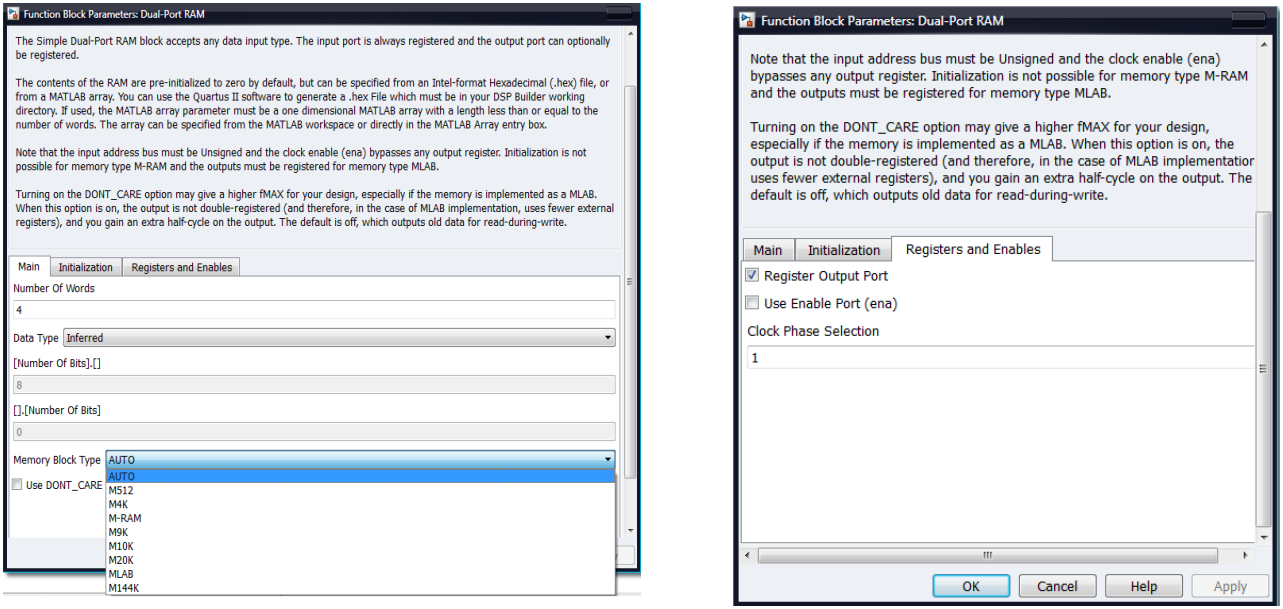
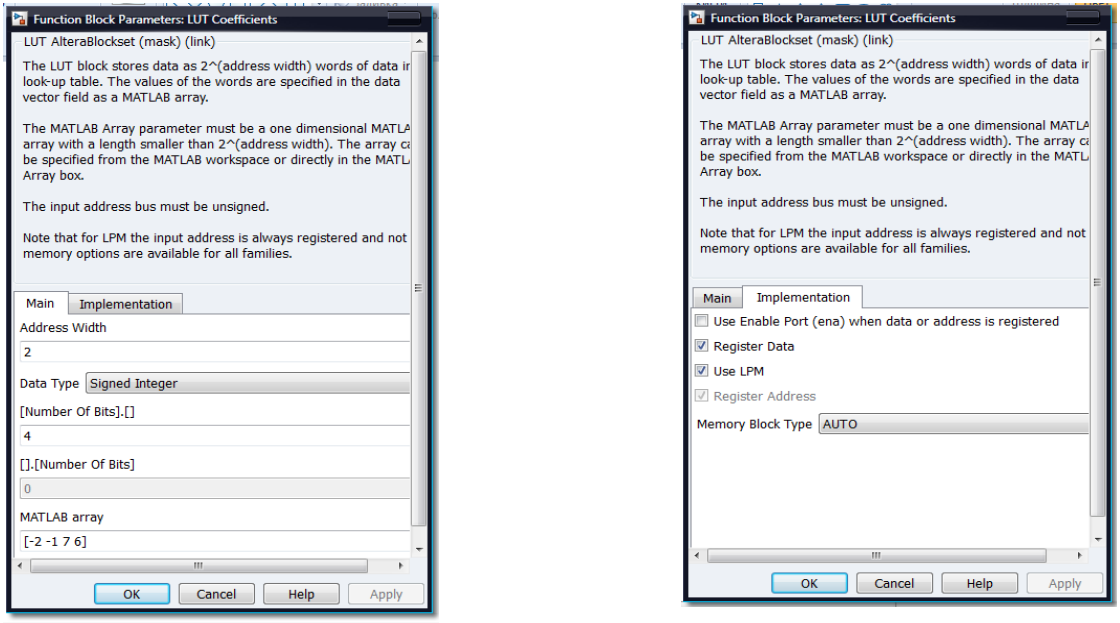


Рис. 6.9. Линия задержки на основе двух портовой памяти (функциональный блок Dual-Port RAM) и мультиплексора (функциональный блок Multiplexer)



а) б)

Рис. 6.10. Настройки функционального блока Dual-Port RAM: а) - закладка Main (задается четыре 8-разрядных слова, тип памяти Auto); б) – закладка Registers end Enables (выходной порт шины данных ОЗУ регистерный)



а) б)

Рис. 6.11. 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)

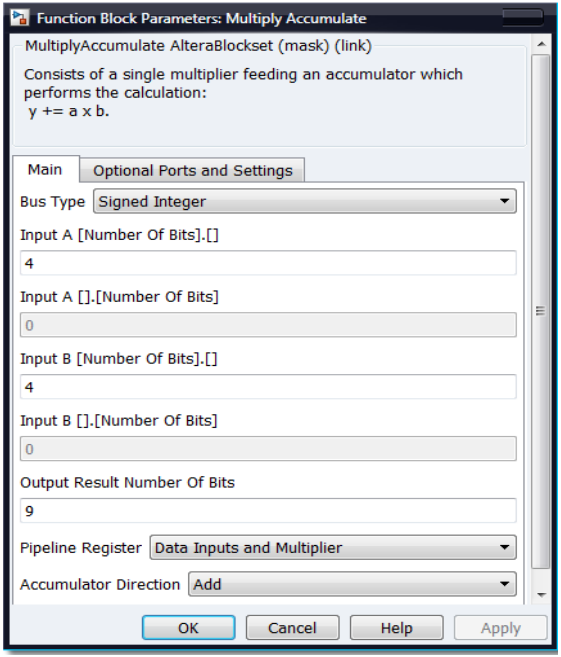


Рис. 6.12. Настройки функционального блока выполняющего роль операции умножения с накоплением (MultiplyAccumulate). Задается разрядность шин A и B (четыре бита, целые числа со знаком) и точность представления результата вычисления (девять бит)

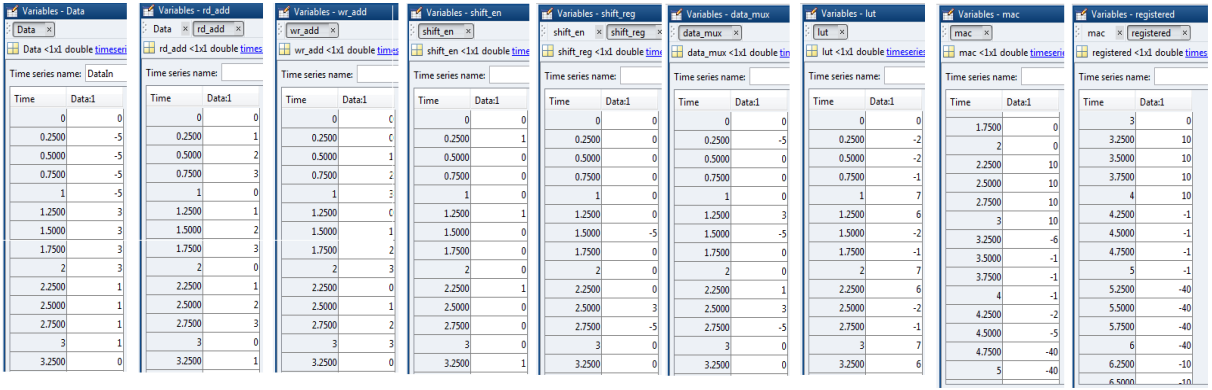


Рис. 6.13. Информационные потоки: а) сигнал подлежащий фильтрации после операции задержки; б) и в) - rd_add и wr_add – адреса на шинах связанных с операциями чтения и записи информации в ОЗУ; г) shift_en – сигнал разрешения записи новых значений сигнала подлежащего фильтрации в линию задержки; д) shift_reg – значения на выходной шине ОЗУ (шина q_a, рис.10); е) data_mux – сигнал на выходе мультиплексора или значения на входной шине данных ОЗУ (шина d, рис.10); к) lut – коэффициенты КИХ-фильтра; л) mac – значения сигнала после операции умножения и накопления; м) registered – профильтрованные зхначения. Обозначение сигналов согласно рис.6 и рис.10

На рис. 6.14 показано имитационное моделирование в системе Matlab/Simulink 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, 26, 6 и 0 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, то воспользуемся версией Altera Quarus II 13.1. На рис. 6.15 показано функциональное моделирование КИХ-фильтра на четыре отвода в САПР Quartus II 13.1. 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. Сравнивая рис. 6.14 и рис. 6.15 можно сделать вывод что имитационная и функциональная модели фильтров работают корректно.

Если установлен Altera-ModelSim то необходимо добавить в модель функциональный блок генератор испытательных стендов (Testbench Cenerator). На рис. 6.16 показано создание тестбенча проекта в автоматическом режиме для последущей симуляции в Altera-ModelSim. На рис. 6.17 показано функциональное моделирование КИХ-фильтра на четыре отвода в Altera-ModelSim.

В табл. 6.1 и 6.2 даны сравнительные оценки ресурсов ПЛИС при реализации КИХ-фильтра на четыре отвода с использованием с использованием пакета расширения Altera DSP Builder и System Generator IDS системы визуально-имитационного моделирования Matlab/Simulink.

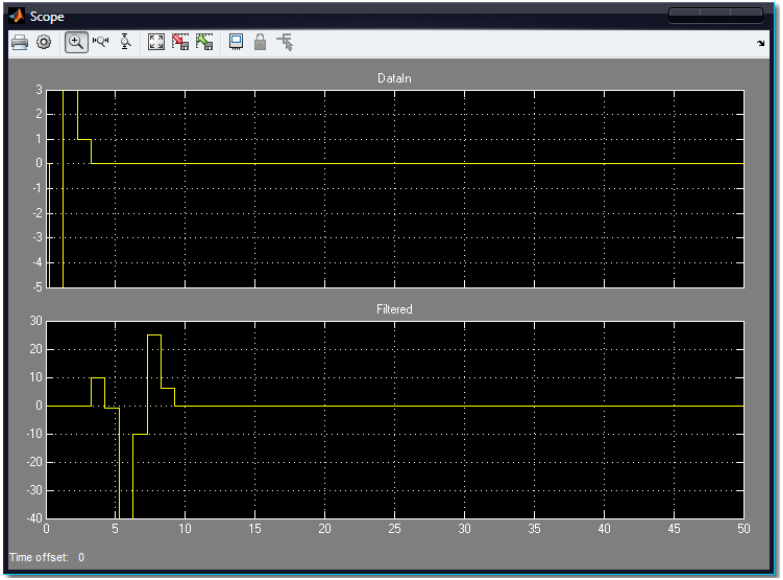


Рис. 6.14. Имитационное моделирование в системе Matlab/Simulink КИХ-фильтра на четыре отвода. На вход фильтра поступает сигнал -5, 3, 1, 0, 0 и 0 т.д. Правильные значения на выходе фильтра: 10, -1, -40, -10, 26, 6 и 0 т.д.

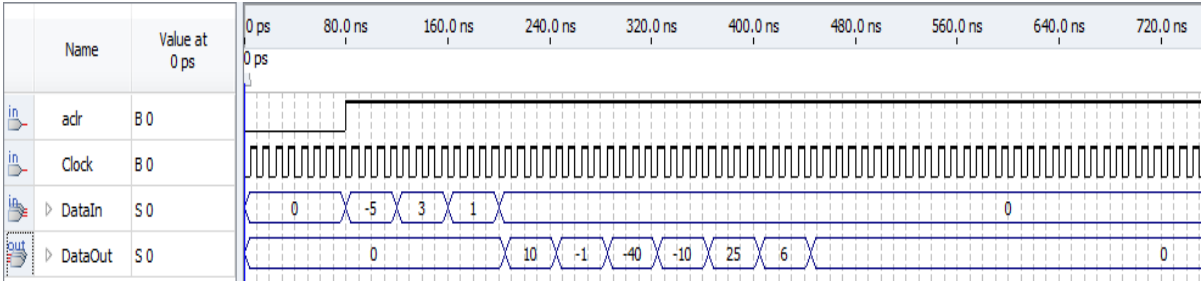


Рис. 6.15. Функциональное моделирование КИХ-фильтра на четыре отвода в САПР Quartus II 13.1. На вход фильтра поступает сигнал -5, 3, 1, 0, 0 и 0 т.д. Правильные значения на выходе фильтра: 10, -1, -40, -10, 26, 6 и 0 т.д.

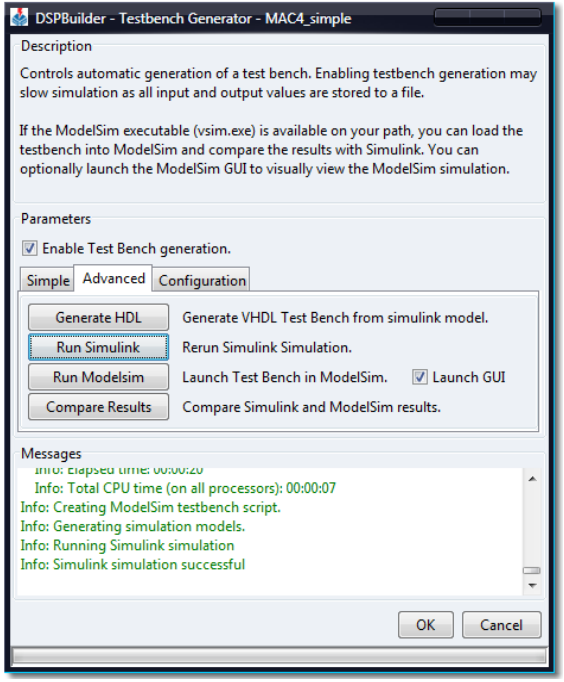


Рис. 6.16. Генератор испытательных стендов (функциональный блок Testbench Cenerator)

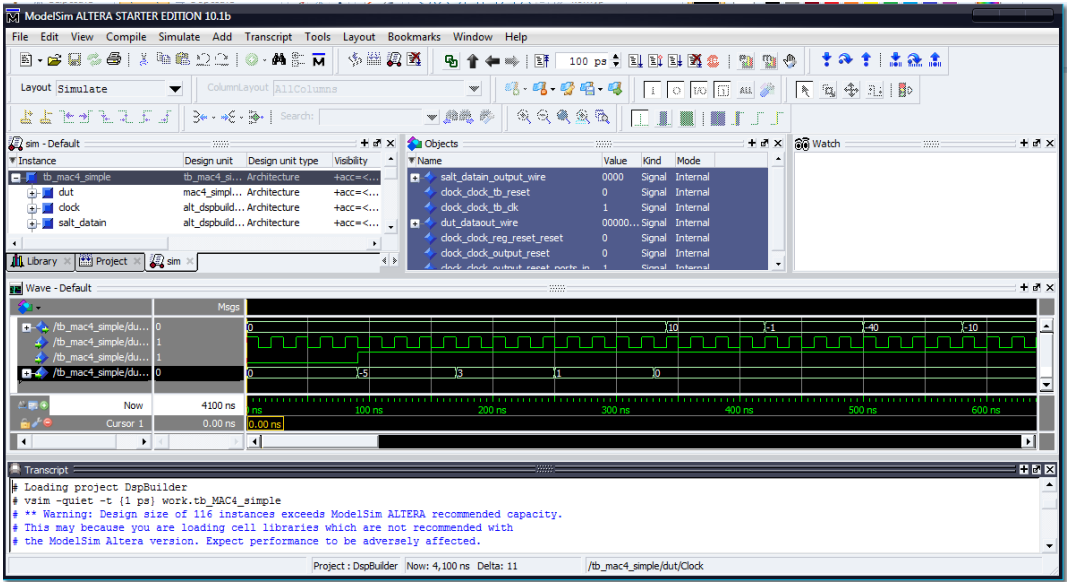


Рис. 6.17. Функциональное моделирование КИХ-фильтра на четыре отвода в Altera-ModelSim. На вход фильтра поступает сигнал -5, 3, 1, 0, 0 и 0 т.д. Правильные значения на выходе фильтра: 10, -1, -40, -10, 26, 6 и 0 т.д.

Таблица 6.1

Общие сведения по числу задействованных ресурсов ПЛИС Altera Cyclone III Device EP3C5F256C6, временная модель Slow-model (напряжение питания ядра 1200 мВ, температура 85 С)

Общее число логических элементов (Total logic elements)	Триггеров логических элементов (Dedicated logic registers)	Аппаратных умножителей с размерностью операндов 9x9 (Embedded Multiplier 9-bit elements)	Рабочая частота в наихудшем случае Fmax, МГц
30	29	1	258

Таблица 6.2

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

Ресурсы ПЛИС	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

Пакет расширения Altera DSP Builder системы визуально-имитационного моделирования Matlab/Simulink является мощным инструментом по разработке устройств цифровой обработки сигналов и по своим возможностям не уступает Xilinx System Generator.

Имитационная модель на основе функционального блока n-tap Dual Port Memory MAC FIR имеет отличие в организации управления работой линии задержки, заключающееся в том что, вначале происходит операция считывания информации хранящейся в памяти ОЗУ по всем

адресам а затем - операция записи нового значения сигнала по конкретному адресу (рис. 6.2). На основе рассматриваемого примера (FIR_MAC32.mdl) происходит одновременное считывание и запись разделенные во времени элементом задержки. В первом случае требуется более сложный управляющий автомат (рис. 6.2) а во втором случае (рис. 6.5) необходим простейший управляющий автомат и обратная связь заключающая в подачи данных с выходной шины ОЗУ на вход через мультиплексор на другой вход которого подается сигнал подлежащий фильтрации.

Разработаем функциональную модель КИХ-фильтра по имитационной (рис. 6.18). В этом случае мы сможем более наглядно разобраться с режимом работы двух портовой памяти ПЛИС.

Увеличим разрядность шины данных ОЗУ с четырех до восьми бит. В отличие от имитационной модели для формирования лог. единицы используем сигнал cout двухразрядного счетчика пропустив его через два триггера. Он будет сигнализировать о том, что коэффициенты фильтра из ПЗУ (мегафункция ROM: 1 port) считаны (аккумулятор за это время должен вычислить “правильную” сумму произведений) и новые данные могут быть записаны в память.

На рис. 6.19 показано, что мегафункция RAM: 2-PORT для выходных значений на шине q настроена на режим Old memory contents appear. На рис. 6.20 представлено функциональное моделирование прохождения сигнала по структуре КИХ-фильтра в режиме Old memory contents appear. Рис. 6.21 показывает функциональное моделирование прохождения сигнала по структуре КИХ-фильтра для режима I do not care. Сравнивая рис. 6.20 и рис. 6.21 видим, что при подачи сигнала -5, 3, 1, 0, 0 и 0 т.д. на вход фильтра, в случае если двух портовая память работает в режиме I do not care, на выходе формируются не требуемые значения 10, -1, -40, -10, 26, 6 и 0 т.д. а значения -40, -10, 26, 6 и 0 т.д., т.е. значения 10, -1 пропущены. Тем не менее, фильтр после 980 нс начинает формировать правильные значения. До этого момента времени выход фильтра не определен.

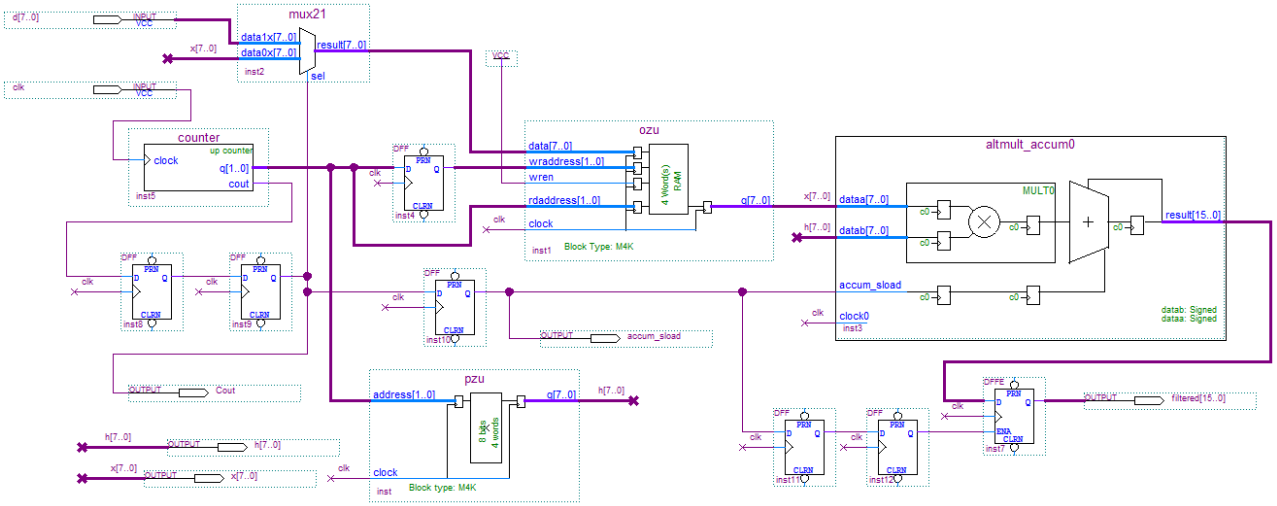


Рис. 6.18. Функциональная модель последовательного КИХ-фильтра на четыре отвода с использованием линии задержки на основе двух портовой памяти

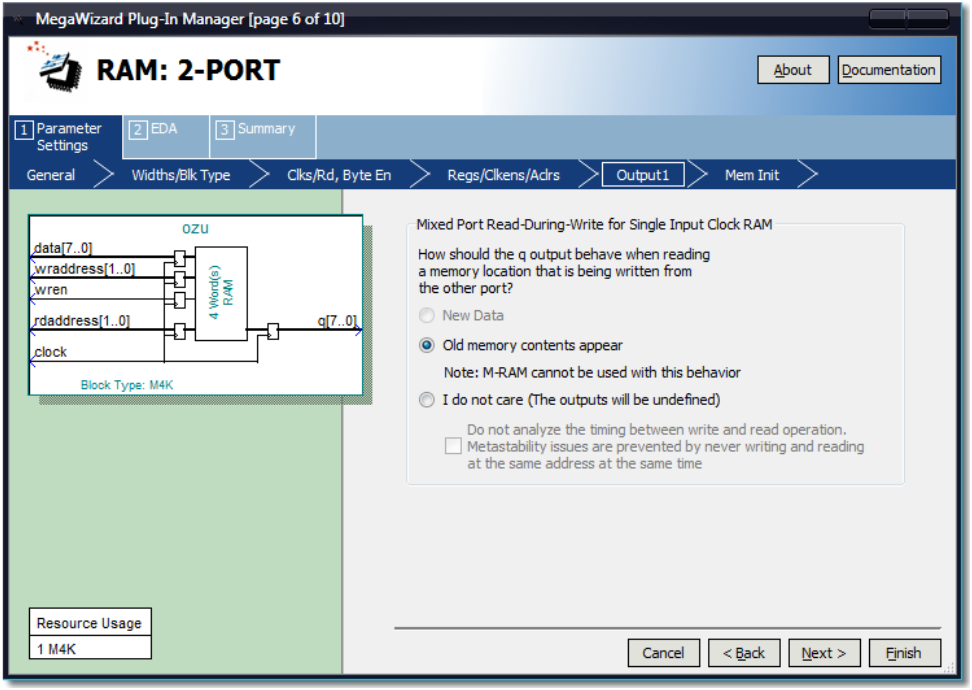


Рис. 6.19. Мегафункция RAM: 2-PORT. Режим Old memory contents appear для выходной шины q

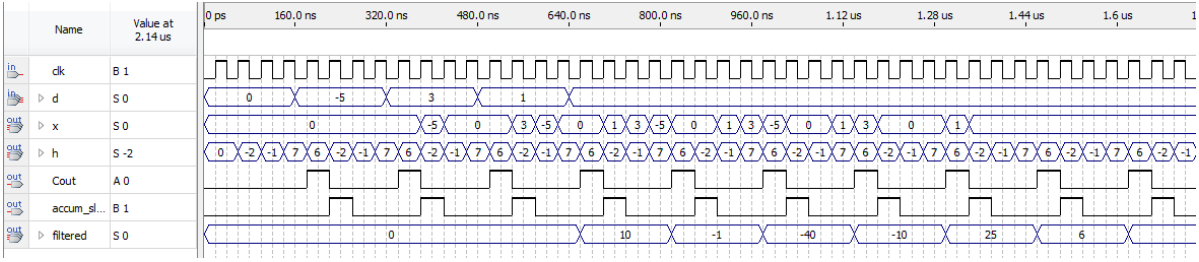


Рис. 6.20. Функциональное моделирование прохождения сигнала по структуре КИХ-фильтра на четыре отвода. Режим Old memory contents appear

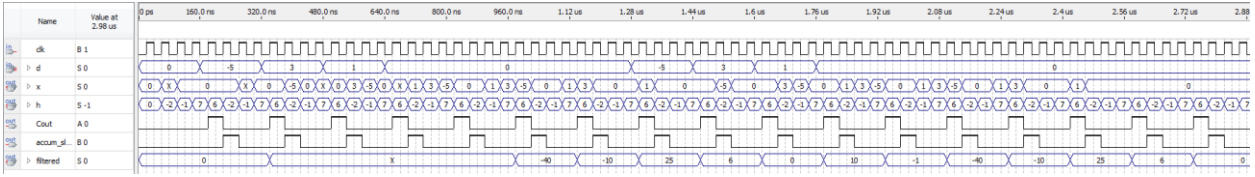


Рис. 6.21. Функциональное моделирование прохождения сигнала по структуре КИХ-фильтра на четыре отвода. Режим I do not care

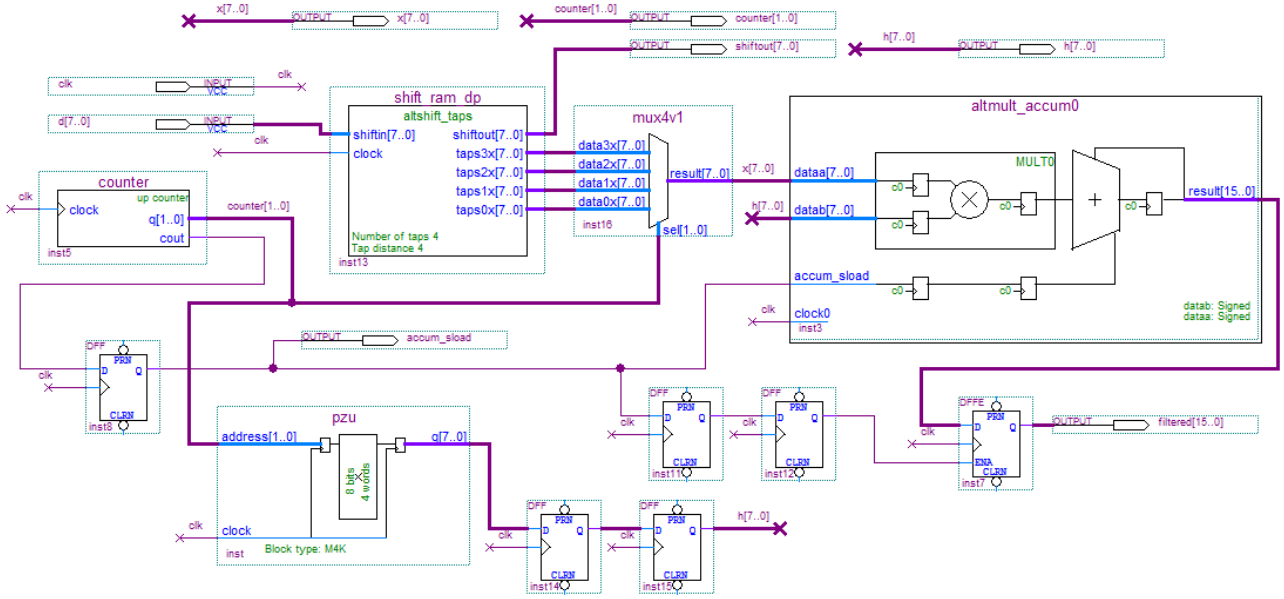


Рис. 6.22. Линия задержки на основе двух портовой памяти (мегафункция Shift Register (RAM-based))

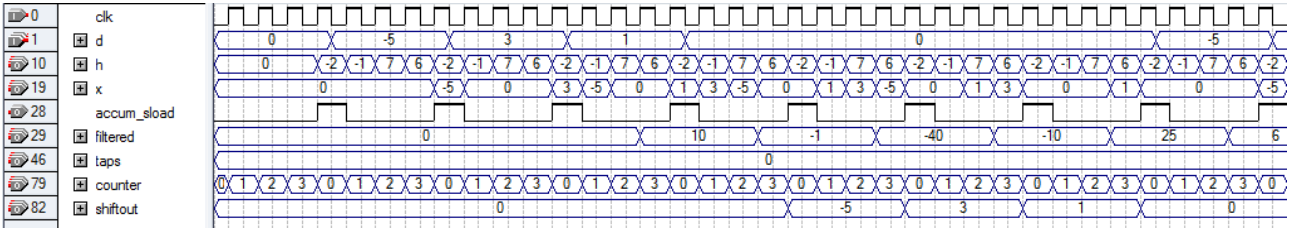


Рис. 6.23. Функциональное моделирование прохождения сигнала по структуре КИХ-фильтра (линия задержки на основе мегафункции Shift Register (RAM-based))

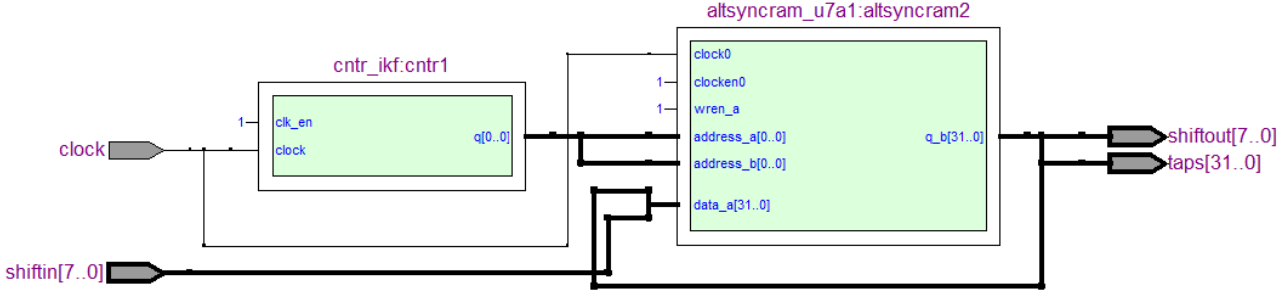


Рис. 6.24. RTL-представление линии задержки на основе мегафункции Shift Register (RAM-based)

Линию задержки можно так же реализовать на основе двух портовой памяти с использованием мегафункции altshift_taps (Shift Register (RAM-based)) (рис. 6.22) со следующими установками: число отводов – 4; дистанция между отводами – 4. Коммутация отводов линии задержки осуществляется с помощью шинного мультиплексора четыре в один на адресный вход которого подключается выход счетчика counter. Для согласования работы линии задержки и ПЗУ используемого для хранения коэффициентов фильтра с блоком умножения и накопления на основе мегафункции ALTMULT_ACCUM к выходу ПЗУ требуется дополнительно подключить два 8-разрядных регистра.

Для выходного порта q используется только режим old. В мегафункции altshift_tap не предоставляется изменение режимов работы выходного порта при одновременном чтении и записи по одинаковому адресу как для мегафункции RAM: 2-PORT. Результаты функционального моделирования показаны на рис. 6.23. На рис. 6.24 показано RTL-представление линии задержки на основе счетчика и блока памяти.

6.2. Проектирование КИХ-фильтров Добеши в системе визуально-имитационного моделирования Matlab/Simulink с использованием Altera DSP Builder

Вейвлет-преобразование одномерного сигнала $s(t)$ состоит в его разложении по базису, сконструированном из обладающей определенными свойствами солитоноподобной функции посредством масштабных изменений и переносов:

$$W(a,b) = a^{-1/2} \int_R s(t) \psi^* \left(\frac{t-b}{a} \right) dt ,$$

a – масштаб анализа; b – сдвиг вейвлета по времени; $W(a,b)$ – коэффициенты вейвлет-преобразования или двумерный массив амплитуд вейвлет-преобразования; t – время; R – множество действительных чисел; ψ – локализованная по времени и частоте, быстростремящееся к нулю, двухпараметрическая вейвлет-функция; звездочка над ψ обозначает комплексное сопряжение. Варьируя значения параметров a и b можно получить вейвлет-спектр (time-scale spectrum (масштабно-временной спектр) или wavelet spectrum). Вейвлет-преобразование обеспечивает двумерную развертку сигнала $s(t)$, при этом частота и координата рассматриваются как независимые переменные.

КИХ-фильтры с небольшим числом отводов на практике могут быть использованы для проектирования фильтров Добеши второго порядка. Для получения информации о вейвлетах Добеши необходимо в командной строке Matlab набрать waveinfo('db'). Вейвлеты Добеши dbN относятся к ортогональным вейвлетам с компактным носителем. В системе Matlab вейвлеты Добеши задаются конечно-импульсной характеристикой – набором весовых вейвлет-коэффициентов.

Рассмотрим четырехточечный фильтр Добеши db2. Графики вейвлетов Добеши db2 в системе Matlab можно увидеть следующим образом:

```
[phi,psi,x]=wavefun('db2',10);
subplot(121);plot(x,phi);
title('y=\phi(x)'); axis square; grid on;
subplot(122);
plot(x,psi);
title('y=\psi(x)'); axis square; grid on;
```

phi обозначает масштабирующую функцию - $\phi(x)$; psi – материнский вейвлет - $\psi(x)$; X - массив значений независимой переменной x; 10 – число итераций (степень итерационного уточнения). Функции $\phi(x)$ и $\psi(x)$ быстрозатухающие, имеют компактный носитель и не имеют аналитических выражений.

С помощью команд можно так же посмотреть масштабирующую функцию $\phi(x)$ вейвлета Добеши db2 и весовые вейвлет-коэффициенты фильтров декомпозиции и реконструкции сигнала (рис. 6.25). Коэффициенты фильтра даются с учетом нормировки множителя $1/\sqrt{2}$.

```
load db2; w = db2; iter = 10; wav = 'db2';
wn=sqrt(2)*w;
[phi,psi,xval] = wavefun(wav,iter);
subplot(321); plot(xval,psi); title('Wavelet');
subplot(322); stem(wn); title('Original scaling filter');
[Lo_D, Hi_D, Lo_R, Hi_R] = orthfilt(wn);
subplot(323); stem(Lo_D); title('Decomposition low-pass filter');
subplot(324); stem(Hi_D); title('Decomposition high-pass filter');
subplot(325); stem(Lo_R); title('Reconstruction low-pass filter');
subplot(326); stem(Hi_R); title('Reconstruction high-pass filter');
```

Посмотреть графики вейвлетов Добеши можно и с помощью команды wavemenu и в появившемся окне с описанием разделов вейвлет преобразования нажать кнопку Wavelet Display. Выводится следующее окно, в котором, выбрав имя, wname, можно просмотреть весовые вейвлет-коэффициенты фильтров декомпозиции (Lo _D – low-pass (ФНЧ), Hi _D – high-pass (ФВЧ)) и реконструкции (Lo _R , Hi_ R) сигнала.

Низкочастотные (h) и высокочастотные (g) коэффициенты фильтра Добеши db2 задаются следующими коэффициентами:

$h_0 = \left(1+\sqrt{3}\right)/\left(4\sqrt{2}\right) = 0.4830$, $h_1 = \left(3+\sqrt{3}\right)/\left(4\sqrt{2}\right) = 0.8365$,

$h_2 = \left(3-\sqrt{3}\right)/\left(4\sqrt{2}\right) = 0.2241$, $h_3 = \left(1-\sqrt{3}\right)/\left(4\sqrt{2}\right) = -0.1294$,

$g_0 = h_3$, $g_1 = -h_2$, $g_2 = h_1$, $g_3 = -h_0$.

Передаточную функцию фильтра Добеши db2 можно записать в виде:

$$G(z) = \frac{\left(1+\sqrt{3}\right)+\left(3+\sqrt{3}\right)z^{-1}+\left(3-\sqrt{3}\right)z^{-2}+\left(1-\sqrt{3}\right)z^{-3}}{4\sqrt{2}},$$

$$G(z) = 0.4830 + 0.8365z^{-1} + 0.2241z^{-2} - 0.1294z^{-3}.$$

После умножения на масштабный множитель 256 коэффициенты округлим до ближайшего целого числа:

$$G(z) = 124 + 214z^{-1} + 57z^{-2} - 33z^{-3}.$$

Для получения правильного результата фильтрации необходимо в дальнейшем предусмотреть деление на масштабный множитель 256:

$$y/256 = 124x_0 + 214x_1 + 57x_2 - 33x_3.$$

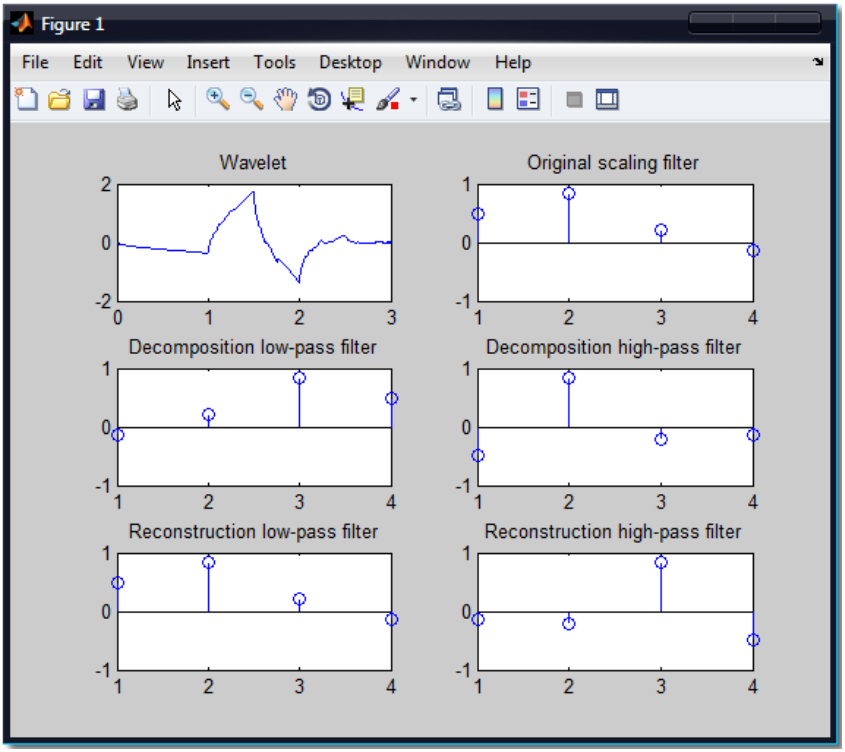


Рис. 6.25. Масштабирующая функциия $\phi(x)$ вейвлета и коэффициенты фильтра Добеши db2 а так же весовые вейвлет-коэффициенты фильтров декомпозиции и реконструкции сигнала

Вейвлет-преобразование сигналов может быть представлено как банк фильтров. Быстрый алгоритм Малла (дискретное вейвлет преобразование) для трех уровней разложения сигнала s показан на рис. 6.26, где a1, a2, a3 – аппроксимирующие коэффициенты а d1, d2 и d3 – детализирующие коэффициенты, цифры 1, 2, 3 обозначают уровни разложения сигнала. Верхняя часть схемы соответствует процедуре декомпозиции а верхняя процедуре реконструкции сигнала.

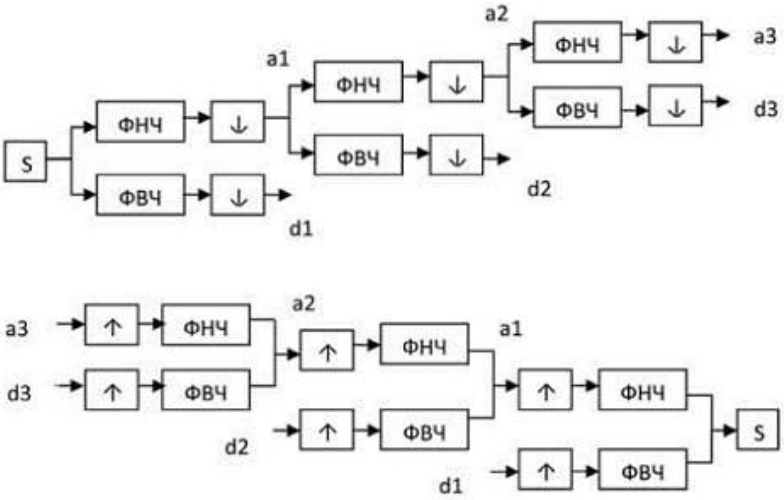


Рис. 6.26. Быстрый алгоритм Малла для трех уровней разложения сигнала

Рассмотрим проектирование односкоростного (Single-Rate FIR) КИХ-фильтра Добеши в САПР Quartus II с использованием мегафункции Altera fir_compiler_v13_1 (рис. 6.27). Коэффициенты фильтра возьмем целочисленными со знаком 124, 214, 57, -33. Предположим, что они считываются из текстового файла и не подвергаются масштабированию (опция Coefficients Scaling None), т.к. эта операция была проделана ранее путем умножения на масштабный множитель 256.

Выберем структуру фильтра на последовательной распределенной арифметике с одним уровнем конвейеризации. Предположим, что коэффициенты фильтра и отсчеты будут храниться в логических ячейках ПЛИС и блочная память использоваться не будет. Спецификация по входу следующая: одноканальный фильтр, на вход фильтра поступают целые числа со знаком, точность представления которых в дополнительном коде – девять разрядов. Спецификация по выходу: полная точность представления результата

фильтрации – 18 разрядов (определяется в автоматическом режиме по соответствующей нотации, основывается на методе Actual Coefficients). Таким образом для сигнала и коэффициентов фильтра используется формат с фиксированной запятой для целых чисел со знаком (Integer Fixed-Point Representation).

На рис.6.28 показан проект КИХ-фильтр Добеши db2 на четыре отвода в САПР Quartus II ver.13.1 с использованием мегафункции FIR Compiler. Латентность фильтра составляет 9 тактов синхроимпульсов (рис.6.29).

Для учета влияния эффектов квантования на импульсную характеристику фильтра при переходе к формату с фиксированной запятой для дробных чисел со знаком (Fractional Fixed-Point Representation известен как Q-формат) необходимо коэффициенты загрузить из файла в виде дробных чисел со знаком $0.4830, 0.8365, 0.2241, -0.1294$ и воспользоваться опцией Signed Binary Fractional (нотация), при этом в поле битовая ширина (Bit Width) выставим формат:

.

В рассматриваемом случае на знак выделяется один разряд а на дробную часть числа восемь разрядов, при этом общая длина разрядной сетки будет девять разрядов).

В качестве примера кратко рассмотрим нотацию Qm.n наиболее часто применяемой для представления чисел в цифровых сигнальных процессорах, m -число разрядов целой части числа, n – число разрядов дробной части (рис. 6.30).

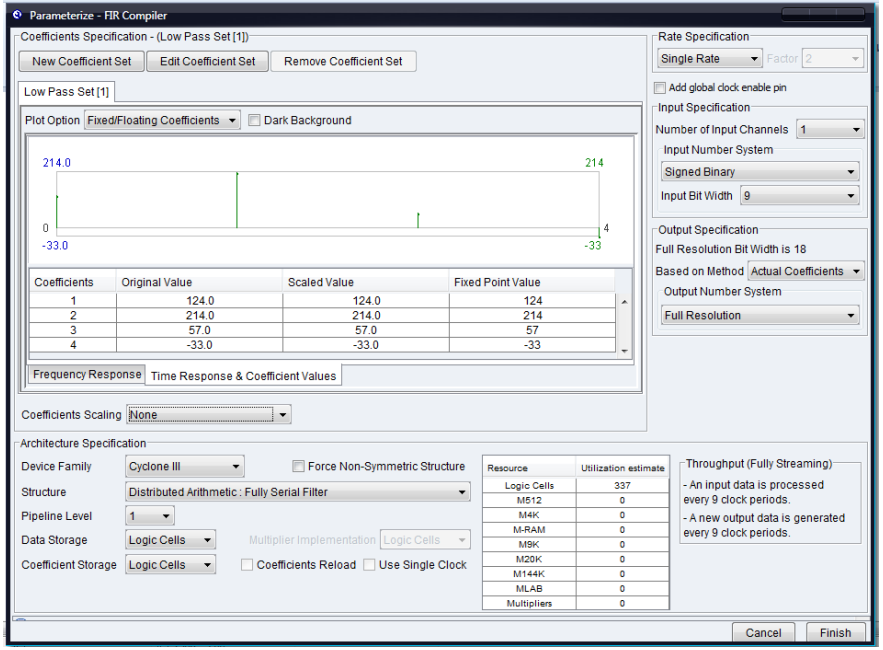


Рис. 6.27. Настройки мегафункции Altera fir_compiler_v13_1. Показана импульсная характеристика КИХ-фильтра (коэффициенты загружаются целочисленными, предварительно умноженные на масштабный множитель 256)

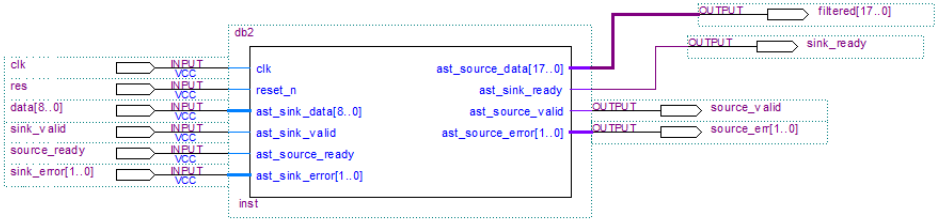


Рис. 6.28. Проект КИХ-фильтра Добеши db2 на четыре отвода в САПР Quartus II ver.13.1 с использованием мегафункции FIR Compiler

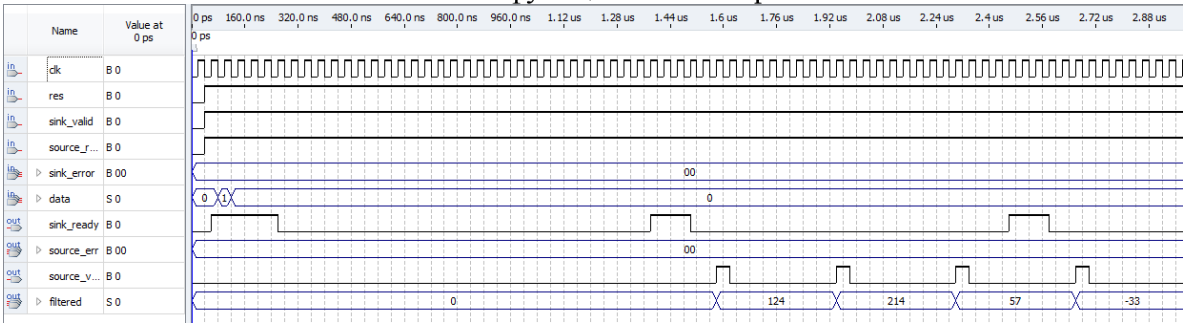


Рис. 6.29. Функциональное моделирование с использованием встроенного векторного редактора (импульсная характеристика КИХ-фильтра Добеши db2). Коэффициенты фильтра 124, 214, 57, - 33

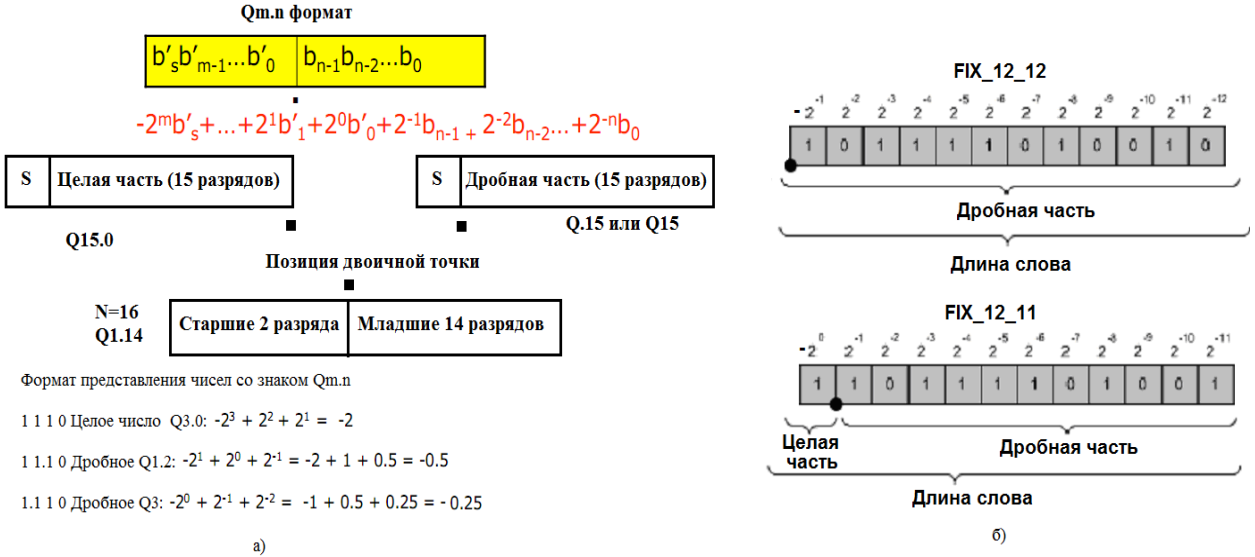


Рис. 6.30. Представление чисел в формате с фиксированной запятой: а) формат Qm.n б) формат FIX в System Generator

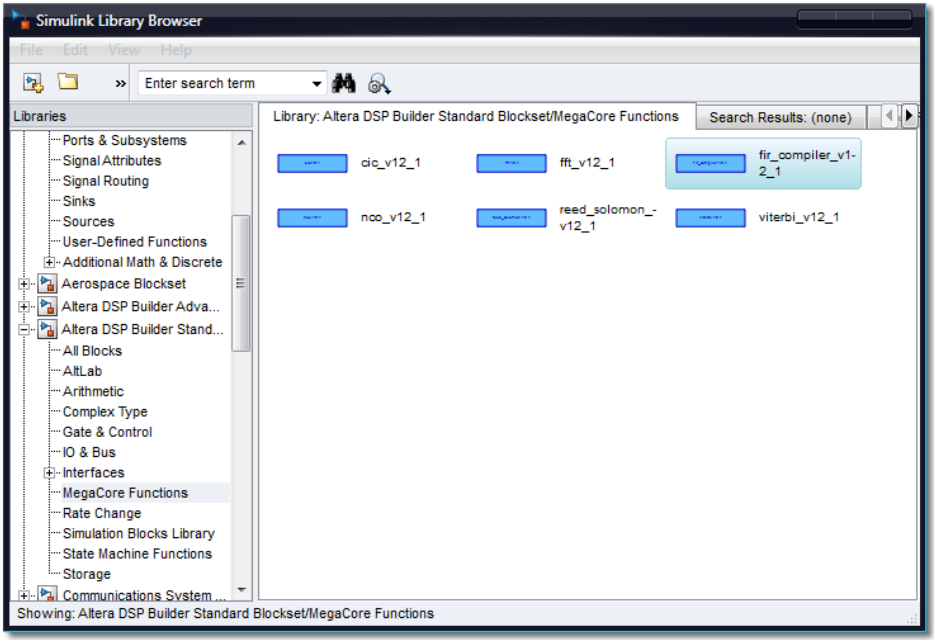


Рис. 6.31. Функциональный блок fir_compiler_v12_1 библиотеки Altera DSP Builder Standard Blokset/MegaCore Function

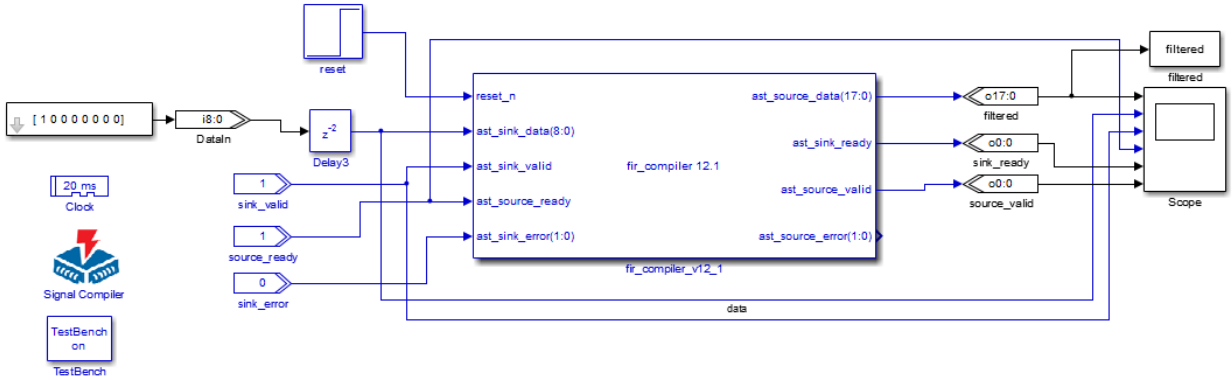


Рис. 6.32. Имитационная модель КИХ-фильтра Добеши db2 на четыре отвода с использованием функционального блока fir_compiler_v12_1

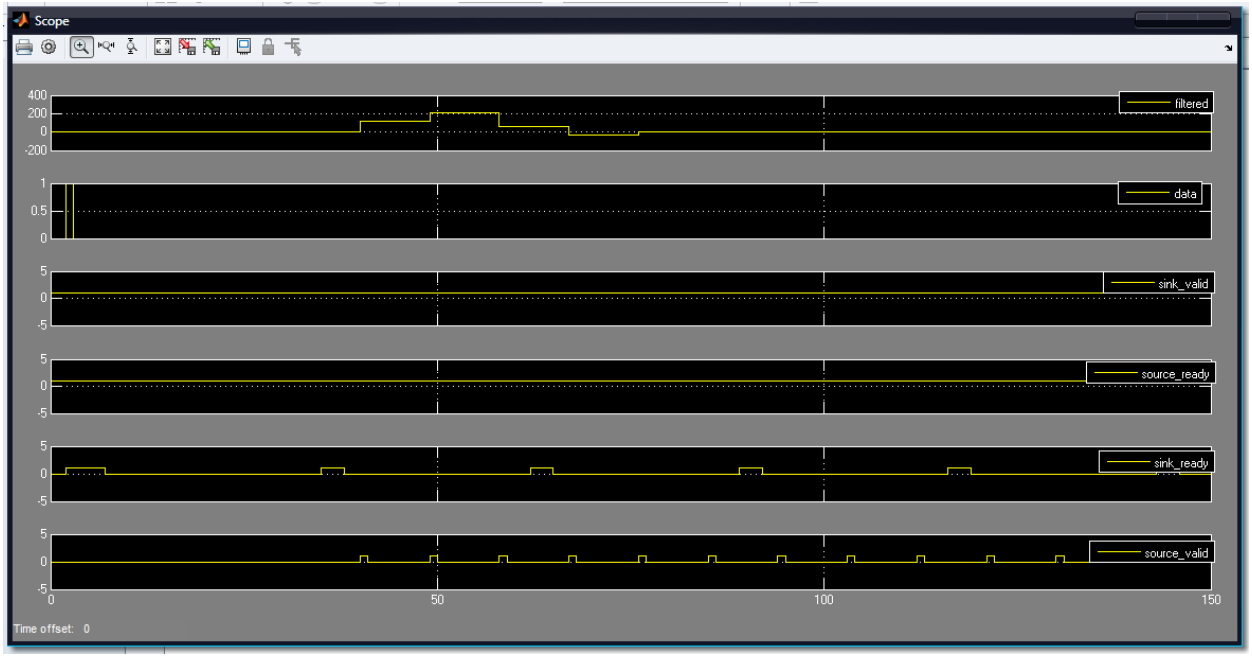


Рис. 6.33. Имитационное моделирование импульсной характеристики в системе Matlab/Simulink КИХ-фильтра Добеши db2 на четыре отвода

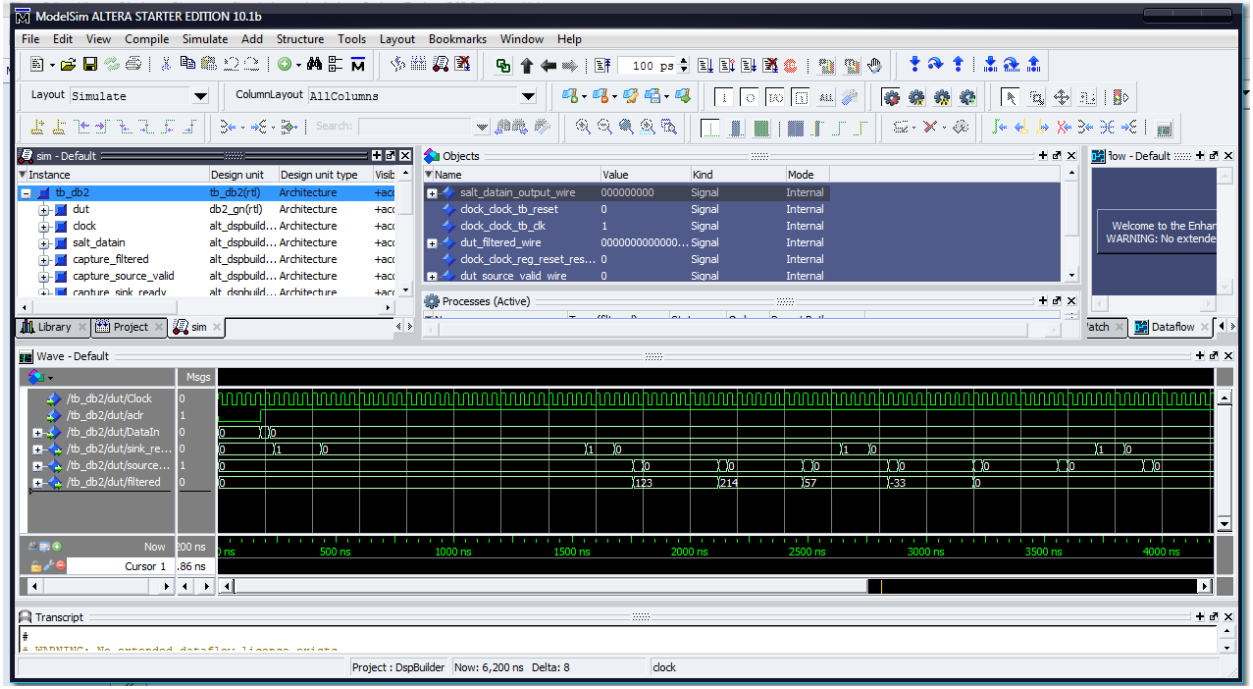


Рис. 6.34. Функциональное моделирование импульсной характеристики КИХ-фильтра Добеши db2 на четыре отвода в Altera-ModelSim

Общее число разрядов N=m+n+1 для чисел со знаком. Для представления чисел со знаком в формате с фиксированной запятой Xilinx System Generator использует нотацию FIX, а для без знаковых - UFIX. Формат FIX можно рассматривать как пару чисел M.N, где M - общее число двоичных разрядов; N – число разрядов дробной части [11].

Для 16-разрядного числа N=16 можно записать: Q2.13, где 2 – число разрядов целой части числа, 13 – число разрядов дробной части числа + 1 разряд на знак (СЗР, старший значащий разряд). Для целых чисел со знаком N=16 и Q15.0, для дробных чисел со знаком N=16 и Q0.15 так же известен как формат Q.15 или Q15. В отечественном учебном пособии по цифровой обработке сигналов Сергиенко А.Б. основанном в большей части на документации системы Matlab этот формат известен как формат с фиксированной запятой 1.15 (целые числа необходимо делить на коэффициент $2^{15} = 32768$ равный числу разрядов дробной части). Применительно к нашему случаю это будет формат Q8.

В Altera f ir_compiler в САПР Quartus II и в fir_compiler DSP Builder знак включен в целую часть числа. Из справочной документации можно узнать что нотация Signed Binary Fractional представления дробных чисел со знаком в Altera fir_compiler следующая:

<Знак><Число разрядов целой части>.<Число разрядов дробной части числа>

Формат представления сигнала подлежащего фильтрации и коэффициентов фильтра следующие:

<Знак><x1 разрядов целой части>.<y1 разрядов дробной части числа>

<Знак><x2 разрядов целой части>.<y2 разрядов дробной части числа>.

При этом полная точность представления результата фильтрации:

<Знак><i разрядов целой части>.<y1+y2 разрядов дробной части числа>,

где i=ceil(log2(число коэффициентов))+x1+x2.

Полная точность представления результата фильтрации в случае если сигнал и коэффициенты фильтра 9-разрядные (x1=0, x2=0, ceil(log2(4))=2):

<Знак><2+0+0>.<8+8 >,

составит 18 разрядов по методу Actual Coefficients (фактически задействованные разряды) и 20 по методу Bit Width Only (только битовая ширина поля).

Если же выбрать опцию Auto with Power 2 поле Bit Width 9 то получим следующие коэффициенты в формате с фиксированной запятой 123, 214, 57, -33, что приведет к потере точности.

Рассмотрим разработку имитационной модели КИХ-фильтра с использованием последовательной распределенной арифметики с применением функционального блока fir_compiler_v12_1 пакета расширения Altera DSP Builder системы Matlab/Simulink (рис. 6.31). Функциональный блок fir_compiler_v12_1 являетяся аналогом мегафункции FIR Compiler САПР Quartus II. Функциональный блок fir_compiler_v12_1 входит в состав библиотеки Altera DSP Builder Standard Blokset/MegaCore Function.

Следует заметить что в Simulink также существуют функциональный блок FIR Compiler v6.3 (FIR Compiler v5.0, распределенная арифметика) пакета расширения System Generator являющийся аналогом функции FIR Compiler v6.3 (FIR Compiler v5.0) САПР Xilinx ISE получаемой с помощью генератора параметризированных ядер XLogiCORE IP.

На рис. 6.32 представлена модель КИХ-фильтра Добеши db2 на четыре отвода с использованием функционального блока fir_compiler_v12_1 библиотеки Altera DSP Builder а на рис. 6.33 и рис. 6.34 показано имитационное и функциональное моделирование импульсной характеристики.

Пакет расширения Altera DSP Builder системы визуально-имитационного моделирования Matlab/Simulink позволяет быстро и эффективно разрабатывать сложные устройства цифровой обработки сигналов. Так же как и пакет расширения Xilinx System Generator обеспечивает поддержку работы со сложными функциональными блоками являющимися аналогами мегафункций САПР ПЛИС например, мегафункция fir_compiler. Использование мегафункции fir_compiler при проектировании КИХ-фильтров в проектах пользователя является наиболее эфеективным решением, т.к. при этом учитываются архитектурные особенности ПЛИС а так же влияние эффектов квантования при переходе к формату с фиксированной запятой на импульсную характеристику фильтра.

ЗАКЛЮЧЕНИЕ

В учебном пособии на обширном иллюстративном материале показаны методы обработки цифровых сигналов базисе ПЛИС с учетом их архитектурных особенностей с применением высокоуровневого языка описания аппаратных средств.

Изложены основы проектирования умножителей цифровых сигналов. Подробно рассмотрен алгоритм реализации умножения целых чисел, представленных в дополнительном коде, методом правого сдвига и сложения с накоплением. Даются общие сведения по программным умножителям в базисе ПЛИС.

Показан пример расчета спецификации КИХ-фильтра, показаны эффекты квантования при работе в формате с фиксированной запятой, а также продемонстрировано имитационное моделирование модели КИХ-фильтра в системе Matlab/Simulink с переходом на функциональные модели в САПР ПЛИС.

Демонстрируются различные варианты реализации параллельных и последовательных КИХ-фильтров в базисе ПЛИС с использованием перемножителей 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 Matlab/Simulink.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Computer Arithmetic: Algorithms and Hardware Designs (Oxford U. Press, 2nd ed., 2010, ISBN 978-0-19-532848-6).
2. Michael J.S. Smith. Application-Specific Integrated Circuits. VLSI Design Series. P.1040. ISBN 0-201-50022-1. LOC TK7874.6.S63. Addison Wesley Longman, http://www.awl.com.
3. Israel Koren. University of MASSACHUSETTS Dept. of Electrical & Computer Engineering. Digital Computer Arithmetic. ECE 666. Part 3. Sequential Algorithms for Multiplication and Division.
4. Uwe Meyer-Baese. Digital Signal Processing with Field Programmable Gate Arrays. Fourth Edition. http://www.springer.com/series/4748.
5. Xilinx. DSP: Designing for Optimal Results High-Performance DSP Using Virtex-4 FPGAs. DSP Products Advanced Design Guide. Edition 1.0. March 2005.
6. Уилкинсон, Б. Основы проектирования цифровых схем [Текст]: пер. с англ. / Б. Уилкинсон. - М.: Издательский дом Вильямс, 2004. - 320 с.
7. Армстронг, Дж. Р. Моделирование цифровых систем на языке VHDL [Текст]: пер. с англ. / Р. Дж. Армстронг. - М.: Мир, 1992. - 348 с.
8. Максфилд, К. Проектирование на ПЛИС: курс молодого бойца [Текст]: пер. с англ. / К. Максфилд. М.: Издательский дом Додэка XXI, 2007. 408 с.
9. Уэйкерли, Джон Ф. Проектирование цифровых устройств: пер. с англ. / Ф. Джон Уэйкерли. М.: Постмаркет, 2002. 533 с.
10. Рабаи, Ж.М. Цифровые интегральные схемы. Методология проектирования [Текст] / Ж.М. Рабаи, А. Чандракасан, Б. Николич. М.: Вильямс, 2007. - 911 с.
11. Угрюмов, Е.П. Цифровая схемотехника [Текст] / Е.П. Угрюмов. СПб.: БХВ, 2004. - 528 с.

12. Стешенко, В. ПЛИС фирмы ALTERA: проектирование устройств обработки сигналов [Текст] / В. Стешенко. М.: Додэка, 2000. - 457 с.

13. Ефремов, Н.В. Введение в систему автоматизированного проектирования Quartus II [Текст] / Н.В. Ефремов. М.: ГОУ ВПО МГУЛ, 2011. - 147 с.

14. Суворова, Е.А. Проектирование цифровых систем на VHDL [Текст] / Е.А. Суворова, Ю.Е. Шейнин. СПб.: БХВ-Петербург, 203. – 576 с.

15. Строгонов, А.В. Цифровая обработка сигналов в базисе программируемых логических интегральных схем [Текст] / А.В. Строгонов. Учебное пособие. – 2-е изд., испр. и доп. – СПб.: Издательство “Лань”, 2015. -310 с.

16. 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. - N12. - С.6-10.

17. 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. - N1. - С.94-100.

18. Строгонов, А.В. Проектирование цифровых фильтров в системе Matlab/Simulink и САПР ПЛИС Quartus [Текст] / А.В. Строгонов // Компоненты и технологии. - 2008. - N6. - С.32-36.

19. Строгонов, А.В. Проектирование параллельных КИХ-фильтров в базисе ПЛИС [Текст] / А.В. Строгонов, А.В. Быстрицкий // Компоненты и технологии. - 2013. - N6. - С.62-67.

20. Строгонов, А.В. КИХ-фильтр на распределенной арифметике: проектируем сами [Текст] / А.В. Строгонов, А.В. Быстрицкий // Компоненты и технологии. - 2013. - N3. - С.131-138.

21. Строгонов, А.В. КИХ-фильтры на параллельной распределенной арифметике: проектируем сами [Текст] / А.В.

Строгонов, А.В. Быстрицкий // Компоненты и технологии. 2013. - N5. - С.44-48.

22. Строгонов, А.В. Систолические КИХ-фильтры в базисе ПЛИС [Текст] / А.В. Строгонов, А.В. Быстрицкий // Компоненты и технологии. - 2013. - N8. - С.30-35.

23. Строгонов, А.В. Проектирование систолических КИХ-фильтров в базисе ПЛИС с помощью системымоделирования ModelSim-Altera [Текст] / А.В. Строгонов, А.В. Быстрицкий // Компоненты и технологии. - 2013. - N9. - С.24-28.

24. Строгонов, А.В. Эффективность разработки конечных автоматов в базисе ПЛИС FPGA [Текст] / А.В. Строгонов, А.В. Быстрицкий // Компоненты и технологии. - 2013. - N1. - С.66-72.

25. www.labfor.ru. Учебный лабораторный стенд LESO 2.1. Паспорт и Инструкция по эксплуатации. Новосибирск. 2009.

26. FIR Compiler. User Guide. Software Version: 11.0. May 2011. Altera Corporation.

27. 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. - N11. - С.96-102.

28. 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. - N2. С.38-43.

29. 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. - N5. - С.6-15.

30. 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. - N6. - С.146-152.

31. 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. - N7. - С.110-116

32. Строгонов А.В. Проектирование последовательных КИХ-фильтров 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. – 2015. - N11. - С.22-27.

33. Строгонов А.В. Проектирование КИХ-фильтров 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. – 2015. - N12. - С.22-27.

34. Строгонов А.В. Проектирование последовательных КИХ-фильтров в САПР ПЛИС Quartus II [Текст] / А.В. Строгонов, С.А. Цыбин, П.С. Городков П // Компоненты и технологии. – 2016. - N1. - С.10-15.

35. 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. - N4. - С.40-46.

ОГЛАВЛЕНИЕ

Учебное издание

Строгонов Андрей Владимирович

РЕАЛИЗАЦИЯ АЛГОРИТМОВ ЦИФРОВОЙ ОБРАБОТКИ СИГНАЛОВ В БАЗИСЕ ПРОГРАММИРУЕМЫХ ЛОГИЧЕСКИХ ИНТЕГРАЛЬНЫХ СХЕМ

В авторской редакции

Компьютерная верстка А.В. Строгонова

Подписано к изданию 26.10.2016.

Объем данных 9,5 Мб

ФГБОУ ВО «Воронежский государственный технический университет»

394026 Воронеж, Московский просп., 14