СОДЕРЖАНИЕ

Введение

Лабораторный практикум по дисциплине «Алгоритмические и аппаратные средства обработки измерительной информации» предназначены для проведения лабораторно-практических занятий с магистрантами направления 12.04.01 «Приборостроение».

Занятия проводятся в соответствии с рабочей программой дисциплины, разработанной в соответствии с ГОС ВО по направлению подготовки 12.04.01 «Приборостроение». Данный стандарт утвержден приказом № 957 от 22 сентября 2017 года.

Проведение занятий осуществляется при последовательном изучении тем дисциплины и представляет собой выполнение обучаемыми набора заданий предметной области дисциплины с целью выработки у них практических навыков их решения.

В ходе выполнения и защиты лабораторно-практических работ у обучаемых магистрантов формируется одна профессиональная компетенция ПКс-3: способность оформлять отчеты, статьи, рефераты по результатам проведенных научных исследований.

Индикаторами достижения данной компетенции являются полученные базовые знания, умения и приобретенный опыт для дальнейшей профессиональной деятельности магистрантов в области приборостроения:

- знает методы анализа, обработки и оформления данных, полученных при проведении измерительного эксперимента с использованием систем сбора на базе микропроцессорной техники и ПЛИС;

- умеет применять аппаратные и программные средства на традиционных компонентах и компонентах программируемой логики при проведении научных исследований;

- владеет опытом анализа и оформления данных измерений, необходимых для составления отчетов по результатам научных исследований в области приборостроения.

Перед проведением лабораторно-практического занятия преподаватель информирует обучаемых магистрантов о теме занятия, уделяет внимание вопросам методики обработки результатов работы на основе изученной информации на лекционных занятиях, сообщает о целях и задачах занятия, порядке его проведения и критериях оценки результатов выполненной работы.

По каждой лабораторно-практической работе магистрантам выдаются варианты заданий, и определяется необходимое время для их выполнения. После выполнения полученных заданий лабораторно-практическая работа оформляется в виде отчета, содержание которого конкретизируется по каждой работе индивидуально.

Оформленные отчеты по лабораторно-практическим работам представляются преподавателю, который проводит проверку правильности их выполнения. Если при проверке работы преподавателем выявлены существенные ошибки ее выполнения, то такая работа возвращается магистранту с указанием причин (ошибок) возврата.

Зачет по каждой лабораторно-практической работе ставится преподавателем после проведения собеседования, в ходе которого магистранту задаются несколько вопросов по тематике работы.

Ответы магистранта по каждой лабораторно-практической работе могут оцениваться по традиционной шкале («отлично», «хорошо», «удовлетворительно») с целью дальнейшего формирования объективной итоговой оценки по дисциплине.

Данные лабораторно-практические работы рассчитаны на использование современных персональных компьютеров в качестве лабораторного оборудования. При этом требуется, чтобы на компьютерах было установлено следующее программное обеспечение:

Операционная система Windows XP и выше;

Математический пакет Scilab версии 5.0 и выше;

Программный пакет PROTEUS версии 8.1 и выше;

Программный пакет Quartus II версии 13.0 и выше.

1. Анализ линейной дискретной системы во временной области

Краткая теория

Определения

Сигнал – функциональная зависимость одной физической величины от другой. Например, зависимость температуры некоторого объекта от времени. Зависимость напряжения в некоторой точке электрической схемы от времени также является сигналом.

Рассмотрим вначале одномерные сигналы, зависящие от времени, которые будем обозначать как $x(t)$. Почти весь материал, представленный далее, допускает обобщение и на многомерный случай.

Система – это некоторое устройство, состоящее из элементов (блоков), осуществляющее преобразование сигнала. Система преобразует входной сигнал $x(t)$ в выходной сигнал $y(t)$, т. е. $x(t) \to y(t)$.

Обычно все рассматриваемые системы инвариантны к сдвигу, т.е. если $x(t) \to y(t)$, то $x(t + T) \to y(t + T)$. Это означает, что форма выходного сигнала зависит только от входного сигнала, а не зависит от времени начала подачи входного сигнала. Далее мы будем рассматривать только такие системы.

Очень большое количество реальных систем можно считать инвариантными к сдвигу. Например, микрофон, переводящий сигнал «плотность воздуха» в сигнал «напряжение на его выходе», удовлетворяет этому свойству, если пренебречь изменением его свойств со временем.

Линейная система – это система, в которой выполняется следующее свойство:

если $x_1(t) \to y_1(t)$ и $x_2(t) \to y_2(t)$, то $Ax_1(t) + Bx_2(t) \to Ay_1(t) + By_2(t)$.

Большое количество реальных систем преобразования сигналов можно считать линейными. Например, микрофон является линейной системой (с достаточной степенью точности), так как если в него бу-

дут говорить одновременно 2 человека с разной громкостью, то электрический сигнал на выходе будет взвешенной суммой сигналов (от каждого человека в отдельности) на входе, а коэффициенты будут означать громкость разговора первого и второго человека. В дальнейшем мы будем рассматривать только линейные инвариантные к сдвигу системы.

Свойства линейных систем:

1. Постоянный (константный) сигнал переводится любой линейной системой в постоянный сигнал.
2. При прохождении через линейную систему синусоида остается синусоидой. Могут измениться лишь ее амплитуда и фаза (сдвиг во времени).

Второе свойство особенно важно, т.к. оно указывает на важнейший метод анализа линейных систем с помощью разложения входных и выходных сигналов на синусоиды (Фурье-анализ).

Что означает «прохождение синусоиды через линейную систему»? Это значит, что синусоида подается на вход системы бесконечно долго (время от нуля до бесконечности). Если в начальный момент времени сигнала на входе не было, а синусоиду начали подавать в некоторый конкретный момент времени, то после начала ее подачи на вход на выходе синусоида получается не сразу. Выходной сигнал постепенно начнет приобретать синусоидальную форму. Скорость «восстановления синусоиды» на выходе зависит от конкретной линейной системы.

Дискретные и непрерывные сигналы

Большинство реальных сигналов являются непрерывными функциями времени. Для обработки на компьютере требуется перевести сигналы в цифровую форму. Один из способов сделать это – равномерно по времени измерить значения сигнала на определенном про-

межутке времени и ввести полученные значения сигнала в компьютер. Если делать измерения часто, то по полученному *дискретному сигналу* можно будет достаточно точно восстановить вид исходного непрерывного сигнала.

Процесс замера величины сигнала через равные промежутки времени называется *дискретизацией*. Многие устройства для ввода данных в компьютер осуществляют дискретизацию. Например, звуковая карта осуществляет дискретизацию сигнала с микрофона, сканер осуществляет дискретизацию сигнала, поступающего с фотоэлемента. В результате дискретизации непрерывный (*аналоговый*) сигнал переводится в последовательность чисел. Устройство, выполняющее этот процесс, называется *аналогово-цифровым преобразователем* (АЦП, analogue-to-digital converter, ADC). Частота, с которой АЦП производит замеры аналогового сигнала и выдает его цифровые значения, называется *частотой дискретизации*.

При осуществлении процедуры АЦП возникает вопрос, при каких условиях на исходный сигнал и на частоту дискретизации можно с необходимой степенью точности восстановить исходный сигнал по его цифровым значениям. Ответ на этот вопрос дает теорема Котельникова (Найквиста, Шеннона). Однако чтобы ее понять, необходимо познакомиться с понятием *спектра непрерывного сигнала*.

Как известно из математического анализа, любая непрерывная функция разлагается в ряд Фурье. Смысл этого разложения состоит в том, что функция представляется суммой синусоид кратных частот с различными амплитудами. Коэффициенты (амплитуды) при синусоидах, отображенные на оси частот, называются *спектром* функции. У гладких функций спектр быстро убывает (с ростом частоты коэффициенты быстро стремятся к нулю). Для относительно «ломаных» функций спектр убывает медленно, т. к. для представления разрывов и «изломов» функции нужны синусоиды с большими частотами.

Говорят, что сигнал имеет ограниченный спектр, если выше определенной частоты все коэффициенты спектра равны нулю. В этом случае говорят, что спектр сигнала лежит ниже частоты F (ограничен частотой F), где F – частота, выше которой спектр равен нулю.

Теорема Котельникова (Найквиста, Шеннона): если сигнал таков, что его спектр ограничен частотой F, то после дискретизации сигнала с частотой не менее 2F можно восстановить исходный непрерывный сигнал по цифровому сигналу практически точно. Для этого нужно сделать интерполяцию цифрового сигнала «между отсчетами» специального вида функциями (sinc-функциями).

На практике эта теорема имеет огромное значение. Например, известно, что большинство звуковых сигналов можно с некоторой степенью точности считать сигналами с ограниченным спектром. Их спектр, в основном, лежит ниже 20 кГц. Это значит, что при дискретизации с частотой не менее 40 кГц мы можем потом весьма точно восстановить исходный аналоговый звуковой сигнал по его цифровым отсчетам. Абсолютной точности достичь не удастся, так как в природе не бывает сигналов с идеально ограниченным спектром.

Устройство, которое интерполирует дискретный сигнал до непрерывного, называется *цифро-аналоговым преобразователем* (ЦАП, digital-to-analogue converter, DAC). Эти устройства применяются, например, в проигрывателях компакт-дисков для восстановления звука по цифровому звуковому сигналу, записанному на компакт-диск. Частота дискретизации звукового сигнала при записи на компакт-диск составляет 44100 Гц.

Импульсная характеристика

Далее рассматриваются только линейные дискретные системы (ЛДС), то есть системы, работающие с дискретными сигналами. На вход такой системы подается последовательность чисел $x[n]$ (*дискретный сигнал*), и на выходе получается последовательность чисел

$y[n]$ (рис. 1.1). Свойства линейности для дискретных систем формулируются практически так же, как и для непрерывных линейных систем.

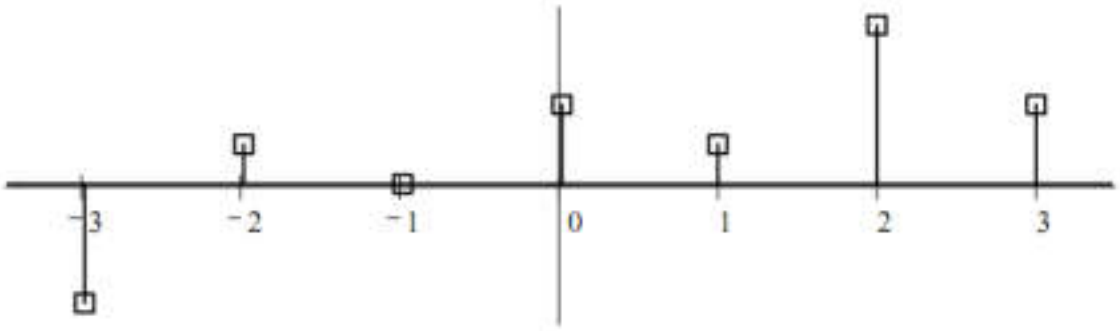


Рис. 1.1. Пример дискретного сигнала: ось абсцисс – дискретные моменты времени, ось ординат – значения сигнала в эти моменты времени

Рассмотрим, каким образом ЛДС может преобразовывать входной сигнал в выходной. Для этого рассмотрим реакцию этой системы на цифровую дельта-функцию (функцию Кронекера).

Функция Кронекера – это сигнал вида: $\delta[n]=\begin{cases}1, n=0\\0, n\neq 0\end{cases}$, т. е. короткий единичный импульс (рис. 1.2).

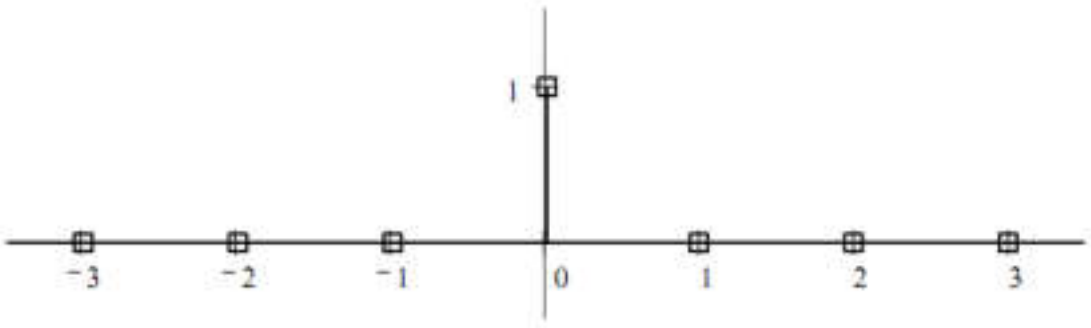


Рис. 1.2. Функция Кронекера

Очевидно, что любой дискретный сигнал можно разложить в сумму таких функций, сдвинутых во времени.

Исследуем отклик (выходной сигнал) ЛДС на цифровую дельта-функцию. Для этого используем дельта-функцию, как входной сигнал ЛДС, и измерим выходной сигнал. Пусть выходной сигнал представлен функцией $h[n]$, т.е. $\delta[n] \rightarrow h[n]$ (рис. 1.3).

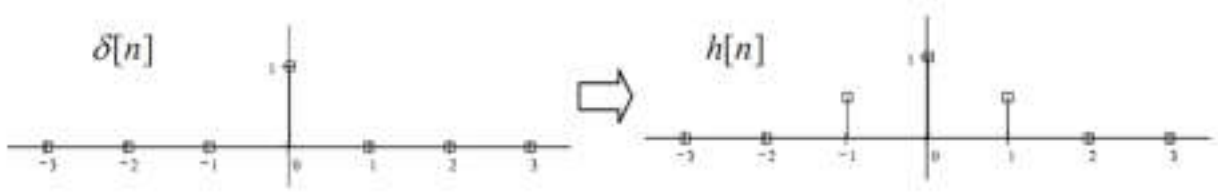


Рис. 1.3. Отклик ЛДС на цифровую дельта-функцию

Если известна функция $h[n]$ (отклик ЛДС на дельта-функцию), то можно вычислить отклик ЛДС на любой входной сигнал. Так как любой входной сигнал является линейной комбинацией сдвинутых во времени дельта-функций, то выходной сигнал будет также линейной комбинацией сдвинутых во времени функций $h[n]$. Это следует из линейности системы и инвариантности к сдвигу по времени. Сигнал $h[n]$ называется *импульсной характеристикой* (*impulse response*) ЛДС или ее откликом на единичный входной импульс.

Свёртка

Существует несколько способов вычисления отклика линейной системы на произвольное воздействие. Каждая точка сигнала превращается в функцию h (сдвинутую в нужную позицию и умноженную на величину сигнала в данной точке), а потом все эти функции складываются.

$$y[n] = \sum_{k=-\infty}^{+\infty} x[n-k] \cdot h[k]. \qquad (1.1)$$

Другой способ состоит в вычислении значения каждой точки в результирующем сигнале как взвешенной суммы некоторого множества соседних точек исходного сигнала. Коэффициенты этой суммы

совпадают с импульсной характеристикой линейной системы, перевернутой относительно точки «ноль». Для одномерного случая:

$$y[n] = \sum_{k=-\infty}^{+\infty} h[n-k] \cdot x[k]. \qquad (1.2)^1$$

В общем случае операция получения выходного сигнала по исходному сигналу получила название *свёртки* (*convolution*). Любая линейная система осуществляет свёртку входного сигнала со своей импульсной характеристикой. Это записывается так: $y[n] = x[n] \cdot h[n]$. Функция $h[n]$ называется *ядром свёртки* (*kernel*) или импульсной характеристикой.

Все реальные сигналы имеют конечную длительность (т.е. отличны от нуля лишь на конечном отрезке времени). Кроме того, реакция реальной системы на входное воздействие не может возникнуть раньше самого входного воздействия.

Свойства свёртки:

1. $x[n] \cdot h[n] = h[n] \cdot x[n]$ (т.е. можно переставлять местами исходный сигнал и ядро свёртки);
2. $(x[n] \cdot y[n]) \cdot z[n] = x[n] \cdot (y[n] \cdot z[n])$ (т.е. вместо того, чтобы проводить свёртку по очереди в разных системах, можно получить систему с ядром $(y[n] \cdot z[n])$, которая является суперпозицией $y[n]$ и $z[n]$);
3. $x[n] \cdot y[n] + x[n] \cdot z[n] = x[n] \cdot (y[n] + z[n])$.

---

[1] Выходные отклики в различные моменты времен определяются так:

$y[0] = x[0] \cdot h[0]$

$y[1] = x[0] \cdot h[1] + x[1] \cdot h[0]$

$y[2] = x[0] \cdot h[2] + x[1] \cdot h[1] + x[2] \cdot h[0]$

$y[3] = x[0] \cdot h[3] + x[1] \cdot h[2] + x[2] \cdot h[1] + x[3] \cdot h[0]$

…

$y[n] = x[0] \cdot h[n] + x[1] \cdot h[n\text{-}1] + \ldots + x[n] \cdot h[0]$

Задание: в программе Scilab для своего варианта вычислить реакцию ЛДС в виде 20 отсчетов при нулевых начальных условиях в режиме калькулятора и с использованием программы. Построить графики импульсной характеристики, воздействия и реакции.

Содержание отчета:

- листинги программы;
- графики импульсной характеристики, воздействия и реакции;
- выводы по работе.

Вариант		**Значения отсчетов**									
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1	*h(n)*	2	3	4	3	3	2	1	1	0	1
	x(n)	4	3	2	2	1	0	2	3	1	0
2	*h(n)*	2	2	3	3	3	4	4	2	2	0
	x(n)	5	4	3	2	1	0	1	2	3	4
3	*h(n)*	3	2	3	3	5	4	3	3	0	1
	x(n)	2	3	2	3	4	5	2	4	2	0
4	*h(n)*	1	3	2	2	3	3	4	3	1	0
	x(n)	5	1	3	5	2	1	1	0	1	2
5	*h(n)*	2	1	2	3	4	4	3	2	1	1
	x(n)	3	2	1	0	1	1	2	2	3	3
6	*h(n)*	1	3	3	1	4	3	3	1	2	0
	x(n)	5	4	3	0	2	1	4	0	5	2
7	*h(n)*	3	1	2	2	4	4	2	2	1	1
	x(n)	1	2	0	3	5	2	3	1	0	2
8	*h(n)*	2	2	3	3	3	4	4	2	1	0
	x(n)	3	1	2	1	4	3	1	5	0	2
9	*h(n)*	1	2	3	4	5	2	3	4	5	1
	x(n)	4	0	1	2	3	4	2	3	0	3
10	*h(n)*	3	3	2	1	1	2	3	4	2	0
	x(n)	2	4	3	5	0	1	1	3	1	2
11	*h(n)*	1	1	2	2	2	3	3	1	1	1
	x(n)	3	5	4	0	3	2	1	2	0	3

Вариант		Значения отсчетов									
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
12	*h(n)*	2	1	2	3	1	4	5	4	3	1
	x(n)	4	2	3	0	1	2	4	2	1	2
13	*h(n)*	3	0	2	4	4	3	1	0	1	1
	x(n)	5	3	1	2	3	4	2	1	2	0

Контрольные вопросы

1. Какими свойствами обладает одномерный сигнал?
2. Какую функцию выполняет линейная система?
3. Какими свойствами обладает линейная система?
4. Чем отличается дискретный сигнал от непрерывного сигнала?
5. Какими свойствами должна обладать частота дискретизации?
6. Что собой представляет спектр функции?
7. Каково значение теоремы Котельникова?
8. Каким образом получают импульсную характеристику?
9. Что собой представляет операция свёртки сигнала?

2. Анализ линейной дискретной системы в частотной области

Краткая теория

Преобразование Фурье

Многие реальные сигналы удобно анализировать, раскладывая их на простейшие гармонические составляющие (спектр сигнала), так как гармоники являются «собственными функциями» линейных систем (они проходят через линейные системы без изменения своей формы, меняя лишь фазу и амплитуду).

Преобразование Фурье (*Fourier transform*) – это разложение функций на гармонические составляющие, представляющие собой частотный спектр. Существует несколько видов преобразования Фурье.

1. Интеграл Фурье непериодического непрерывного сигнала.
2. Бесконечный ряд Фурье периодического непрерывного сигнала.
3. Интеграл Фурье непериодического дискретного сигнала.
4. Конечный ряд Фурье периодического дискретного сигнала.

Спектральный анализ предполагает использование компьютера, который способен работать только с ограниченным объемом данных, поэтому реальны вычисления только последнего вида преобразования Фурье. Рассмотрим этот вид преобразования более подробно.

Дискретное преобразование Фурье вещественного сигнала

Представим, что некоторый дискретный сигнал $x[n]$ имеет период N точек. В этом случае его можно представить в виде конечного ряда дискретных гармонических составляющих:

$$x[n] = \sum_{k=0}^{N/2} C_k \cos \frac{2\pi k(n+\varphi_k)}{N}. \qquad (2.1)$$

С точки зрения вычисления этот конечный ряд удобно представить в другой форме, которая является эквивалентной записью предыдущей формулы, но не содержит начальные фазы гармоник:

$$x[n] = \sum_{k=0}^{N/2} A_k \cos \frac{2\pi kn}{N} + \sum_{k=0}^{N/2} B_k \sin \frac{2\pi kn}{N}. \qquad (2.2)$$

Коэффициенты A_k и B_k вычисляются в виде суммы произведений дискретных сигналов и их базисных функций:

$$A_k = \frac{2}{N} \sum_{i=0}^{N-1} x[i] \cos \frac{2\pi ki}{N}, \qquad (2.3)$$

$$B_k = \frac{2}{N}\sum_{i=0}^{N-1} x[i]\sin\frac{2\pi ki}{N}. \qquad (2.4)$$

Вычисление коэффициентов A_k и B_k по формулам (2.3) и (2.4) требует значительного числа умножений и вычисления тригонометрических функций. Есть более скоростной метод выполнения этих вычислений, который называется *быстрым преобразованием Фурье* (БПФ, FFT). Он основан на том, что среди множителей есть много повторяющихся значений. По этому методу слагаемые с одинаковыми множителями группируются вместе, значительно сокращая число умножений.

Комплексное дискретное преобразование Фурье

Выводы, полученные при рассмотрении действительных сигналов, можно обобщить на случай комплексных сигналов. Предположим, что известен исходный комплексный сигнал $x[n]$, где n=0,…, N-1. Тогда его комплексный спектр, состоящий из N комплексных чисел, имеет вид

$$X[k] = \sum_{n=0}^{N-1} x[n]\cdot e^{-jnk(2\pi/N)}. \qquad (2.5)$$

Разложение в спектр действительного сигнала по формуле (2.5) имеет весьма интересную для практики особенность. Первые $N/2+1$ комплексных коэффициентов спектра будут совпадать со спектром действительного дискретного преобразования Фурье, а остальные коэффициенты представляют их симметричное отражение относительно половинной частоты дискретизации.

Задание: по варианту предыдущей работы для заданного входного сигнала $x(n)$ определить значения амплитудного и фазового откликов, используя формулу дискретного преобразования Фурье (2.5).

Построить графики амплитудного и фазового откликов. Амплитудный отклик определяют как модуль комплексного числа с учетом формулы Эйлера:

$$e^{ix} = \cos x + i \sin x.$$

Содержание отчета:

- листинги программы;
- графики импульсной характеристики, воздействия и реакции;
- выводы по работе.

Контрольные вопросы

1. Что собой представляет преобразование Фурье?
2. Какие виды преобразования Фурье используют на практике?
3. Что собой представляет дискретное преобразование Фурье?
4. Что такое фаза гармоники?
5. Каково назначение базисных функций?
6. Каковы особенности у быстрого преобразования Фурье?
7. Что собой представляет комплексное дискретное преобразование Фурье и комплексный спектр?

3. Проектирование нерекурсивных КИХ-фильтров при помощи весовых оконных функций

Краткая теория

Осуществить качественную фильтрацию сигналов измерительных датчиков в условиях индустриальных помех чрезвычайно важно для современной высокоточной измерительной техники.

Однако получить селективные характеристики, удовлетворяющие современным требованиям по качественной фильтрации, с помощью традиционных *аналоговых фильтров* весьма проблематично.

Бурное развитие цифровой техники, происходящее в настоящее время, отразилось и на реализации устройств частотной селекции, то есть фильтров. Были разработаны цифровые устройства частотной селекции, которые получили название цифровых фильтров.

В общем случае под *цифровым фильтром* понимают аппаратную или программную реализацию специального математического алгоритма. В качестве входного воздействия берется некоторый исходный цифровой сигнал, а выходным откликом является другой цифровой модифицированный сигнал. При этом в выходном отклике изменена определенным образом *амплитудная* и *фазовая* характеристики.

Если говорить о назначении фильтров, то в общем понимании *фильтр* можно рассматривать как устройство, пропускающее или подавляющее частоты определенного диапазона, которые входят в состав спектра некоторого исходного входного сигнала.

По выполняемым функциям принято различать четыре основных типа фильтров частотной селекции: *низкочастотный* (НЧ), *высокочастотный* (ВЧ), *полосовой* (П) и *режекторный* (Р).

Фильтр нижних частот (ФНЧ) пропускает все частоты ниже некоторой *частоты среза полосы пропускания* f_{c1} и подавляет все частоты, превышающие некоторую другую частоту f_{c2}, которая называется *частотой среза зоны непрозрачности* фильтра.

Амплитудно-частотную характеристику (АЧХ) идеального ФНЧ можно представить в виде графика, показанного на рис.3.1.

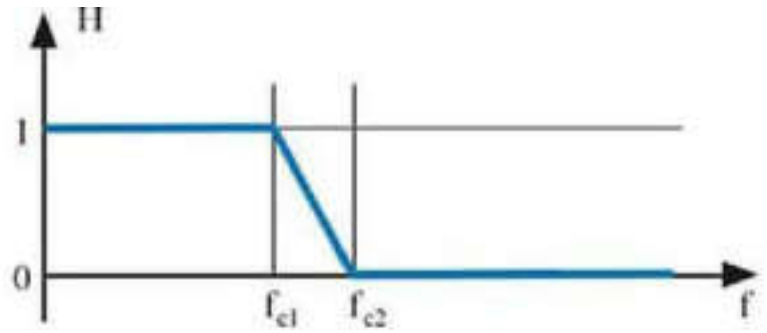


Рис. 3.1. АЧХ идеального ФНЧ

Для частот среза должно выполняться условие $f_{c1} < f_{c2}$. При этом, чем меньше их разность, тем больше показатель прямоугольности:

$$\alpha = \frac{f_{c1}}{f_{c2} - f_{c1}}, \tag{3.1}$$

а чем меньше f_{c1}, тем больше показатель узкополосности фильтра:

$$\beta = \frac{f_{Д}}{f_{c1}}, \tag{3.2}$$

где $f_{Д}$ – частота дискретизации входного сигнала.

Фильтр верхних частот (ФВЧ) противоположен ФНЧ по своим частотным свойствам. Он пропускает все частоты выше заданной частоты среза полосы пропускания f_{c2} и подавляет частоты ниже частоты среза зоны непрозрачности f_{c1} (рис. 3.2).

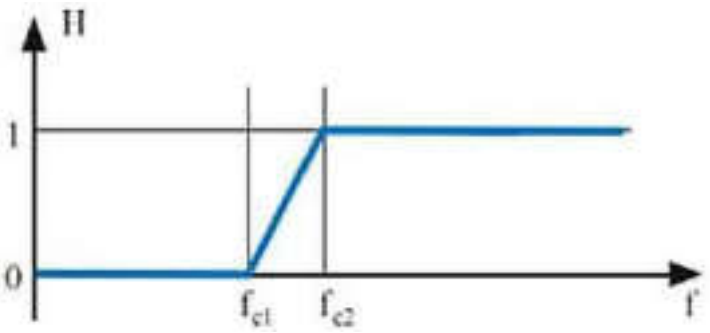


Рис. 3.2. АЧХ идеального ФВЧ

Полосовой фильтр (ПФ) можно представить последовательным соединением ФВЧ и ФНЧ – он пропускает только сигнал определенной полосы частот и подавляет оставшуюся часть сигнала (рис. 3.3).

Фактически это означает, что он характеризуется двумя частотами пропускания и двумя частотами подавления и имеет соответственно две зоны подавления и одну зону пропускания.

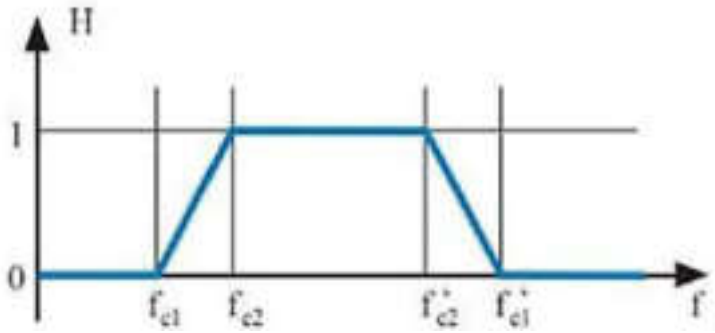


Рис. 3.3. АЧХ идеального ПФ

Режекторный фильтр (РФ) можно представить как противоположность полосовому фильтру с точки зрения пропускания и подавления сигналов различных частот.

Этот фильтр позволяет подавить частоты сигналов из определенного диапазона, пропуская на выход все остальные сигналы частот вне этого диапазона (рис. 3.4).

Для РФ можно сказать, что он, так же как и ПФ, характеризуется двумя частотами пропускания и двумя частотами подавления и имеет соответственно одну зону подавления и две зоны пропускания.

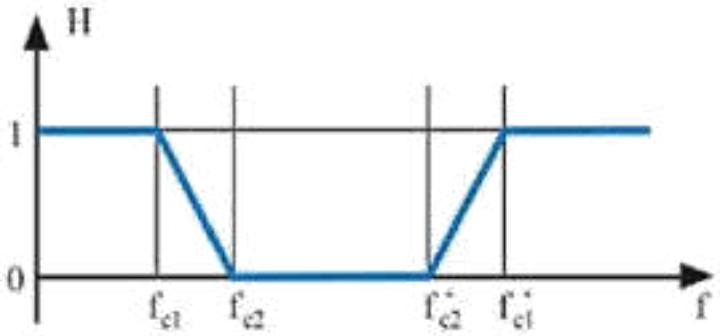


Рис. 3.4. АЧХ идеального РФ

В таблице 3.1 представлены формулы для расчета импульсных характеристик идеальных фильтров различных типов (f_c, f_1, f_2 – нормированные частоты краев полос пропускания или заграждения).

Таблица 3.1

Тип фильтра	$h(n)$, $n\neq0$	$h(0)$
ФНЧ	$\frac{1}{n\pi}\sin(\omega_c n)$	$2f_c$
ФВЧ	$-\frac{1}{n\pi}\sin(\omega_c n)$	$1\text{-}2f_c$
ПФ	$\frac{1}{n\pi}\sin(\omega_2 n)-\frac{1}{n\pi}\sin(\omega_1 n)$	$2(f_2\text{-}f_1)$
РФ	$\frac{1}{n\pi}\sin(\omega_1 n)-\frac{1}{n\pi}\sin(\omega_2 n)$	$1\text{-}2(f_2\text{-}f_1)$

Характеристики идеальных фильтров не достигаются в реальности. Всегда существует погрешность, которая выражается в том, что коэффициент передачи фильтра в зоне непрозрачности не равен 0, а в полосе пропускания – не равен 1. На рисунке 3.5 показаны графики, иллюстрирующие разницу между АЧХ идеального и реального ФНЧ.

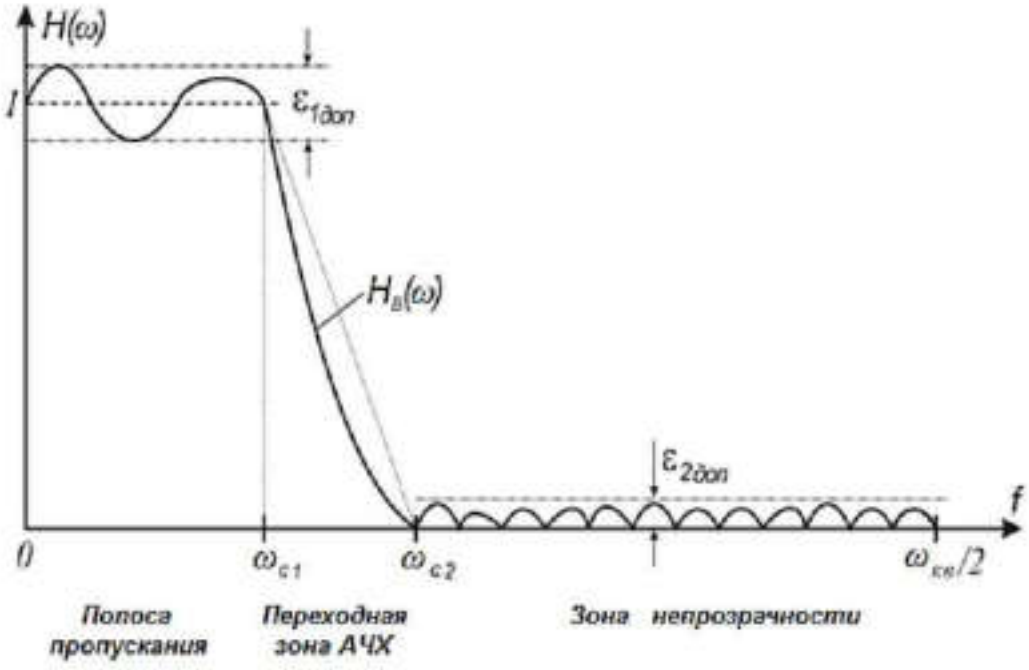


Рис. 3.5. Графики АЧХ идеального $H(\omega)$ и реального $H_B(\omega)$ ФНЧ

Из вышесказанного следует, что перед тем как начать проектирование фильтра, нужно задать допустимые уровни погрешности воспроизведения желаемой АЧХ: $\varepsilon_{1доп}$ – допустимая неравномерность АЧХ фильтра в полосе пропускания, $\varepsilon_{2доп}$ – допустимый уровень боковых лепестков в зоне непрозрачности.

Уровни допустимой погрешности воспроизведения желаемой АЧХ обычно задают в логарифмических единицах (децибелах). Например, подавление в зоне непрозрачности, равное 60 $дБ = 20lg10^3$, означает, что выходной сигнал ослабляется в $10^{60/20} = 1000$ раз, а неравномерность АЧХ в полосе пропускания $0{,}1дБ$ соответствует изменчивости его амплитуды не более чем в $10^{0,1/20} \approx 1{,}01158$ раз.

Подводя итог описанию краткой теории, можно сказать, что при разработке цифрового фильтра исходными данными являются:

- частота дискретизации – $f_Д$;
- частоты среза полосы пропускания – f_{c1}, f'_{c1} и частоты среза зоны непрозрачности – f_{c2}, f'_{c2} (это справедливо для ФНЧ и РФ; для ФВЧ и ПФ частоты среза меняются местами);
- допустимый уровень неравномерности АЧХ в полосе пропускания – $\varepsilon_{1доп}$ (0,1-0,5 дБ);
- допустимый уровень боковых лепестков АЧХ в зоне непрозрачности – $\varepsilon_{2доп}$ (40-120 дБ).

Порядок N фильтра с конечной импульсной характеристикой (КИХ) можно оценить по формуле:

$$N = \alpha\beta L\{\varepsilon_{1доп}, \varepsilon_{2доп}\}, \tag{3.3}$$

где L – логарифмический показатель частотной избирательности,

$$L = \frac{2}{3}\lg\{10 \cdot \varepsilon_{1доп} \cdot \varepsilon_{2доп}\}. \tag{3.4}$$

Проектирование цифрового фильтра заключается в расчете значений его коэффициентов. В данной лабораторной работе осуществляется проектирование цифрового фильтра с конечной импульсной характеристикой (FIR – finite impulse response).

Устройства такого типа при подаче на их вход единичного импульса на выходе имеют конечное число ненулевых отсчетов $h(n)$, $n = 0 .. N-1$. Чем больше порядок фильтра N, тем более продолжительной будет его реакция на входное воздействие.

Входное воздействие $x(k)$ и выходной отклик $y(k)$ КИХ-фильтра связаны между собой следующим выражением:

$$y(k) = x(k-n) \cdot h(n) = \sum_{n=0}^{N} x(k-n) \cdot h(n). \qquad (3.5)$$

Высокий порядок фильтра требует больших вычислительных затрат и ресурсов оперативной памяти. Однако чем больше порядок N, тем ближе реальные частотные характеристики фильтра к их идеальным частотным характеристикам.

Порядок фильтра для идеальной амплитудно-частотной характеристики бесконечен. Поэтому при решении практических задач обработки сигналов импульсная характеристика должна иметь конечную длину. Наиболее простой способ получить конечную импульсную характеристику из идеальной бесконечной характеристики – это умножить ее на специальную весовую оконную функцию.

Разработан целый ряд аналитических оконных функций, каждая из которых дает свое приближение к идеальной логарифмической амплитудно-частотной характеристике (ЛАЧХ) фильтра. В таблице 3.2 приведены аналитические выражения для весовых оконных функций, используемых в данной работе.

Таблица 3.2

Наименование функции	Аналитическое выражение
Прямоугольное окно	$w(n) = \begin{cases} 1, \text{при } 0 \le n \le N-1, \\ 0 \quad\quad \text{при} \quad n \ge N \end{cases}$
Окно Хэмминга	$w(n) = \begin{cases} 0.54 - 0.46\cos\dfrac{2\pi n}{N-1}, \text{при } 0 \le n \le N, \\ 0 \quad\quad \text{во всех других случаях} \end{cases}$
Окно Хэннинга	$w(n) = \begin{cases} 0.5\left(1 - \cos\dfrac{2\pi k}{n-1}\right), \text{где } k = 1..n-1, \\ 0 \quad\quad \text{во всех других случаях} \end{cases}$

Ход выполнения работы

1. Используя формулу (3.3) и данные варианта, рассчитать порядок фильтра.
2. Средствами SCILAB получить коэффициенты фильтра $h(n)$, $n = 0..N-1$, воспроизводящего желаемую частотную характеристику с заданной точностью оконными методами, которые представлены в таблице 3.2.
3. Построить АЧХ для каждого оконного метода и определить значения неравномерности АЧХ, уровни затухания в зоне непрозрачности и перерегулирования. Результаты расчетов представить в виде таблицы.
4. Для всех оконных методов увеличить, а затем уменьшить порядок фильтра на ±20. Сделать выводы по изменению его АЧХ.

Варианты индивидуальных заданий

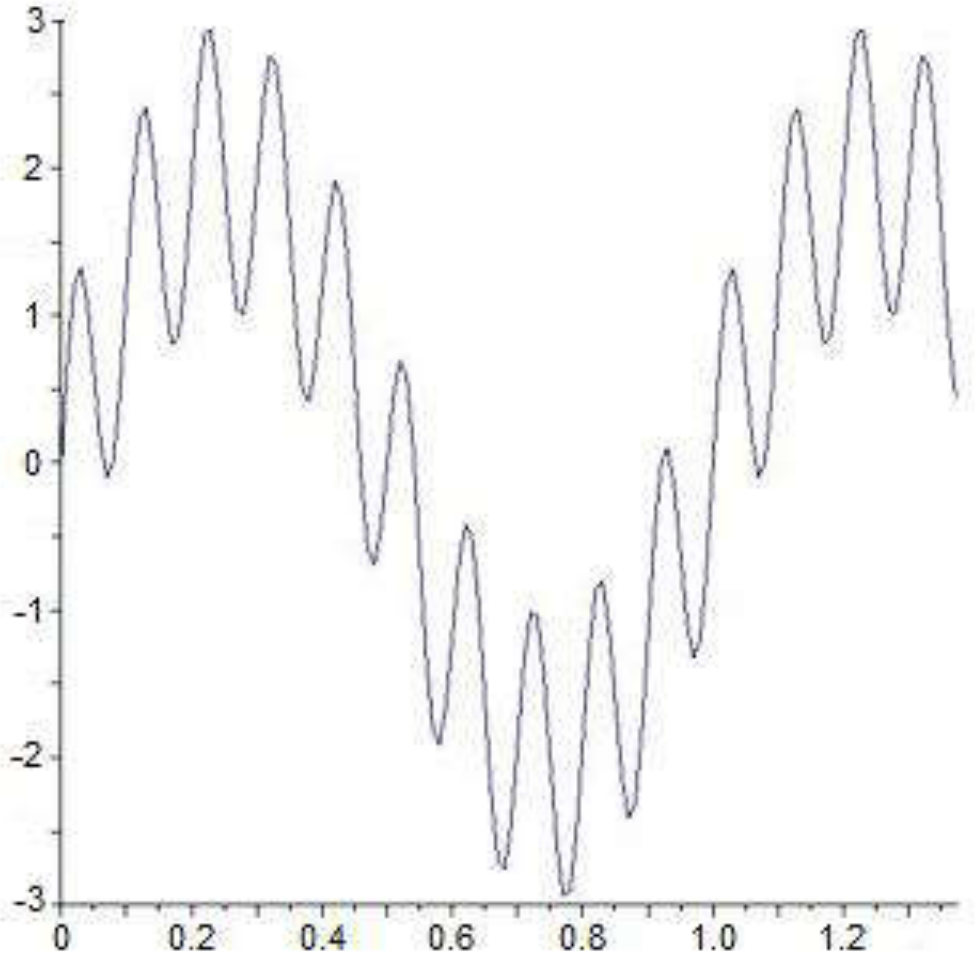
№	Тип ЦФ	$f_Д$, Гц	Частота подавления, Гц	Частота пропускания, Гц	Уровень подавления ($\varepsilon_{2доп}$), дБ	Неравномерность полосы пропускания, ($\varepsilon_{1доп}$), дБ
1	НЧ	1000	150	100	70	0,5
2	П	2000	150±100	150±50	40	0,5
3	ВЧ	1000	100	150	70	0,5
4	Р	2000	150±50	150±100	40	0,5
5	НЧ	1000	100	50	60	0,5
6	Р	2000	250±50	250±150	80	0,5
7	ВЧ	1000	50	100	60	0,5
8	П	2000	250±150	250±50	80	0,5
9	НЧ	1000	150	50	140	0,5
10	П	2000	175±125	175±25	70	0,5
11	ВЧ	1000	50	150	140	0,5
12	Р	2000	175±25	175±125	70	0,5
13	НЧ	1000	75	25	80	0,5
14	П	2000	200±125	200±50	60	0,5
15	ВЧ	1000	25	75	80	0,5
16	Р	2000	200±50	200±125	60	0,5
17	НЧ	1000	50	25	40	0,5
18	П	2000	250±225	250±150	60	0,5
19	ВЧ	1000	25	50	40	0,5

Контрольные вопросы

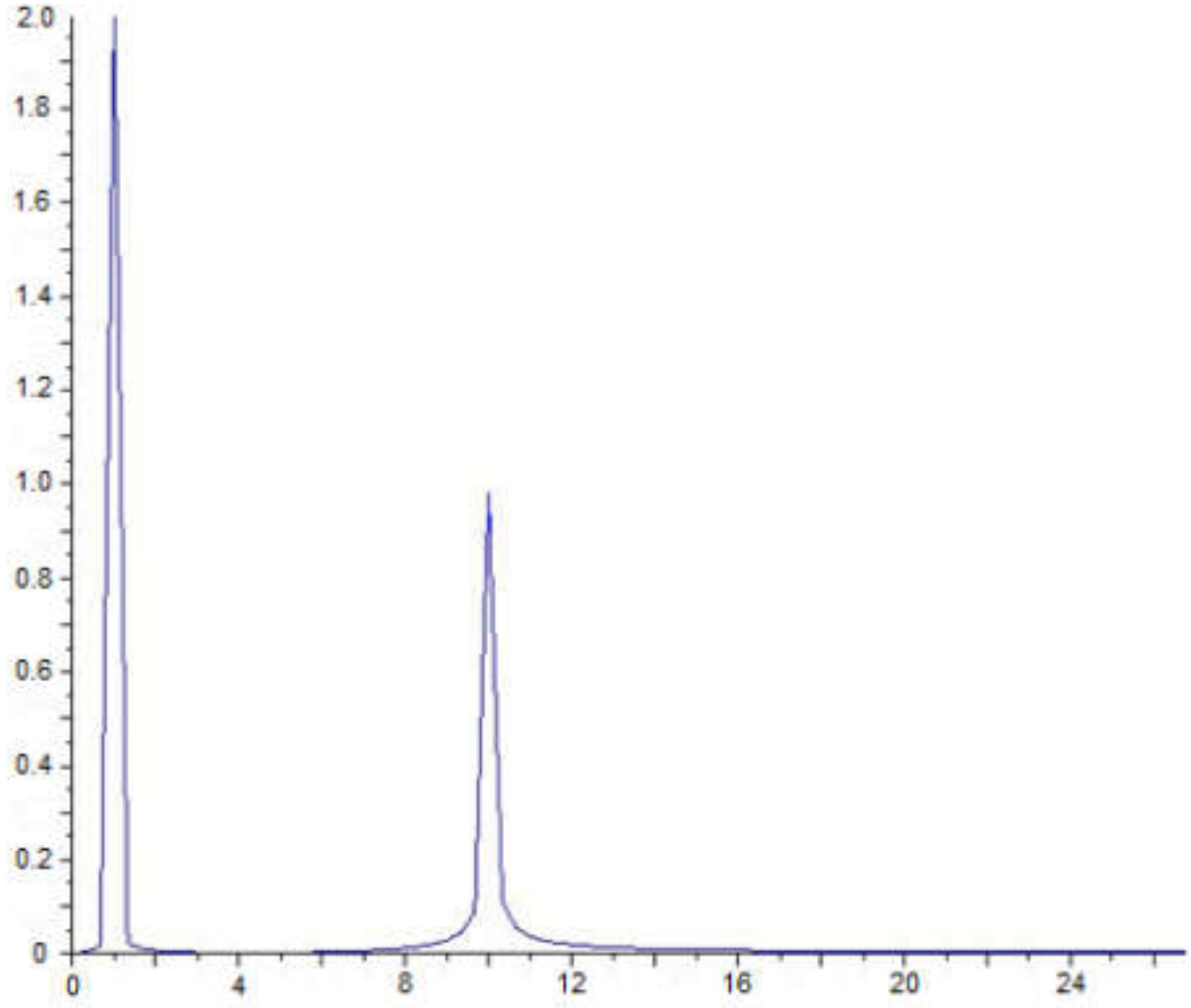
1. Каковы определения терминов: полоса пропускания, зона подавления, частота дискретизации, показатели частотной избирательности?
2. Что собой представляет КИХ-фильтр?
3. Какие виды цифровых фильтров вы знаете?
4. Какие данные необходимы для разработки цифрового фильтра?
5. В чем суть оконного метода расчета коэффициентов КИХ-фильтра?

Расчет ВЧ КИХ-фильтра в SCILAB

```
Fs = 100;                    // частота дискретизации сигнала 100 Hz
t = 0:1/Fs:1;                //временной интервал дискретизации
n = length(t);
f = linspace(0,Fs,n);        // вектор частот
x1 = sin(2*%pi*1*t);         // 1 Hz синусоида
x2 = sin(2*%pi*10*t);        // 10 Hz синусоида
x = 2*x1 + x2;               // Сумма 1 Hz и 10 Hz синусоид
plot(t,x);                   // входной сигнал во временной области
```




```
X = fft(x)./(length(x)/2);          // спектр сигнала
plot(f(1:n/2),abs(X(1:n/2)));       // вывод графика спектра сигнала
```



Постановка задачи. Пусть необходимо отфильтровать сигнал $x2$, представляющий собой синусоиду 10 Гц (то есть подавить сигнал $x1$ синусоиды 1 Гц). Для этого, например, можно применить высокочастотный фильтр порядка N=21.

В программе Scilab значение частоты среза нормируется на значение частоты дискретизации. Так как фильтруется дискретный сигнал, то возможные значения нормированной частоты среза будут расположены в диапазоне от 0 до 0,5.

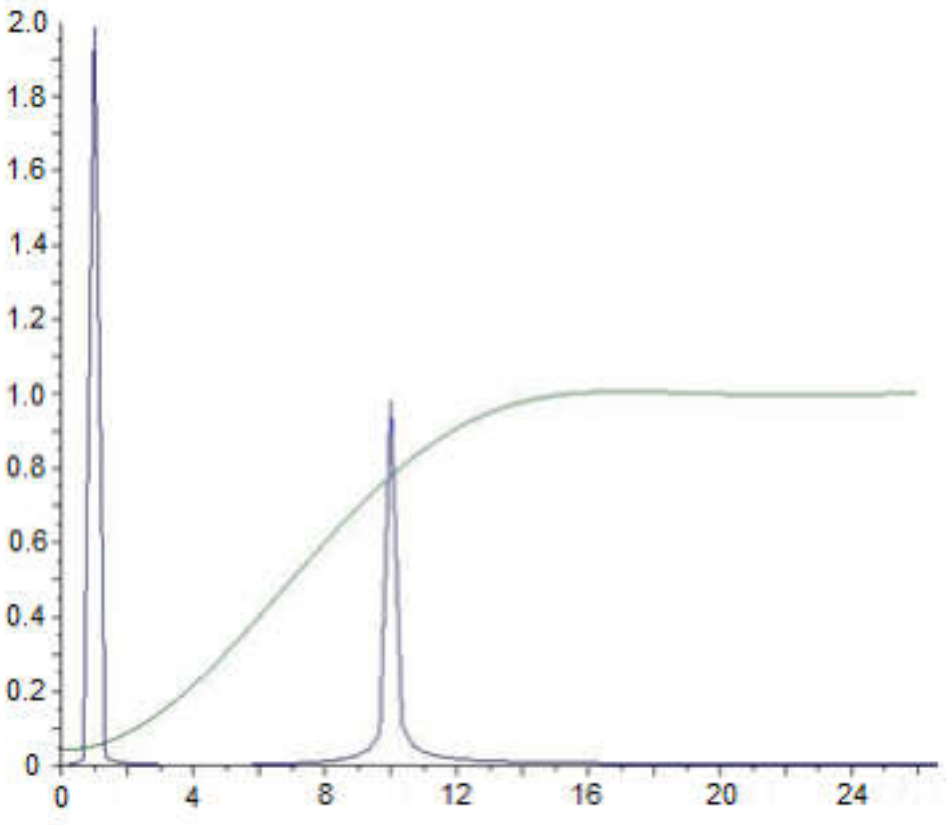
Например, синусоиде $x2$ с частотой 10 Гц при частоте дискретизации 100 Гц соответствует значение 0,1 (10/100), а синусоиде с частотой 1 Гц при этой же частоте дискретизации соответствует значение 0,01 (1/100).

Для выделения полезного сигнала в фильтрах используют частоту среза, составляющую примерно 70% от частоты среза полосы пропускания (в данном примере это значение равно 0,07).

```
// синтез высокочастотного КИХ-фильтра
[valcoeff, filtamp, filtfreq] = wfir ('hp', 21, [.07 0], 'hn', [0 0]);
```

Параметр ('hp') – тип фильтра (ФВЧ), параметр (21) – порядок фильтра и последние параметры – нормированные частоты среза (для 'lp' и 'hp' необходимо использовать только первое значение).

```
//вывод импульсной характеристики фильтра и спектра сигнала
fr2 = filtfreq.*Fs;
plot(f(1:n/2),abs(X(1:n/2)),fr2,filtamp);
```




```
//вывод значений коэффициентов фильтра в командное окно
disp('Коэффициенты фильтра:')
disp( valcoeff);
```

Коэффициенты фильтра:

column 1 to 6

0.0000000 0.0006309 0.0013987 -0.0005885 -0.0088300 -0.0257518

column 7 to 12

-0.0511616 -0.0815882 -0.1109208 -0.1322131 0.86000 -0.1322131

column 13 to 17

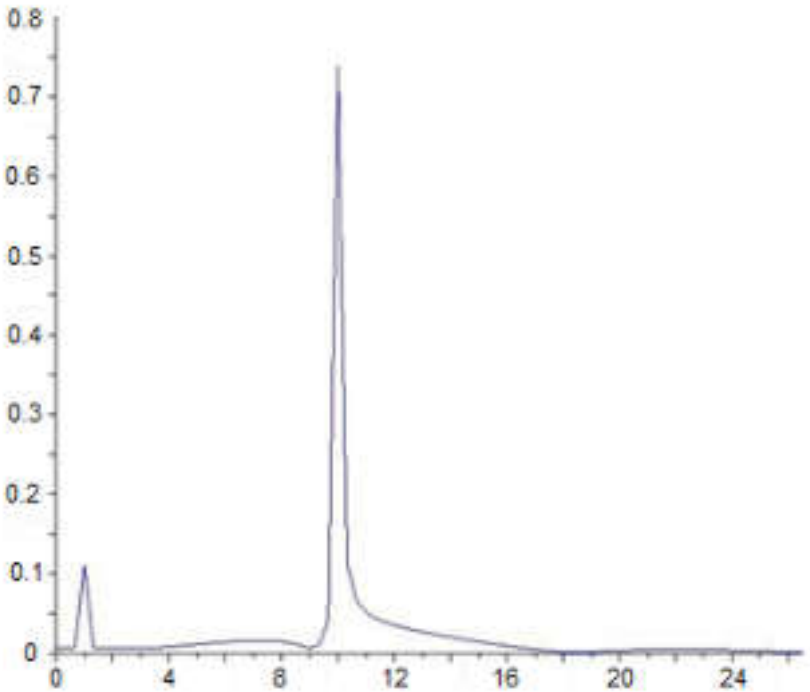
- 0.0257518 0.1109208 -0.0815882 -0.0511616 - 0.0088300

column 18 to 21

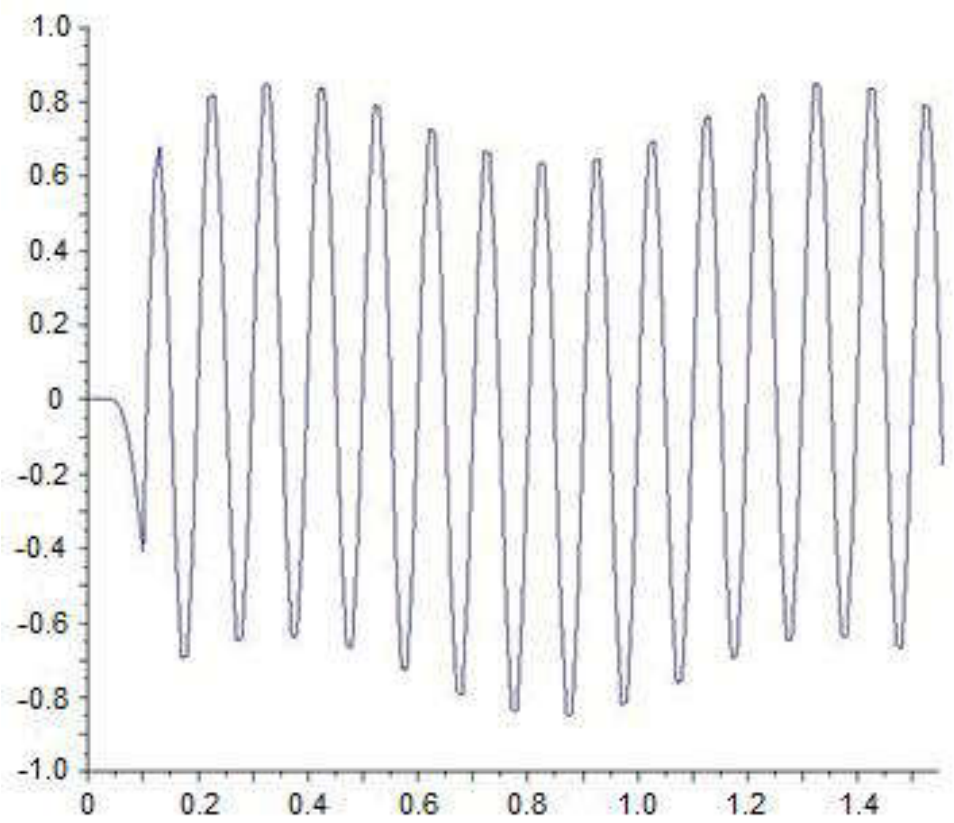
0.0005885 0.0013987 0.0006309 0.000000

Для фильтрации сигнала после получения коэффициентов фильтра необходимо построить передаточную функцию.

```
hpoly = poly (valcoeff, 'z','coeff');
hz = horner (hpoly, (1/%z));
lisys = syslin ('d', hz);
y = flts(x, lisys);                    //применение фильтра к сигналу
Y = fft(y)./(length(x)/2);             //вычисление спектра сигнала
plot(f(1:n/2),abs(Y(1:n/2)));          //вывод графика спектра
```



```
plot(t,y);     //представление выходного сигнала во временной области
```

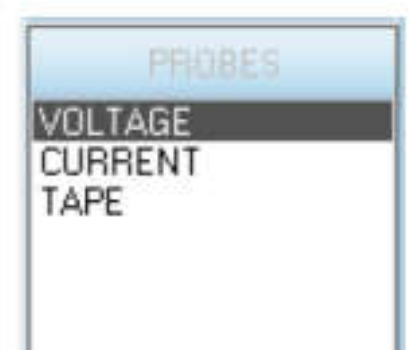


4. Реализация нерекурсивных КИХ-фильтров на базе микроконтроллеров

Краткая теория

В данной работе используется редактор ISIS программного пакета Proteus версии 8.1, поэтому рассмотрим кратко особенности работы с электрическими схемами в этом редакторе

Для выполнения проверки постоянного (переменного) напряжения или тока на участке цепи в программе Proteus можно использовать измерительные пробники тока (CURRENT) и напряжения (VOLTAGE).



Пробники размещаются в тех точках схемы, за которыми предполагается наблюдать. Выбрать нужный пробник можно на панели PROBES (рис. 4.1), которая может быть открыта посредством нажатия на кнопку Probe Mode на левой панели инструментов редактора ISIS.

Рис. 4.1. Панель PROBES редактора ISIS

Генераторы используются для подачи тестовых сигналов в исследуемую схему. Выбрать нужный генератор можно на панели GENERATORS (рис. 4.2), которая открывается нажатием на кнопку Generator Mode на левой панели инструментов редактора ISIS. Это же действие можно выполнить при помощи команды контекстного меню Place/Generator/.

Рис. 4.2. Панель GENERATORS редактора ISIS

Измерительные пробники тока и напряжения размещают в проекте Proteus до запуска симуляции схемы. Пробники напряжения используют в аналоговой и цифровой симуляции, а пробники тока – только для аналоговой симуляции. Пробники не имеют собственной лицевой панели как у виртуального прибора. Настройка их параметров выполняется в окне свойств до запуска симуляции схемы.

Результаты измерения (напряжение, ток) отображаются после запуска симуляции рядом с пиктограммой пробника (рис. 4.3). Для пробника тока важна его ориентация, которая указывает направление протекания тока.

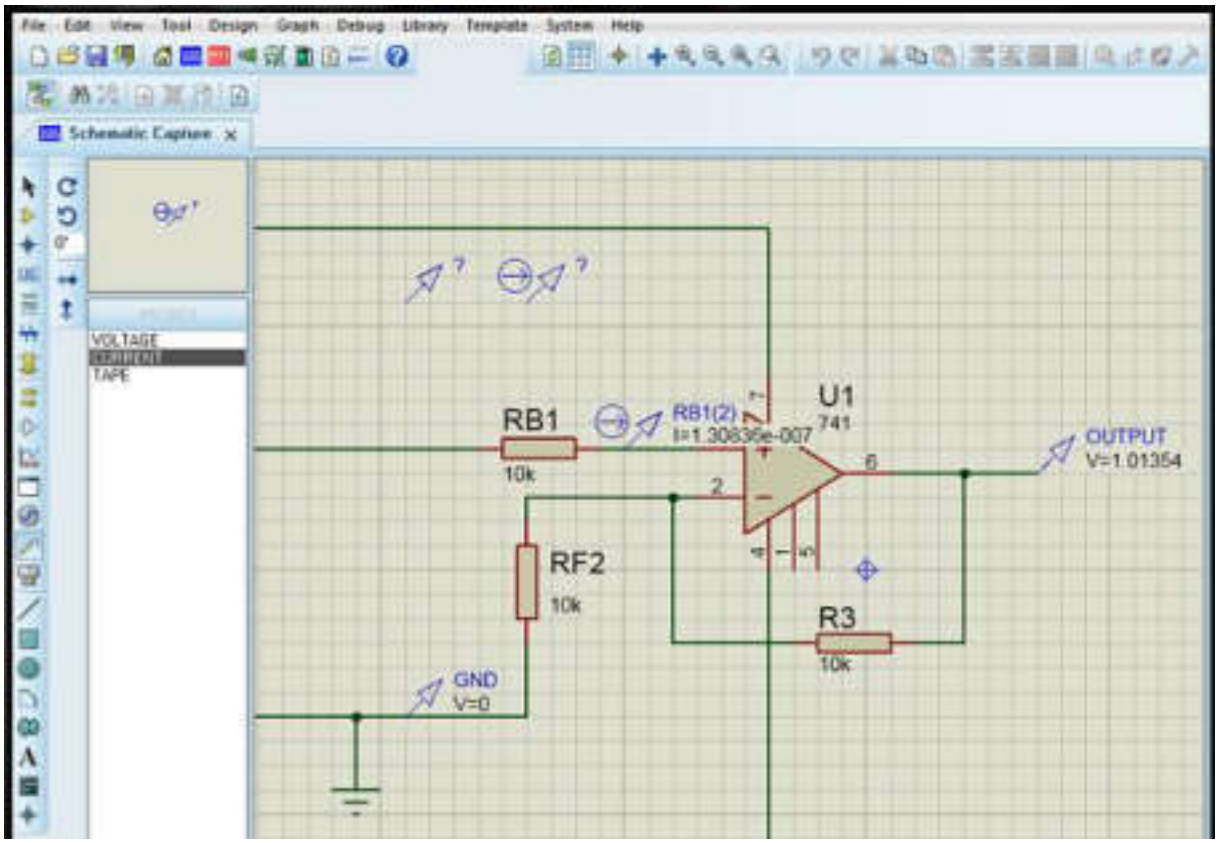


Рис. 4.3. Пример подключения пробников тока и напряжения к схеме

Неподключенные пробники имеют по умолчанию название «?». Когда пробник присоединен к цепи, ему автоматически присваивается имя цепи (компонента). Имя пробника разработчик проекта может изменить по своему усмотрению.

Для редактирования параметров пробника необходимо выделить его при помощи левой кнопки мыши, а затем при помощи правой кнопки мыши вызвать контекстное меню и выбрать пункт Edit Properties. В результате этих действий будет открыто окно Edit Current Probe для токового пробника (рис. 4.4а) и окно Edit Voltage Probe для пробника напряжения (рис. 4.4б).

Рис. 4.4. Окна настройки параметров:
(а) токового пробника

(б) пробника напряжения

Для токового пробника можно установить его имя (Name) и название файла (Filename) для записи данных. Для пробника напряжения кроме этого задают сопротивление нагрузки (Load (Ohms)). Запись файла воспроизводят устройством записи (TAPE).

Генераторы используются для подачи тестовых сигналов в схему. В программе ISIS доступны следующие типы генераторов:

- DC источник постоянного напряжения;
- Sine генератор синусоидального напряжения одной частоты с возможностью настройки параметров амплитуды, частоты и фазы сигнала;

- Pulse аналоговый импульсный генератор с возможностью настройки размаха импульса, периода, а также времени переднего и заднего фронтов;
- Exp экспоненциальный импульсный генератор;
- SFFM частотно-модулированный генератор (сигнал частотной модуляции одного синусоидального сигнала другим);
- Pwlin генератор кусочно-линейных сигналов, используется для создания импульсов и сигналов произвольной формы;
- File производит сигнал, который задается серией временных точек и значений данных, содержащихся в ASCII файле (данные представляются в виде пар «время» «напряжение»);
- Audio используется для проведения испытания схемы с помощью заранее подготовленного .wav (звукового) файла;
- Steady State установившийся логический уровень;
- Single Edge единичный переход от низкого к высокому или от высокого к низкому уровням;
- Single Pulse источник единичного тактового импульса;
- Clock источник цифрового тактового сигнала (последовательности импульсов, разделенных паузами);
- Pattern источник последовательности логических уровней;
- Easy HDL генератор, управляемый при помощи скрипта.

Добавляют генератор в рабочее поле проекта нажатием на строку с его названием на панели GENERATORS и размещением его с помощью мыши в необходимом месте на схеме. Генераторы размещают в рабочем проекте Proteus до запуска процесса симуляции схемы. Они не имеют своей лицевой панели, поэтому настройка их параметров выполняется в окне свойств до запуска симуляции схемы.

В каждой схеме может использоваться несколько генераторов, в том числе и копии одного и того же прибора. Каждая копия прибора настраивается и соединяется отдельно. Ниже (рис. 4.5) показаны ге-

нераторы Sine, Pulse и Clock, а также полученные с их выходов сигналы на экране осциллографа.

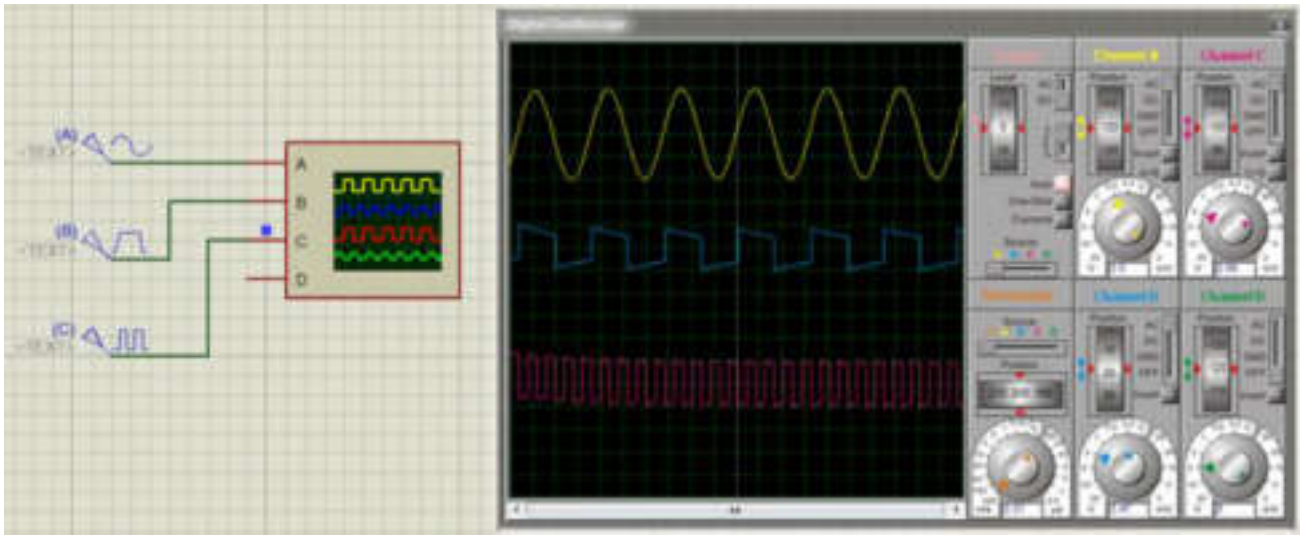
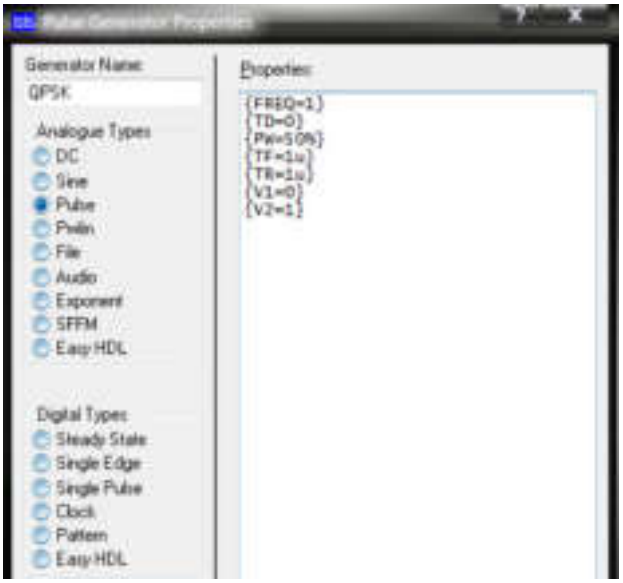


Рис. 4.5. Пример использования генераторов Sine, Pulse и Clock

Неподключенные генераторы имеют по умолчанию название «?». Когда прибор присоединен к цепи, ему автоматически присваивается имя цепи (компонента), которое можно изменить. Для настройки параметров генератора используют пункт Edit Properties. В открывшемся окне Generator Properties имеются следующие поля:

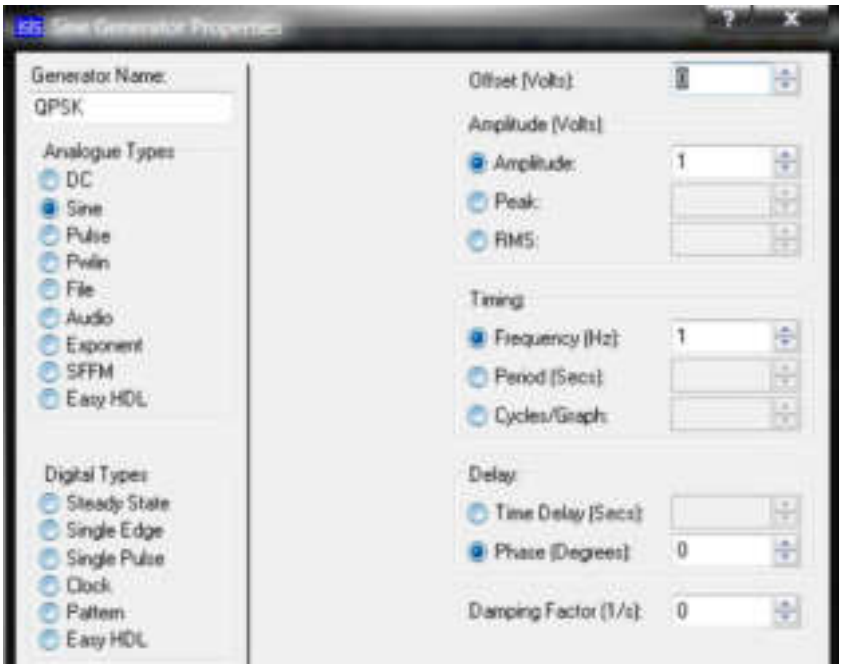
Generator Name; Analogue Types; Digital Types;
Current Source; Isolate Before; Manual Edits.



В последнем случае параметры генератора отображаются в виде текстового списка (рис. 4. 6), который можно редактировать.

Рис. 4.6. Отображение параметров генератора в виде текстового списка

Интерфейс правой части окна Generator Properties меняется по типу генератора и может содержать как поля выбора параметров (рис. 4.7а), так и поле ввода скрипта (рис. 4.7б) для генератора Easy HDL.



а)

б)

Рис. 4.7. Интерфейс в виде: (а) набора полей выбора, (б) поля ввода скрипта

Ход выполнения работы

1. В программном пакете PROTEUS собрать схему, представленную на рисунке 4.8.
2. По варианту задания установить параметры синусоидальных генераторов (амплитуды Un и Uv выбрать так, чтобы выходное напряжение U2 не превышало 2,5 В).

Таблица вариантов

Вариант	1	2	3	4	5	6	7	8	9	10
fn, Гц	8	7	6	9	11	12	5	15	10	4
fv, Гц	43	37	32	47	56	63	27	87	54	22
Un/Uv	5	6	5	6	5	6	5	6	5	6

3. Реализовать программу цифрового фильтр скользящего среднего, а также программы цифровых КИХ-фильтров с прямоугольным окном, окном Хэмминга, окном Блэкмана и окном Кайзера.
4. Вывести графики выходных напряжений для всех типов фильтров в режимах аналогового и Фурье анализа и сравнить между собой.

На рисунке 4.9 представлен пример фрагмента кода обработки данных аналогово-цифровым преобразователем. Этот код можно использовать в качестве основы для всех типов фильтров, добавив свой код в указанное место.

Содержание отчета:

- принципиальная схема реализации цифрового фильтра;
- графики аналогового анализа входного и выходного сигналов;
- графики анализа Фурье выходных сигналов;
- выводы по работе.

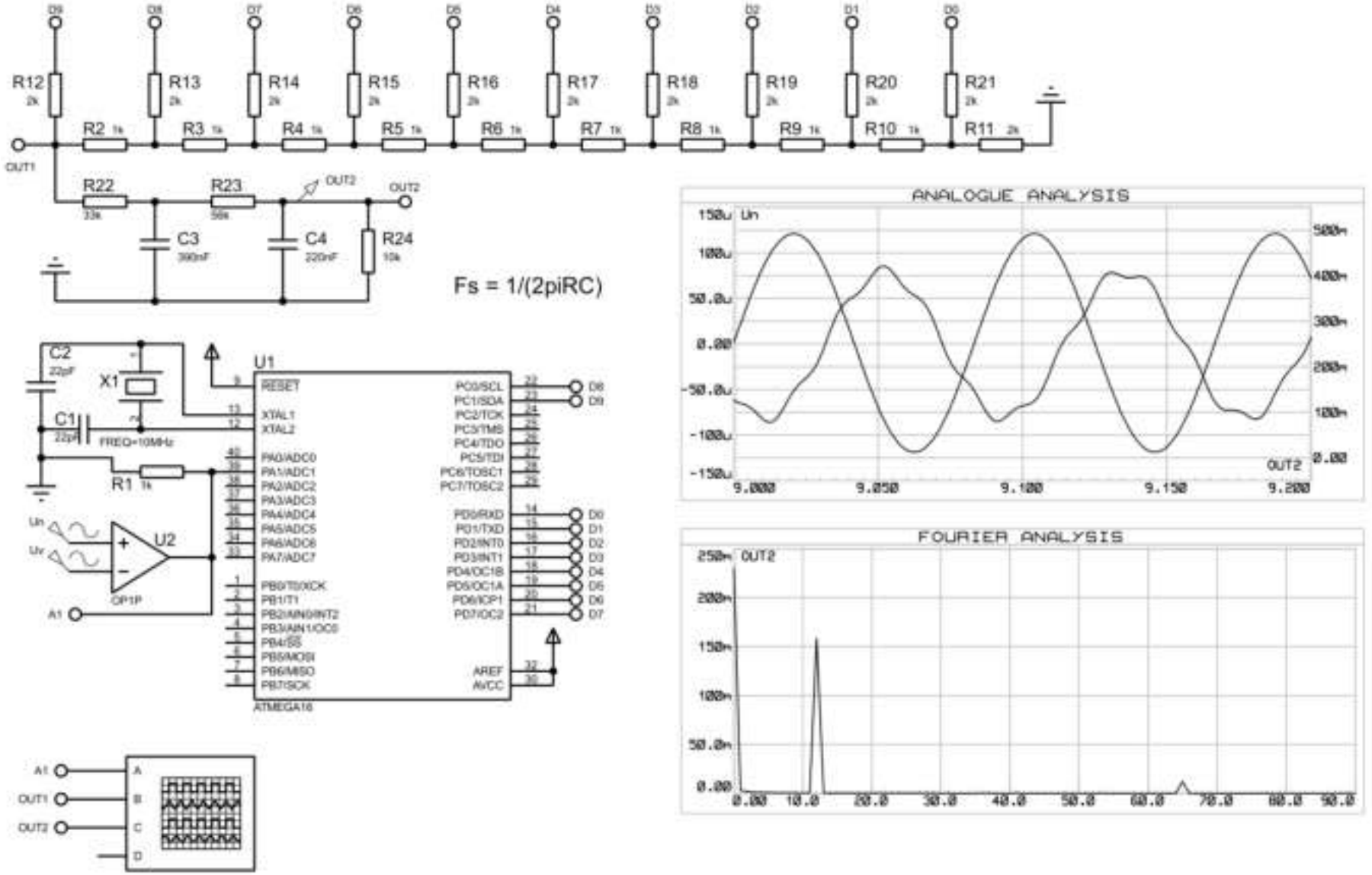


Рис. 4.8. Схема исследования программных цифровых фильтров

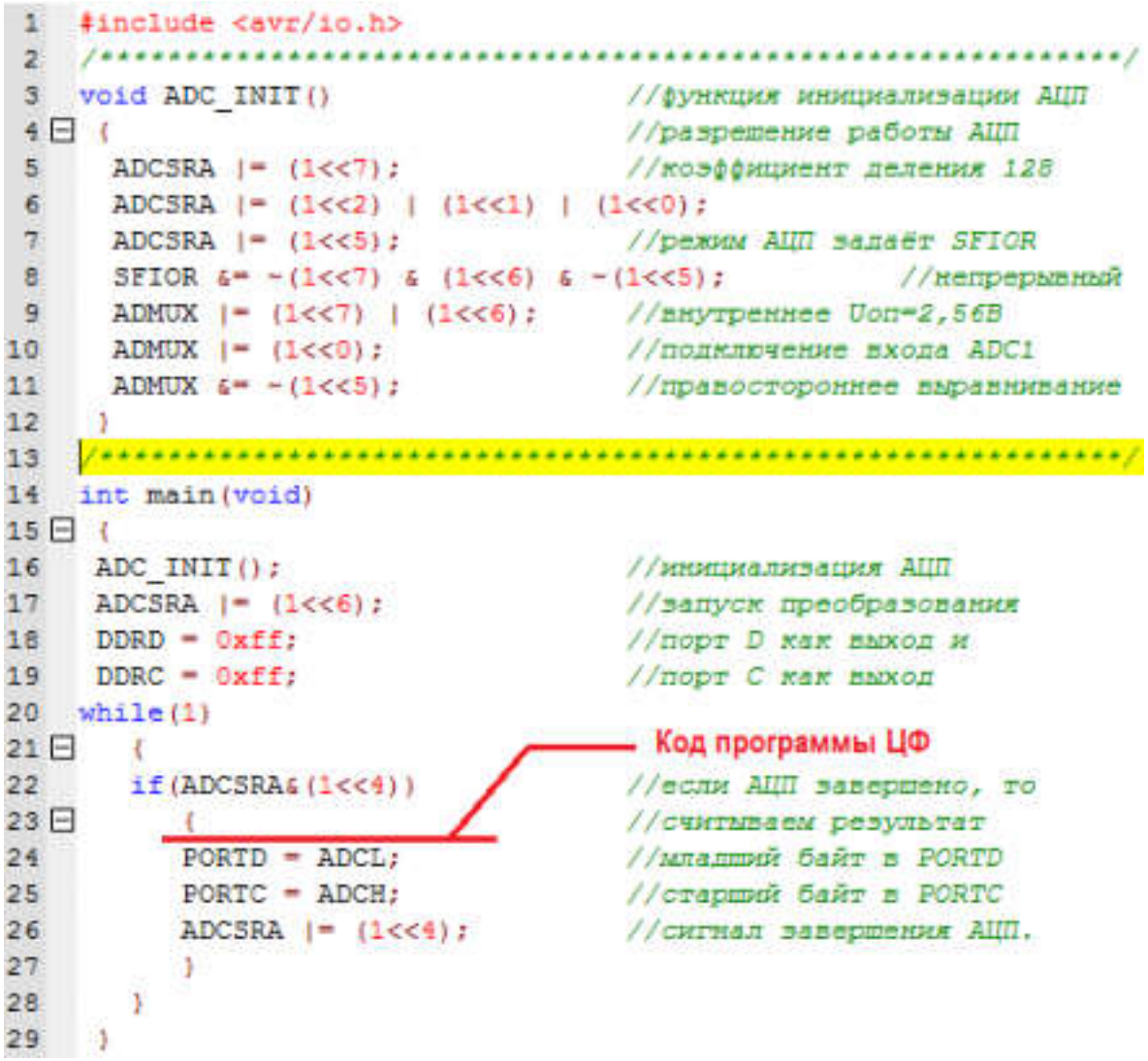


Рис. 4.9. Пример фрагмента кода обработки данных АЦП

Контрольные вопросы

1. Для чего предназначены измерительные пробники?
2. Каково назначение генераторов сигналов?
3. Какие параметры имеют пробники тока и напряжения?
4. Какие типы генераторов доступны в программе ISIS?
5. Какие поля используют для настройки параметров генератора?
6. Для чего предназначено поле генератора Manual Edits?

5. Синтез комбинационной логической схемы по таблице истинности на языке VHDL

Краткая теория

При создании проекта на языке программирования аппаратуры VHDL наиболее часто используются следующие основные модули:

Entity(объект) – объект проекта для задания интерфейса;

Architecture (архитектура) – архитектурное тело проекта для задания алгоритма работы на поведенческом или структурном уровне;

Configuration (конфигурация) – определяет правила соответствия *architecture&entity*;

Package (пакет) – определение часто используемых типов данных, компонент и функций;

Library (библиотека) – библиотеки, содержащие наборы типов данных и операторов;

Driver (драйвер) – источник сигнала (если сигнал генерируют 2 активных источника, то считается, что сигнал имеет два драйвера);

Bus (шина) – группа сигналов (в языке VHDL шина может иметь несколько драйверов);

Attribute (атрибут) – информация, включенная непосредственно в VHDL-объекты (емкость буфера, максимальная температура и т. п.);

Generic – параметр, передающий информацию в объект **entity** (если объект моделирует элемент задержки, то значения величин задержек можно передавать внутрь объекта через параметр generic);

Process (процесс) – основной модуль выполнения программы в языке программирования аппаратуры VHDL.

Структуру проекта на языке программирования аппаратуры VHDL можно представить в виде иерархии (рис. 5.1).

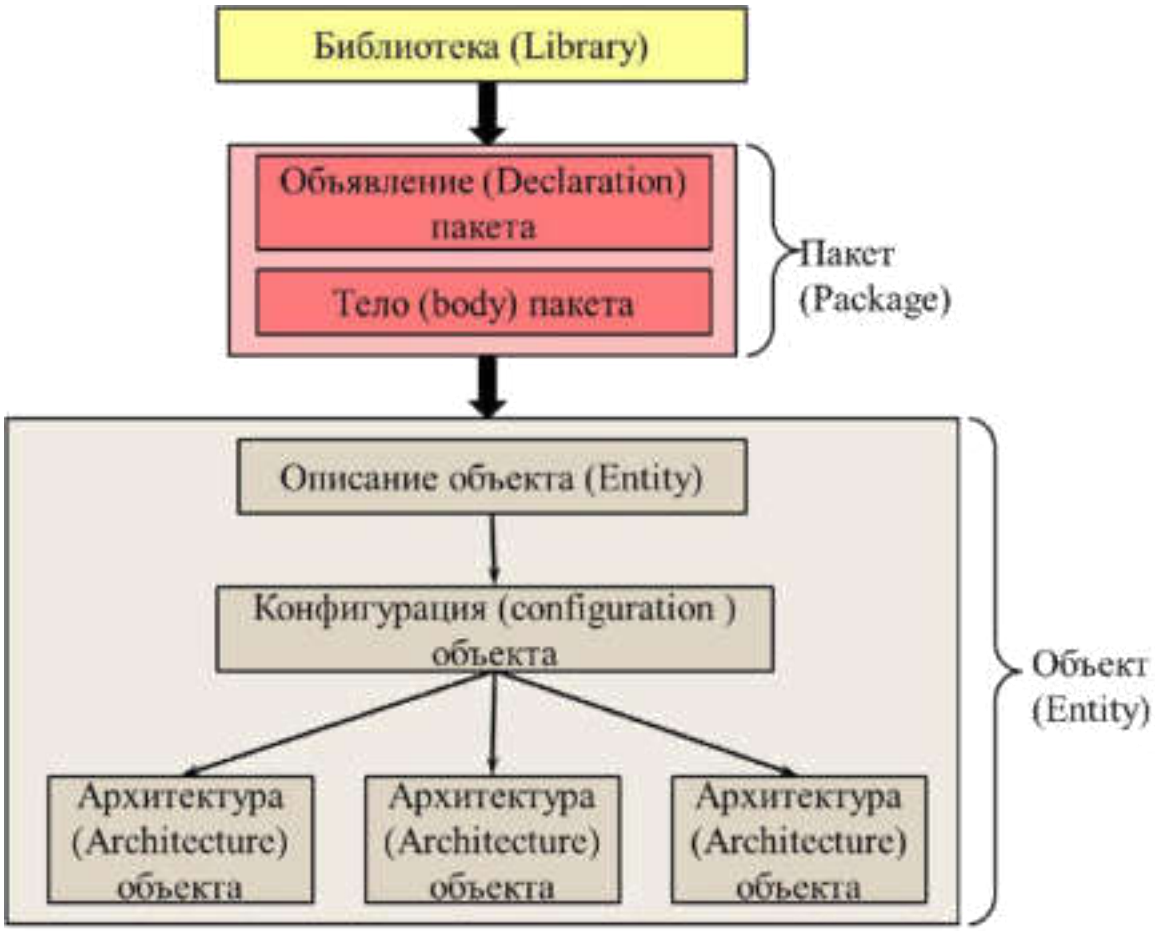


Рис. 5.1. Иерархия проекта на языке VHDL

Наиболее часто используются следующие основные модули:

Библиотеки (Library)

Библиотека – средство хранения модулей проекта, содержащее множество пакетов.

Подключение библиотеки:

LIBRARY library_name

В проекте используют следующие библиотеки:

- рабочая библиотека (**work**);
- библиотека ресурсов:
  - встроенная;
  - стандартная;
  - пользовательская.

Рабочая библиотека (work) – библиотека для компиляции рабочих пакетов и модулей;

Встроенная библиотека STD – библиотека минимального набора типов данных и функций.

Эта библиотека содержит следующие пакеты:
Standard – задает стандартные типы и операторы:

- Bit,
- Bit_vector
- Boolean,
- Integer,
- Real,
- Time,
- Character,
- String.

Textio (операторы ввода-вывода текста).

Встроенная и рабочая библиотеки, всегда активны. Предполагается, что модули проекта неявно содержат следующее объявление:

LIBRARY STD;

LIBRARY WORK;

USE STD.ALL;

Стандартная библиотека IEEE.

Для обеспечения моделирования сигналов от нескольких источников была разработана библиотека, в которой определены типы данных и операции для них. Состав пакетов библиотеки:

IEEE.std_logic_1164 – определяет особые типы **STD_ULOGIC** и **STD_LOGIC**, а также содержит набор логических функций и функций преобразования типов;

IEEE.std_logic_textio – определяет файловые операторы ввода-вывода для типов данных **STD_ULOGIC** и **STD_LOGIC**;

IEEE.std_logic_arith – определяет знаковый и беззнаковый типы, арифметические и логические операторы и функцию преобразования;

IEEE.numeric_bit – определяет для типа **BIT** знаковый и беззнаковый типы; арифметические, логические операторы; операторы сдвига и функции преобразования типов данных;

IEEE.numeric_std – определяет для типа **STD_LOGIC** знаковый и беззнаковый типы; арифметические, логические и операторы сдвига, а также функции преобразования типов данных;

IEEE.std_logic_signed – для типа **STD_LOGIC_VECTOR** определяет знаковые арифметические и логические операторы, а также функцию преобразования типов;

IEEE.std_logic_unsigned – этот пакет библиотеки определяет для типа **STD_LOGIC_VECTOR** беззнаковые арифметические и логические операторы и функцию преобразования типов;

IEEE.math_real – содержит операторы для работы над числами с плавающей запятой;

IEEE.math_complex – содержит операторы комплексных чисел;

Пакет (package).

Для многократного использования собственных модулей в различных проектах их декларации и определения помещают в отдельный *пакет* (сгруппированная совокупность операторов).

В новом проекте в его состав включают соответствующий сформированный пакет. В структуре описания пакета выделяют 3 части:

- блок ссылок на нужные библиотеки и пакеты;
- блок декларации элементов пакета;
- тело пакета.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;

PACKAGE package_name IS
                    -- раздел объявления
END package_name;

PACKAGE BODY package_name IS
                    -- описания функций и процедур
END package_name;
```

Объявление данных внутри пакетов, дает возможность использовать их любых объектах, которые подключили данный пакет. В состав пакета входят *две части*: декларативная часть пакета и тело пакета.

Декларативная часть пакета определяет его интерфейс почти так же, как определяется интерфейс внутри объекта **entity**.

Тело пакета определяет поведение пакета так же, как архитектура определяет поведение объекта **entity**.

Декларативная часть пакета может содержать:

- ✓ Объявления подпрограмм;
- ✓ Объявления типов и подтипов;
- ✓ Объявления констант и отложенных констант;
- ✓ Объявления глобальных сигналов;
- ✓ Объявления файлов;
- ✓ Объявления компонент;
- ✓ Объявления атрибутов;
- ✓ Объявления спецификаций;
- ✓ Окончание спецификаций;
- ✓ Секцию USE.

Все элементы, объявленные в декларативной части пакета, видны в любом модуле, который имеет подключение данного пакета при помощи команды **USE**. Объявленные подпрограммы и отложенные константы должны иметь соответственно тело подпрограммы или присвоение нужного значения для константы в теле пакета.

Тело пакета предназначено для присвоения значений отложенным константам и описания тел подпрограмм, объявленных в части деклараций. Тело пакета содержит следующие описания:

- ✓ Объявления подпрограмм;
- ✓ Тела подпрограмм;
- ✓ Объявления типов и подтипов;
- ✓ Объявления констант;
- ✓ Присвоение значений константам;
- ✓ Объявления файлов;
- ✓ Секцию USE.

Все объекты тела пакета (кроме присвоения значений отложенным константам и описаний тел подпрограмм) видны только внутри его тела.

Объект (entity)

Цифровые устройства, реализуемые на VHDL, описываются объектами **entity**. Если устройство состоит из одного уровня, то оно описывается одним объектом **entity**. При иерархической структуре объект верхнего уровня включает в себя один или несколько объектов нижнего уровня.

Описание объекта **entity** должно включать в себя имя объекта, входные и выходные порты объекта, а также информацию о его связи с другими объектами.

```
ENTITY mux IS
  PORT (
        a, b, c, d: IN BIT;
        s0, s1: IN BIT;
        x: OUT BIT
        );
END mux;
```

Служебное слово **ENTITY** обозначает старт описания объекта, имя которого – mux. В разделе **PORT** описано семь портов ввода/вывода типа **BIT**. Порты могут быть четырех типов:

IN – входной порт;

OUT – выходной порт;

INOUT – двусторонний порт;

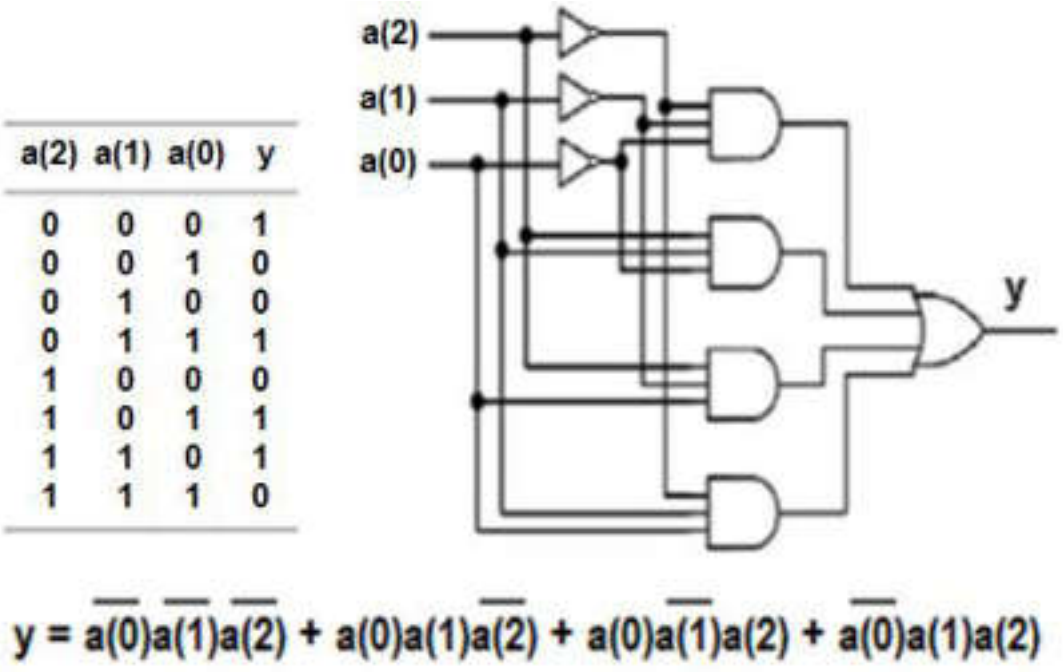
BUFFER – выходной порт с внутренней петлей обратной связи.

Архитектура (architecture)

Объект **entity** описывает только интерфейс VHDL модуля. Архитектура описывает схему проектируемого модуля устройства, на которой строится объект или алгоритм ее работы.

Архитектура всегда связана с объектом и в общем случае описывает либо поведение этого объекта (*описание на поведенческом уровне*), либо его структуру (*описание на структурном уровне*).

На рис. 5.2 представлена схема определения четности. Эта схема построена на основе булевского выражения дизъюнктивно нормальной формы (ДНФ), которое получено на основе таблицы истинности.



a(2)	a(1)	a(0)	y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Рис. 5.2. Схема определения четности

Описание на поведенческом уровне (алгоритм)

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
ENTITY even_det IS
  PORT(
     a: IN std_logic_vector(2 DOWNTO 0);
     y: OUT std_logic);
END even_det;

ARCHITECTURE soe_arch OF even_det IS
  SIGNAL p1, p2, p3, p4 : std_logic;
BEGIN
  y <= (p1 OR p2) OR (p3 OR p4)
  p1 <= (NOT a(2)) AND (NOT a(1)) AND (NOT a(0))
  p2 <= (NOT a(2)) AND a(1) AND a(0)
  p3 <= a(2) AND (NOT a(1)) AND a(0)
  p4 <= a(2) AND a(1) AND (NOT a(0))
END soe_arch;
```

Вначале описываются библиотеки, затем в разделе **ENTITY** описывается внешний интерфейс схемы (порты ввода-вывода).

Далее следует раздел **ARCHITECTURE**, который описывает, как эта схема функционирует.

Между словами **ARCHITECTURE** и **BEGIN** описываются локальные сигналы этой архитектуры с указанием их типа (в примере **std_logic**). Область описания архитектуры начинается со слова **BEGIN**. Все операторы, находящиеся между словами **BEGIN** и **END**, исполняются параллельно.

Задание на лабораторную работу:

Согласно варианту, указанному преподавателем, реализовать схему на языке программирования VHDL по таблице истинности.

Входы			**Вариант реализации выходного сигнала y**											
a(2)	**a(1)**	**a(0)**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
0	0	0	1	0	0	1	0	1	0	1	0	0	1	0
0	0	1	1	1	1	0	1	1	1	0	1	0	0	1
0	1	0	0	1	0	1	1	0	0	1	0	1	0	1
0	1	1	0	0	1	0	1	0	1	1	1	0	1	0
1	0	0	1	1	1	0	0	1	0	0	1	1	1	0
1	0	1	0	1	0	1	0	1	1	0	1	1	1	0
1	1	0	0	0	1	0	1	0	0	1	0	0	0	1
1	1	1	1	0	0	1	0	0	1	0	0	1	0	1

Содержание отчета:

- таблица истинности заданного варианта;
- уравнение дизъюнктивно-нормальной формы;
- программный код на языке VHDL;
- комбинационная схема варианта реализации;
- диаграммы функционального моделирования схемы;
- выводы по работе.

Контрольные вопросы

1. Какие модули используют при создании проекта на VHDL?
2. Какова структура (иерархия) проекта на языке VHDL?
3. Какие типы библиотек существуют в языке VHDL?
4. Какова структура описания проекта в языке VHDL?
5. Какие описания содержит тело пакета?
6. Каково назначение объекта **entity**?
7. Что собой представляет архитектура проекта?

6. Проектирование основных вычислительных узлов цифровых фильтров (последовательный сумматор)

Краткая теория

Последовательный сумматор (сумматор с последовательной обработкой разрядов) представляет собой синхронный автомат с памятью. Разряды слагаемых подаются на входы одноразрядного сумматора последовательно, начиная с младших. В текущем такте работы автомата выдается разряд суммы, а сигнал переноса запоминается D-триггером. В следующем такте работы автомата перенос передается в следующий разряд.

Логическое выражение для сумматора разрядности n имеет вид

$$S_i = x_i \oplus y_i \oplus c_i, \qquad c_{i+1} = x_i \& y_i \vee x_i \& c_i \vee y_i \& c_i, \tag{6.1}$$

где $i = 0 \ldots n\text{-}1$ – номер разряда слагаемых и суммы; c_0 – входной бит переноса сумматора; c_n – выходной бит переноса сумматора. Для последовательного сумматора бит c_0 всегда равен нулю, а бит c_n всегда отбрасывается.

Рабочая ячейка последовательного сумматора показана на рис.6.1 (КС – комбинационная схема)

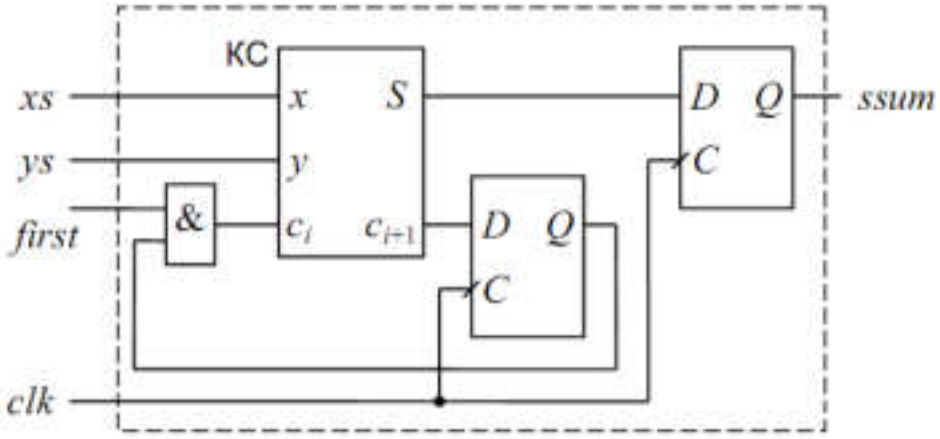


Рис.6.1. Ячейка последовательного сумматора

На входы сумматора последовательно поступают разряды операндов *xs* и *ys*, на контакте *ssum* последовательно формируется выход суммы, сигнал переноса фиксируется по переднему фронту тактовых импульсов *clk* триггером и используется как входной в следующем такте. Следует отметить, что выход суммы тоже должен запоминаться триггером, иначе переключение триггера переноса будет искажать результат.

Сигнал *first* с помощью схемы «И» блокирует цепь входного переноса в нулевом такте работы схемы (т. е. устанавливает $c_0 = 0$). Для формирования *n*-разрядной суммы необходимо *n* тактов работы схемы. Однако для защиты от переполнения операция суммирования должна выполняться (*n*+1) тактов. Знаковый разряд операндов обрабатывается дважды.

Отличительной особенностью ячейки последовательного суммирования является предельно высокая частота работы и малое использование объема ПЛИС. Для преобразования параллельного кода в последовательный код и наоборот используются сдвиговые регистры.

На рис.6.2 показана временная диаграмма работы последовательного сумматора. Цифры внутри временных интервалов соответствуют номерам разрядов операндов.

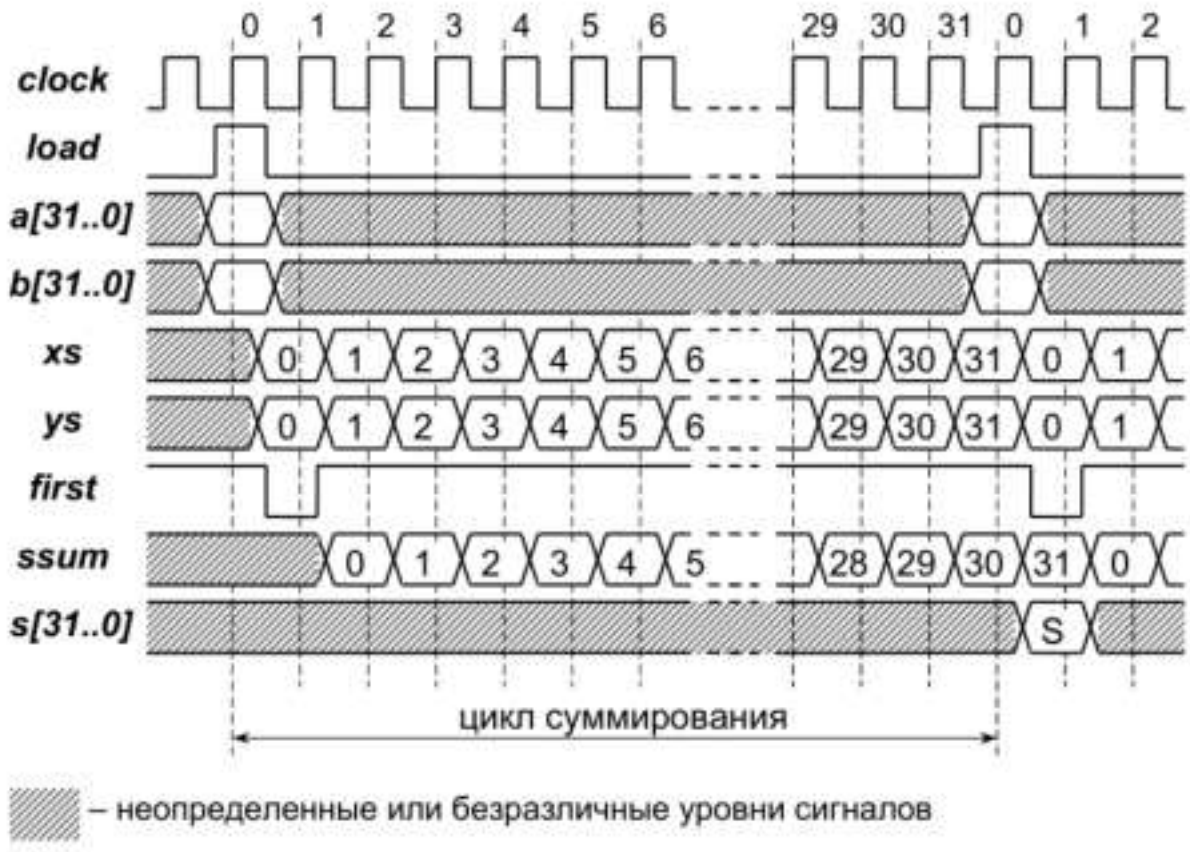


Рис.6.2. Временная диаграмма работы последовательного сумматора

Ход выполнения работы

1. Создать индивидуальный рабочий каталог для лабораторных работ. Путь каталога на диске – по указанию преподавателя.
2. Запустить САПР Quartus II, создать новый проект, выполнив команду **File –> New Project Wizard**...:

- в первом окне (*Introduction*) нажать клавишу *Next*
- во втором окне (*page 1 of 5*) установить путь к созданному каталогу, после чего ввести имя проекта Sersum_V××, где ×× – номер варианта задания, и нажать *Next*
- в третьем окне (*page 2 of 5*) нажать клавишу *Next*
- в четвертом окне (*page 3 of 5*) выбрать семейство Cyclone IV E, в списке *Available devices* выбрать ПЛИС и нажать клавишу *Next*
- в пятом окне (*page 4 of 5*) без установок нажать клавишу *Next*
- в шестом окне (*page 5 of 5*) нажать клавишу *Finish.*

3. Создать исходный файл сумматора *File –> New...*

   В окне *New* на вкладке *Device Design Files* выбрать тип создаваемого файла *Block Diagram/Schematic File*, если будет вводиться схема в графическом редакторе (см. рис.6.3), или *VHDL File*, если будет ввод текста на VHDL. После выбора нажать Ok.

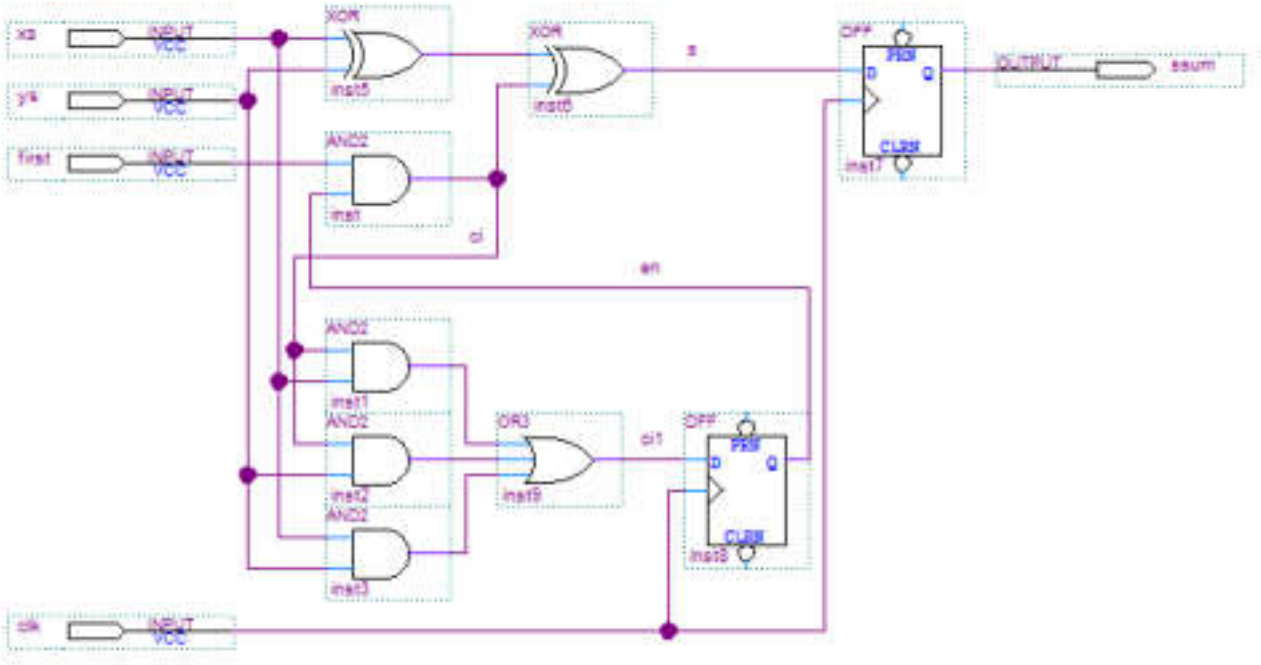


Рис.6.3. Схема последовательного сумматора, созданная в Quartus II

Пример программы сумматора на VHDL

```
library ieee;
use ieee.std_logic_1164.all;
entity SerSumm is
port (xs,ys,first,clk : in std_logic;
      ssum : out std_logic);
end SerSumm;
architecture ss_arch of SerSumm is
     signal ci : std_logic := '0';
begin
process(clk) begin
     if (rising_edge(clk)) then
          if (first = '1') then
          ssum <= xs xor ys xor ci;
          ci <= (xs and ys) or (xs and ci) or (ys and ci);
          end if;
```

```
    end if;
end process;
end ss_arch;
```

5. Ввести текст или схему проектируемого сумматора и сохранить файл командой *File –> Save As...* .
6. Установить созданный файл основным в проекте командой *Project –> Set as Top-Level Entity.*
7. Выполнить компиляцию проекта командой *Processing –> Start Compilation*. Устранить ошибки, если они возникли.
8. Создать файл временных диаграмм для проверки работы блока *File –> New...* В окне *New* выбрать *Vector Waveform File* и нажать Ok
9. В созданном файле щелкнуть правой кнопкой мыши на поле *Name*, после чего в выпавшем меню выбрать *Insert Node or Bus...* В появившемся окне нажать кнопку *Node Finder...* .
10. В окне *Node Finder* в списке *Filter* установить *Pins: all* , после чего нажать кнопку *List*. Слева в списке *Nodes Found* появится список всех контактов проекта. Требуемые сигналы переместить в правый список с помощью кнопок между списками. Нажать *Ok* дважды. Выбранные сигналы появятся на временной диаграмме.
11. Установить время моделирования 1 мкс командой *Edit –> End Time...* и настроить параметры редактора временных диаграмм *Tools –> Optoins...–> Waveform Editor:*
    – установить шаг сетки времени 10 нс *Default grid period 10 ns*
    – привязать сигналы к временной сетке флагом *Snap to grid*
    – сделать сетку видимой *View –> Show time grid.*
12. Выделить на временной диаграмме тактовый сигнал, нажать кнопку *Overwrite Clock* и в появившемся окне *Clock* установить *Period 20 ns.*
13. Отредактировать остальные сигналы временной диаграммы, учитывая следующие замечания:

- переключения входных сигналов должны происходить по *переднему фронту* тактового импульса;
- для задания произвольного значения сигналов на шине необходимо выделить требуемый временной интервал и нажать кнопку нужного сигнала в требуемом коде.

14. Сохранить файл созданной временной диаграммы.
15. Смоделировать работу схемы *Processing –> Start Simulation* и сравнить полученные результаты с ожидаемыми. В случае несовпадения отыскать и исправить ошибки.

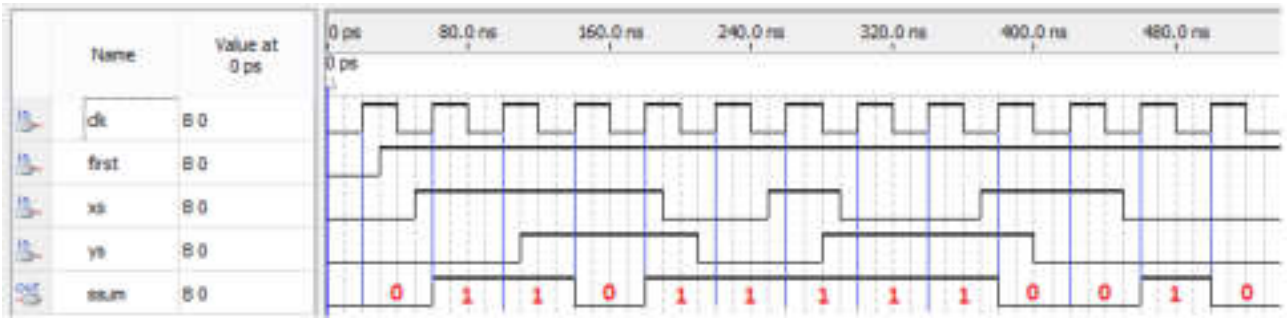


Рис.6.4. Временная диаграмма работы последовательного сумматора

Содержание отчета:

- листинг исходного файла описания блока (.bdf или .vhd);
- диаграмму результата временного моделирования (.vwf);
- символ функционального блока (.bsf)

Варианты индивидуальных заданий

Вариант	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1-е слагаемое	110	120	130	140	150	160	170	180	190
2-е слагаемое	290	280	270	260	250	240	230	220	210

Контрольные вопросы

1. Что собой представляет последовательный сумматор?
2. Каковы логические выражения для последовательного сумматора?
3. Какие блоки есть в рабочей ячейке последовательного сумматора?
4. Что собой представляет цикл суммирования?

7. Проектирование основных вычислительных узлов цифровых фильтров (табличный множитель)

Краткая теория

Табличный множитель является блоком табличного умножителя цифрового фильтра, осуществляющего умножение вектора-строки ***X*** отсчетов входного сигнала на вектор-столбец ***H*** коэффициентов фильтра, которые, по сути, представляют собой вектор констант.

Формулу для вычисления в векторном множителе с размерностью вектора, равной четырем, можно представить в виде:

$$y(n) = s(0)\cdot h(0) + s(1)\cdot h(1) + s(2)\cdot h(2) + s(3)\cdot h(3)$$

или

$$\begin{aligned} y(n) = {} & [-s_{k-1}(0)\cdot 2^{k-1} + s_{k-2}(0)\cdot 2^{k-2} + \ldots + \\ & s_0(0)\cdot 2^0]\cdot h(0) + \\ & + [-s_{k-1}(1)\cdot 2^{k-1} + s_{k-2}(1)\cdot 2^{k-2} + \ldots + \\ & s_0(1)\cdot 2^0]\cdot h(1) + \\ & + [-s_{k-1}(2)\cdot 2^{k-1} + s_{k-2}(2)\cdot 2^{k-2} + \ldots + \\ & s_0(2)\cdot 2^0]\cdot h(2) + \\ & + [-s_{k-1}(3)\cdot 2^{k-1} + s_{k-2}(3)\cdot 2^{k-2} + \ldots + s_0(3)\cdot 2^0]\cdot h(3), \end{aligned} \qquad (7.1)$$

где $s_i\,(n)$ – i-я двоичная цифра операнда n. В дополнительном коде вес знакового разряда – 2^{k-1}, что отражено в выражениях (7.1 и 7.2).

Если раскрыть скобки (7.1) и сгруппировать полученные произведения по степеням двойки, то получим следующее выражение:

$$\begin{aligned} y(n) = {} & -[s_{k-1}(0)\cdot h(0) + s_{k-1}(1)\cdot h(1) + s_{k-1}(2)\cdot h(2) + s_{k-1}(3)\cdot h(3)]\cdot 2^{k-1} + \\ & + \ldots + \\ & + [s_1(0)\cdot h(0) + s_1(1)\cdot h(1) + s_1(2)\cdot h(2) + s_1(3)\cdot h(3)]\cdot 2^1 + \\ & + [s_0(0)h(0) + s_0(1)h(1) + s_0(2)h(2) + s_0(3)h(3)]\cdot 2^0. \end{aligned} \qquad (7.2)$$

Каждую сумму произведений в квадратных скобках (7.2) будем вычислять по таблице *LUT*, подавая на ее адресные входы двоичные

цифры с одинаковым весом сразу 4-х операндов (цифры s_i (3…0)). Фактически в каждом такте работы *LUT* выполняется умножение однобитового вектора s_i (3…0) на многоразрядный вектор h(3…0).

Таблица *LUT* состоит из 16-ти ячеек и при разрядности коэффициентов d содержимое таблицы имеет d+2 двоичных разряда. Для реализации такой *LUT* требуется d+2 логические ячейки.

Таблица 7.1

Номер ячейки	s_i (3)	s_i (2)	s_i (1)	s_i (0)	Содержимое ячейки
0	0	0	0	0	0 + 0 + 0 + 0
1	0	0	0	1	0 + 0 + 0 + h(0)
2	0	0	1	0	0 + 0 + h(1) + 0
3	0	0	1	1	0 + 0 + h(1) + h(0)
4	0	1	0	0	0 + h(2) + 0 + 0
5	0	1	0	1	0 + h(2) + 0 + h(0)
6	0	1	1	0	0 + h(2)+ h(1) + 0
7	0	1	1	1	0 + h(2)+ h(1) + h(0)
8	1	0	0	0	h(3) + 0 + 0 + 0
9	1	0	0	1	h(3) + 0 + 0+ h(0)
10	1	0	1	0	h(3) + 0 + h(1) + 0
11	1	0	1	1	h(3) + 0 + h(1) + h(0)
12	1	1	0	0	h(3) + h(2) + 0 + 0
13	1	1	0	1	h(3) + h(2) + 0 + h(0)
14	1	1	1	0	h(3) + h(2) + h(1) + 0
15	1	1	1	1	h(3) + h(2) + h(1) + h(0)

В случае последовательных вычислений разряды с одинаковыми весами нескольких операндов (обычно четырех) поступают в схему обработки последовательно, начиная с младших. Это соответствует тому, что выражения в квадратных скобках (7.2) вычисляются поочередно, начиная с младшей степени двойки. Результат получают сум-

мированием выходных отсчетов таблицы с учетом номера такта (т. е. степени двойки).

Табличный множитель на *VHDL* реализуют в качестве функции с параметрами. В примере ниже описывается компонент с именем *lut16*, представляющий таблицу из четырех 6-разрядных чисел.

```
LIBRARY IEEE;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.all;

ENTITY  lut16    IS
    GENERIC (C0  : INTEGER:=0;
             C1  : INTEGER:=2;
             C2  : INTEGER:=-5;
             W   : NATURAL:=6);
    PORT (   ADDR : IN INTEGER RANGE 0 to 3;
             DATA : OUT STD_LOGIC_VECTOR(W-1 downto 0));

END lut16;

architecture behavior of lut16 is
signal s : integer RANGE -(2**W)+1 TO 2**W - 1;
begin
     process(ADDR)
     begin
          case ADDR is
               when 0 => s <= C0;
               when 1 => s <= C1;
               when 2 => s <= C2;
               when 3 => s <= C1 + C2;
          end case;
     end process;
     DATA <= CONV_STD_LOGIC_VECTOR(s,W);
end behavior;
```

Декларация *GENERIC* позволяет задать параметры проектируемого модуля табличного множителя: *C0…C2* – коэффициенты умножения (множители), *W* – разрядность выходной шины модуля. Коэф-

фициенты умножения описаны как целые числа со знаком (тип *INTEGER*), оператор := задает значения параметров по умолчанию.

Декларация *PORT* описывает внешние порты блока: входной порт *ADDR* – целое число в диапазоне от 0 до 3 (что соответствует 2-разрядной адресной шине), выходной порт *DATA* – стандартный логический вектор. Размерность выходного вектора равна значению параметра *W*.

Описание архитектуры содержит декларацию сигнала *s* – целый тип в диапазоне $-2^{W-1} + 1 \ldots 2^{W-1} - 1$ (для примера этот диапазон –31…31). Затем описывают процесс, использующий оператор ***case***.

Функция *CONV_STD_LOGIC_VECTOR*, которая содержится в библиотеки *ieee.std_logic_arith*, осуществляет преобразование сигнала *s* в логический вектор размерности *W*. В результате компиляции примера синтезируется *комбинационная* схема табличного множителя, представленная на рисунке 7.1.

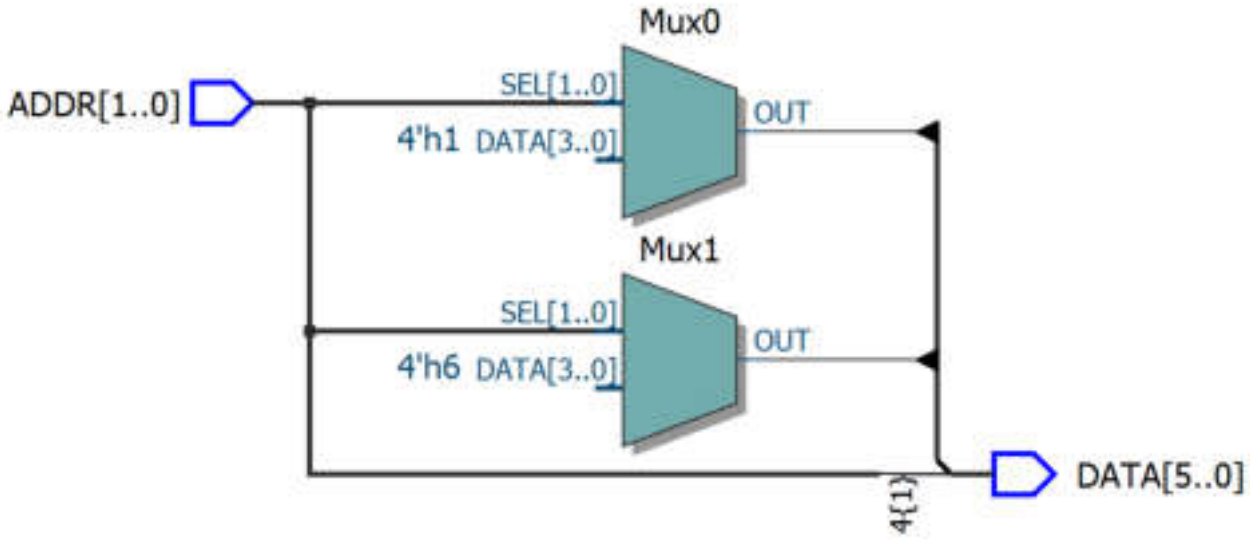


Рис. 7.1. Синтезированная схема табличного множителя

Принцип работы табличного множителя поясняет временная диаграмма входных и выходных сигналов, показанная на рисунке 7.2.

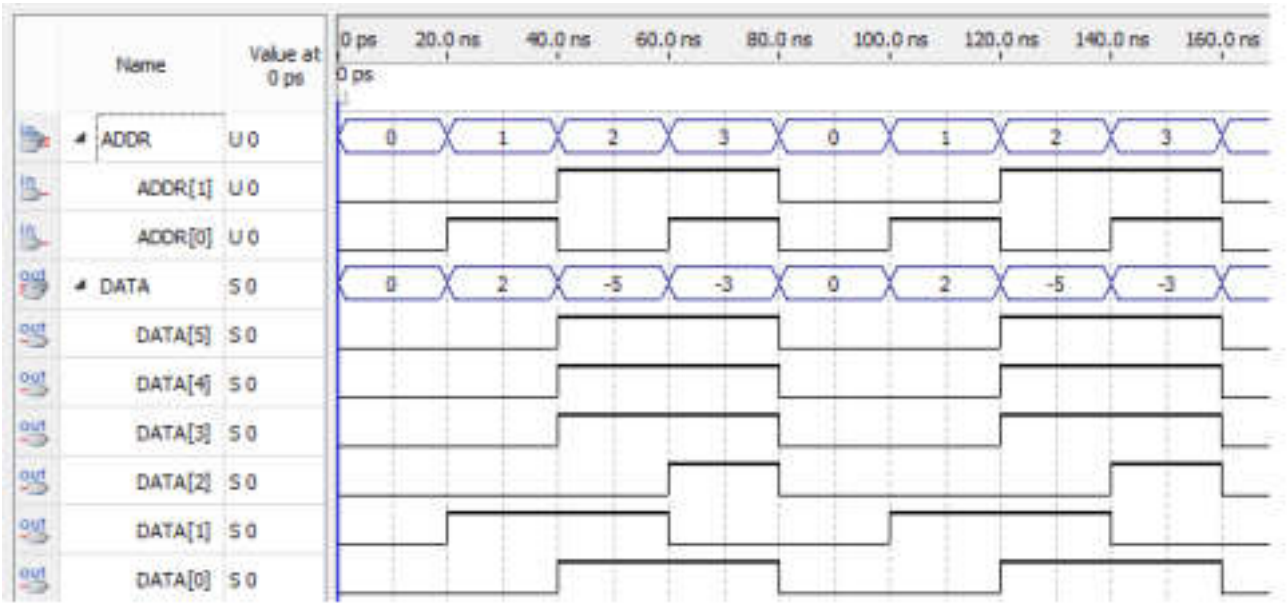


Рис. 7.2. Временная диаграмма работы табличного множителя

С целью повышения общего быстродействия схемы в некоторых случаях ее выход должен иметь регистр. Это значит, что результатом синтеза должен быть автомат с памятью. Для этого в декларации портов должен быть объявлен входной сигнал *clk* (тактовый сигнал).

```
PORT (   ADDR : IN INTEGER RANGE 0 to 3;
         clk  : IN STD_LOGIC;
         DATA : OUT STD_LOGIC_VECTOR(W-1 downto 0));
```

В описании архитектуры требуется добавить процесс со списком чувствительности *clk*, как показано ниже.

```
process(clk)
begin
     if (clk'EVENT AND clk = '1') then
          DATA <= CONV_STD_LOGIC_VECTOR(s,W);
     end if;
end process;
```

Событие, по которому происходит выдача сигналов на выход, это передний фронт импульса *clk* (условие *clk'EVENT AND clk = '1'*). После добавления этих двух фрагментов кода и компиляции схема изменяется и приобретает вид, показанный на рисунке 7.3.

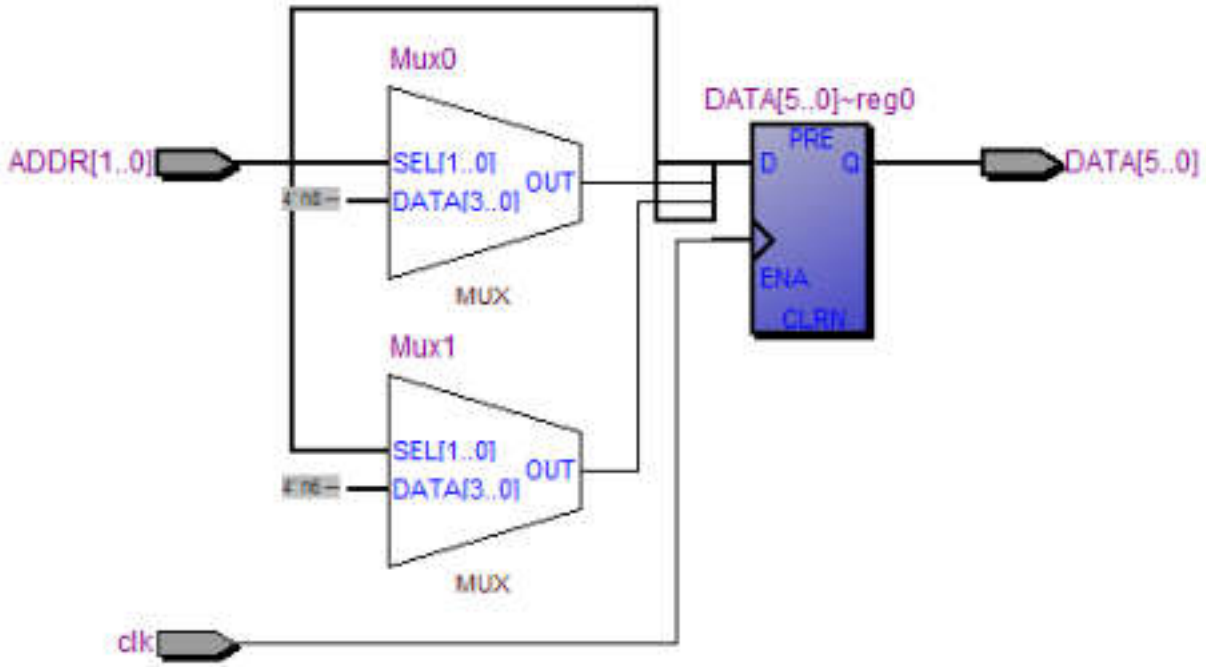


Рис. 7.3. Схема табличного множителя с регистром

Принцип работы табличного множителя с регистром поясняет временная диаграмма сигналов, показанная на рисунке 7.4.

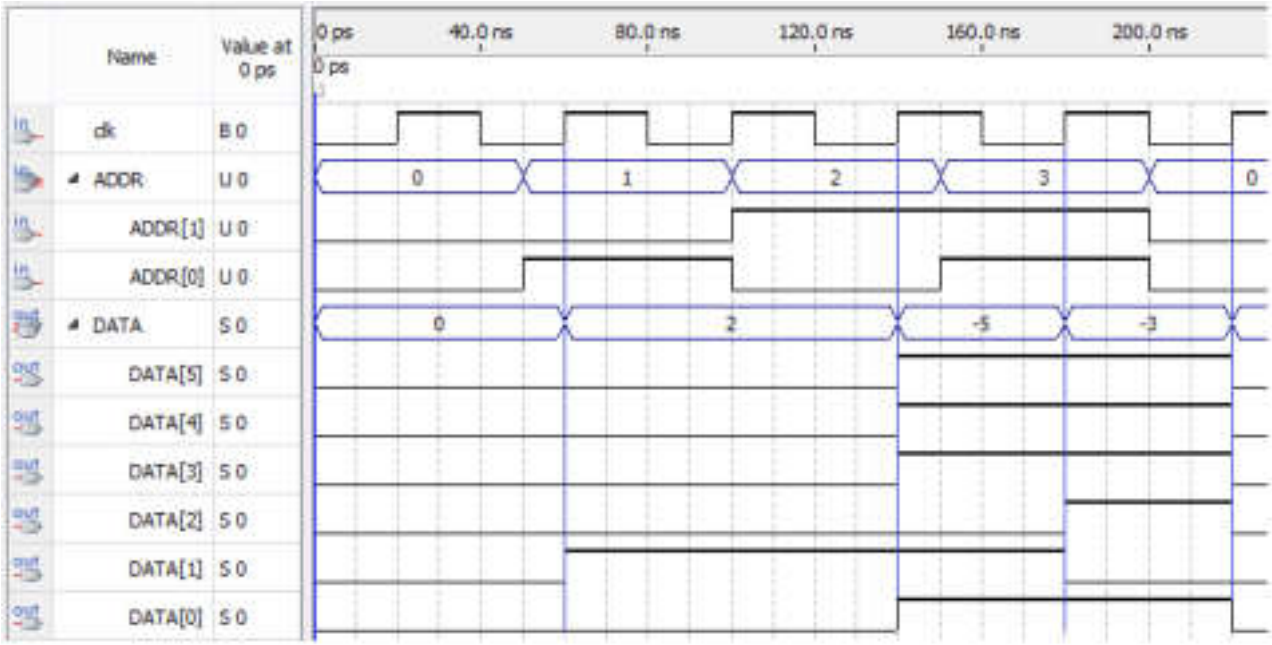


Рис. 7.4. Временные диаграммы работы табличного множителя с регистром

Содержание отчета:

- листинг исходного файла описания блока (.bdf или .vhd);
- диаграмма результата временного моделирования (.vwf);
- символ функционального блока (.bsf)

Варианты индивидуальных заданий

Вариант	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
C0	-1	-2	-3	-4	-5	-6	-7	-8	-9
C1	11	12	13	14	15	16	17	18	19
C2	29	28	27	26	25	24	23	22	21

Контрольные вопросы

1. Что собой представляет табличный множитель?
2. Каковы логические выражения для табличного множителя?
3. Какой состав блоков в простом табличном множителе?
4. Какой состав блоков в табличном множителе с памятью?
5. Что собой представляет временная диаграмма работы множителя?

8. Проектирование основных вычислительных узлов цифровых фильтров (масштабирующий аккумулятор)

Краткая теория

Если рассматривать масштабирующий аккумулятор с точки зрения выполняемых им операций, то он выполняет две операции:

1) умножение входного операнда (для фильтра – выражения в квадратных скобках формулы (7.2)) на текущую степень двойки;

2) суммирование.

Принцип работы масштабирующего аккумулятора заключается в том, что его содержимое на каждом такте сдвигается на один разряд вправо, что соответствует сдвигу очередного вычисленного выражения влево, т. е. умножению на два.

Структура устройства масштабирующего аккумулятора показана на рис.8.1, сдвиг вправо обеспечивается соответствующим соединением проводников шины.

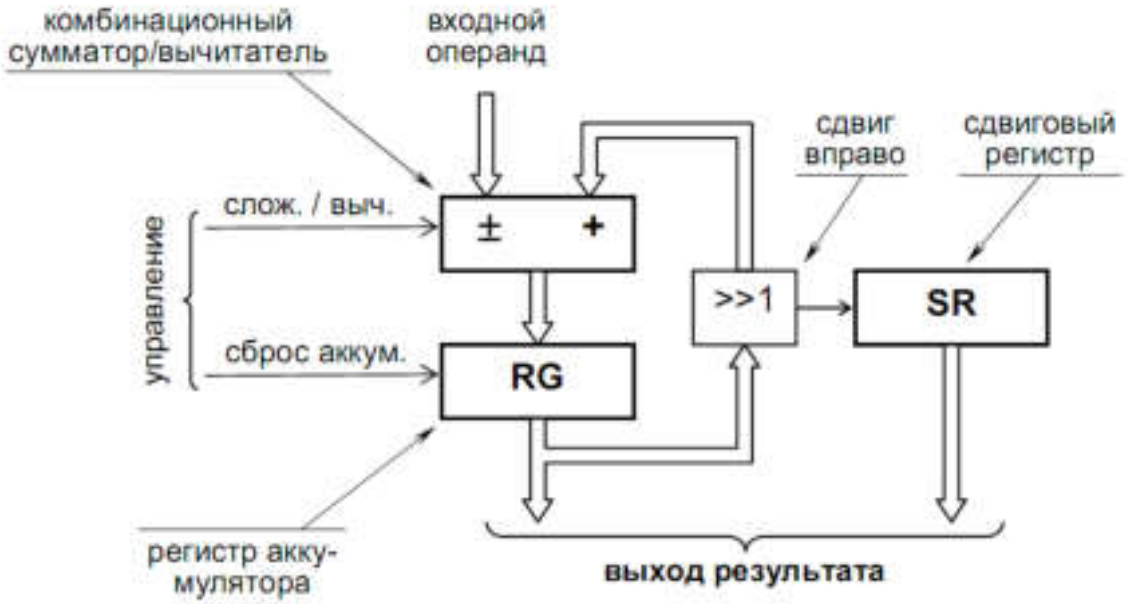


Рис. 8.1. Структурная схема масштабирующего аккумулятора

Пример реализации блока масштабирующего аккумулятора в программе Quartus показан на рис. 8.2. Блок масштабирующего аккумулятора реализован на двух мегафункциях:

LPM_ADD_SUB – сумматор/вычитатель и

LPM_SHIFTREG – регистр сдвига.

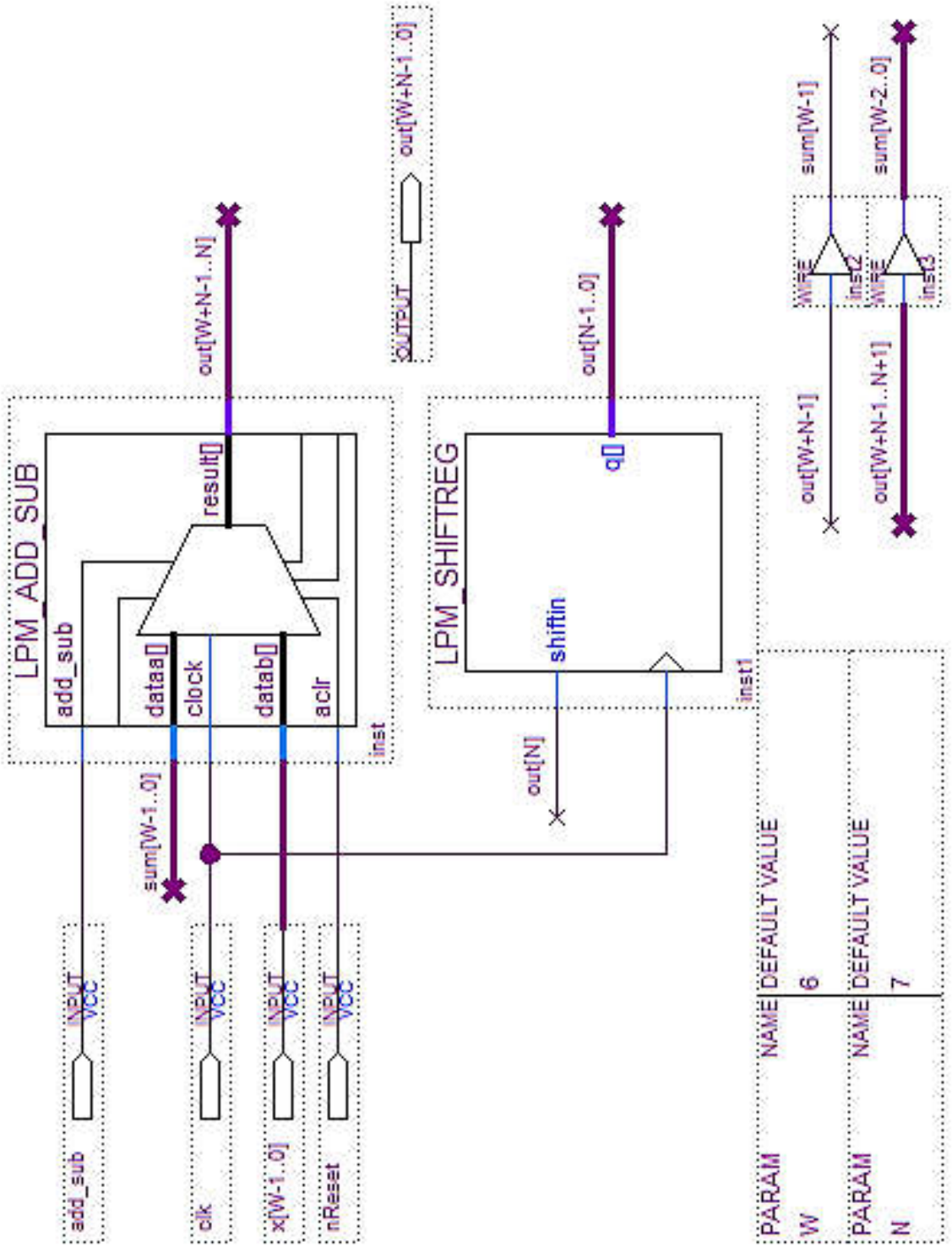


Рис. 8.2. Схема масштабирующего аккумулятора в Quartus II

Общие параметры аккумулятора задаются двумя библиотечными примитивами ***PARAM***: параметр ***W*** определяет разрядность сумматора, а ***N*** – разрядность сдвигового регистра. Эти параметры передаются в соответствующие мегафункции. Примитивы ***WIRE*** обеспечивают правый сдвиг операнда в обратной связи аккумулятора.

В мегафункции сумматора ***LPM_ADD_SUB*** порты ***aclr***, ***add_sub***, ***clock***, ***dataa[]***, ***datab[]***, ***result – used***, остальные – ***unused***. Для порта ***aclr*** дополнительный признак ***Inversion – All***. Параметры сумматора:

```
LPM_DIRECTION           "DEFAULT"
LPM_PIPELINE            1
LPM_REPRESENTATION      "SIGNED"
LPM_WIDTH               W
MAXIMIZE_SPEED
ONE_INPUT_IS_CONSTANT
```

В мегафункции регистра сдвига ***LPM_SHIFTREG*** порты ***clock***, ***q[]***, ***shiftin – used***, остальные – ***unused***. Параметры регистра сдвига:

```
LPM_AVALUE
LPM_DIRECTION       "RIGHT"
LPM_SVALUE
LPM_WIDTH           N
```

По схеме рис. 8.2 может быть получено структурное описание на *VHDL* с использованием мегафункций, которое приводится ниже.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

LIBRARY lpm;
USE lpm.lpm_components.all;

ENTITY acm IS
GENERIC (W   : NATURAL:=6;
         N   : NATURAL:=7);
PORT (add_sub : IN STD_LOGIC;
       clk   : IN STD_LOGIC;
       X     : IN STD_LOGIC_VECTOR(W-1 downto 0);
    nReset   : IN STD_LOGIC;
        DOUT : OUT STD_LOGIC_VECTOR(W+N-1 downto 0));
END acm;
```

```
architecture struct of acm is
COMPONENT lpm_add_sub
 GENERIC (LPM_WIDTH: POSITIVE;
  LPM_REPRESENTATION: STRING := "SIGNED";
  LPM_DIRECTION: STRING := "UNUSED";
  LPM_PIPELINE: INTEGER := 0;
  LPM_TYPE: STRING := "LPM_ADD_SUB";
  LPM_HINT: STRING := "UNUSED");
 PORT(dataa,datab: IN STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO
 0);
  aclr, clock, cin: IN STD_LOGIC := '0';
  clken, add_sub: IN STD_LOGIC := '1';
  result: OUT STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0);
  cout, overflow: OUT STD_LOGIC);
END COMPONENT;
COMPONENT lpm_shiftreg
 GENERIC (LPM_WIDTH: POSITIVE;
  LPM_AVALUE: STRING := "UNUSED";
  LPM_SVALUE: STRING := "UNUSED";
  LPM_PVALUE: STRING := "UNUSED";
  LPM_DIRECTION: STRING := "UNUSED";
  LPM_TYPE: STRING := "LPM_SHIFTREG";
  MAXIMIZE_SPEED: POSITIVE;
  LPM_HINT: STRING := "UNUSED");
 PORT(data: IN STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0) :=
(OTHERS => '0');
  clock: IN STD_LOGIC;
  enable, shiftin: IN STD_LOGIC := '1';
  load, sclr, sset, aclr, aset: IN STD_LOGIC := '0';
  q:OUT STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0); shiftout:
  OUT STD_LOGIC);
END COMPONENT;
signal sum: STD_LOGIC_VECTOR(W-1 downto 0);
signal tmp_out: STD_LOGIC_VECTOR(W+N-1 downto 0);
signal rst: STD_LOGIC;
BEGIN
rst <= NOT nReset;
adder: lpm_add_sub
GENERIC MAP(LPM_DIRECTION => "DEFAULT",
             LPM_PIPELINE => 1,
```

```
            LPM_REPRESENTATION => "SIGNED",
            LPM_WIDTH => W,
          MAXIMIZE_SPEED => 5)
PORT MAP(add_sub => add_sub,
          aclr => rst,
          clock => clk,
          dataa => sum,
          datab => X,
          result => tmp_out(W+N-1 downto N));

shifter: lpm_shiftreg
GENERIC MAP(LPM_DIRECTION => "RIGHT",
            LPM_WIDTH => N)
PORT MAP(shiftin => tmp_out(N),
          clock => clk,
          q => tmp_out(N-1 downto 0));

sum(W-1) <= tmp_out(W+N-1);
sum(W-2 downto 0) <= tmp_out(W+N-1 downto N+1);

DOUT <= tmp_out;

end struct;
```

Мегафункции находятся в библиотеке ***lpm***, которая подключается к проекту декларациями ***LIBRARY*** и ***USE***. Далее описан интерфейс проекта ***ENTITY*** : параметры блока (***GENERIC***) и порты (***PORT***). В структуре проекта приводят описание применяемых мегафункций:

```
COMPONENT lpm_add_sub
 GENERIC ( . . . ,
          . . . );
 PORT ( . . . ,
          . . . );
END COMPONENT;

COMPONENT lpm_shiftreg
 GENERIC ( . . . ,
          . . . );
 PORT ( . . . ,
          . . . );
END COMPONENT;
```

Затем приводится описание двух сигналов ***sum*** и ***tmp_out*** как стандартных логических векторов соответствующей разрядности, которая зависит от значения параметров ***N*** и ***W***.

В структурном описании проекта используемая мегафункция сумматора определяется таким образом:

```
adder: lpm_add_sub
GENERIC MAP ( . . . ,
              . . . )
PORT MAP    ( . . . ,
              . . . ));
```

где *adder* – имя блока в проекте (метка);

lpm_add_sub – имя мегафункции;

GENERIC MAP – это список фактически заданных параметров мегафункции;

PORT MAP – это карта описания сигналов, которые подключаются к портам блока.

Аналогичным образом в проект подключается мегафункция сдвигового регистра (метка *shifter*).

Переименование проводников шины (эквивалент примитивам *WIRE* на рис. 8.2) выполняется с помощью операторов назначения сигналов вида *sum(...) <= tmp_out(...)*.

Принцип работы масштабирующего аккумулятора поясняют его временные диаграммы работы, приведенные на рис. 8.3.

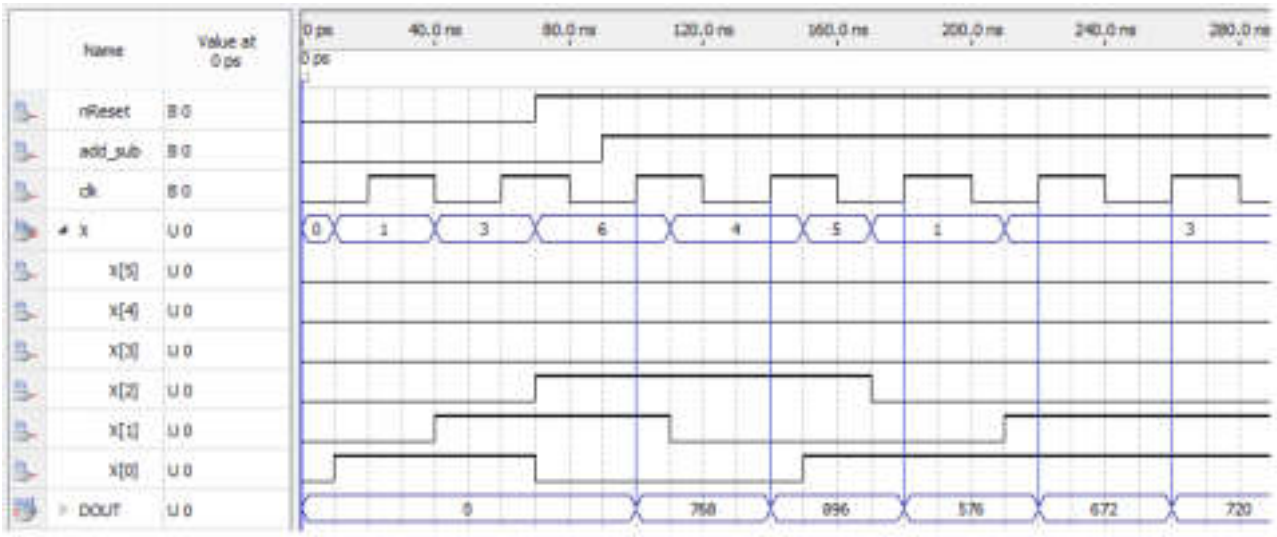


Рис. 8.3. Временные диаграммы работы масштабирующего аккумулятора

Из рис. 8.3 видно, что работа аккумулятора начинается после предварительного сброса и установки режима суммирования.

Карта примитивов и мегафункций в библиотеке схемного редактора

.../libraries/				
	megafunctions			
		arithmetic		
			lpm_add_sub	сумматор-вычитатель
		storage		
			lpm_shiftreg	регистр сдвига
	primitives			
		buffer		
			wire	
		logic		
			and2	элемент 2И
			and3	элемент 3И
			not	элемент НЕ
			or2	элемент 2ИЛИ

			or3	элемент 3ИЛИ
			xor	элемент Искл. ИЛИ
		pin		
			input	входной контакт
			output	выходной контакт
		storage		
			dff	D – триггер
			dffe	DE – триггер

Замечание: при вставке в схему библиотечных мегафункций флажки *Re-peat-insert Mode*, *Insert symbol as block* и *Launch MegaWizard Plug-In* должны быть НЕ установлены.

Установка параметров мегафункций производится щелчком правой кнопки мыши на символе блока и выбором пункта *Properties* в выпадающем динамическом меню.

Содержание отчета:

- листинг исходного файла описания блока (.bdf или .vhd);
- диаграмма результата временного моделирования (.vwf);
- символ функционального блока (.bsf)

Варианты индивидуальных заданий

Вариант	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
W	4	5	6	7	8	4	5	6	7
N	6	7	8	8	9	7	8	9	9
X	11	12	13	14	15	16	17	18	19

Контрольные вопросы

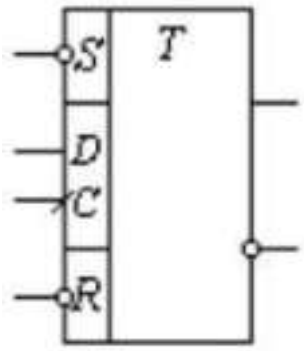
1. Что собой представляет масштабирующий аккумулятор?
2. Каково назначение выводов масштабирующего аккумулятора?
3. Какой состав блоков в масштабирующем аккумуляторе?
4. Что собой представляет временная диаграмма работы масштабирующего аккумулятора?

9. Моделирование цифровых схем на базе параметрических элементов (счетчик с заданным коэффициентом)

Краткая теория

Счетчики и регистры. Регистры и счетчики являются самыми распространенными элементами цифровой вычислительной техники. Они широко используются для построения устройств осуществляющих ввод, вывод и хранение информации, а также для выполнения некоторых арифметических и логических операций.

В качестве элементарных ячеек для построения счетчиков и регистров используют синхронные триггеры, переключение которых происходит только при наличии синхронизирующего сигнала (синхроимпульса) на специальном входе синхронизации «С».



Наиболее часто для построения регистров и счетчиков используется D-триггер, имеющий специальный информационный вход «D», и динамический вход синхронизации «С» (см. рис. 9.1).

Рис. 9.1. Схематичное представление D-триггера

Устройство, называемое *счетчиком*, предназначено для подсчета числа поступающих на вход импульсов в произвольной системе счисления. Двоичные счетчики строятся на основе триггеров, работающих в счетном режиме (Т-триггер, или счетный триггер).

Счетный триггер может быть получен из D-триггера путем соединения его инверсного выхода $\bar{Q}$ с входом D.

Схема счетного триггера и эпюры сигналов, поясняющие его работу, представлены на рис. 9.2.

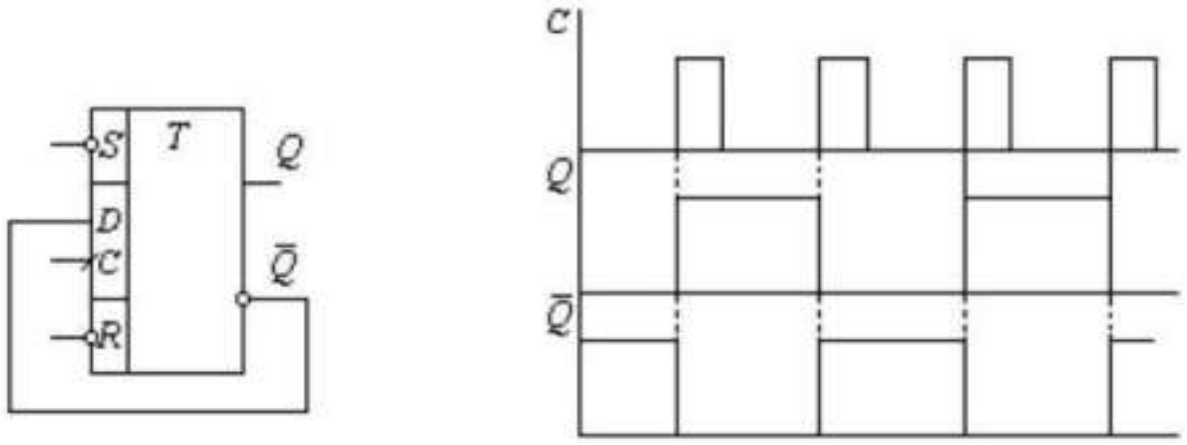


Рис. 9.2. Схема счетного триггера и временные диаграммы его работы

Принцип работы счетного триггера основан на том, что его состояние выхода изменяется на противоположное при поступлении на вход «С» каждого очередного счетного импульса.

Функциональная схема, составленная из D-триггеров, и условное графическое обозначение двоичного счетчика с коэффициентом пересчета 2^3 представлены на рис. 9.3.

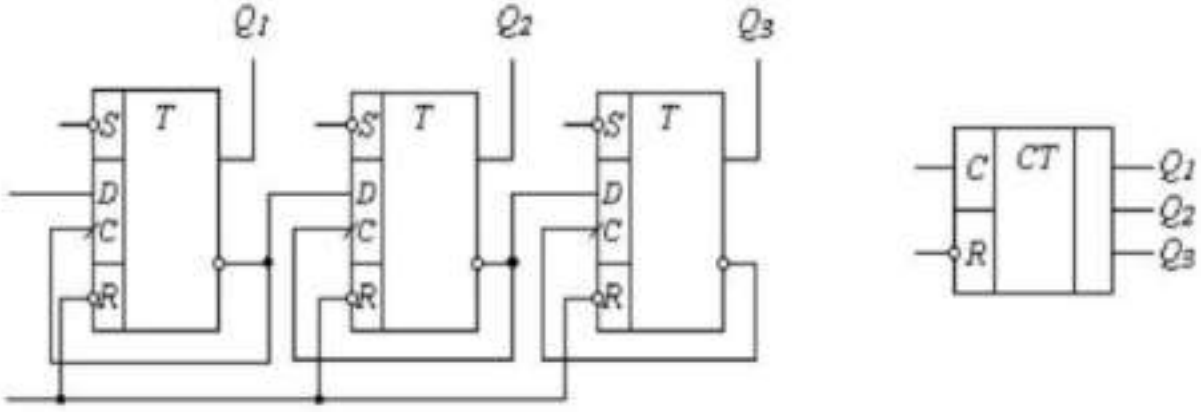


Рис. 9.3. Схема двоичного счетчика и его условное обозначение

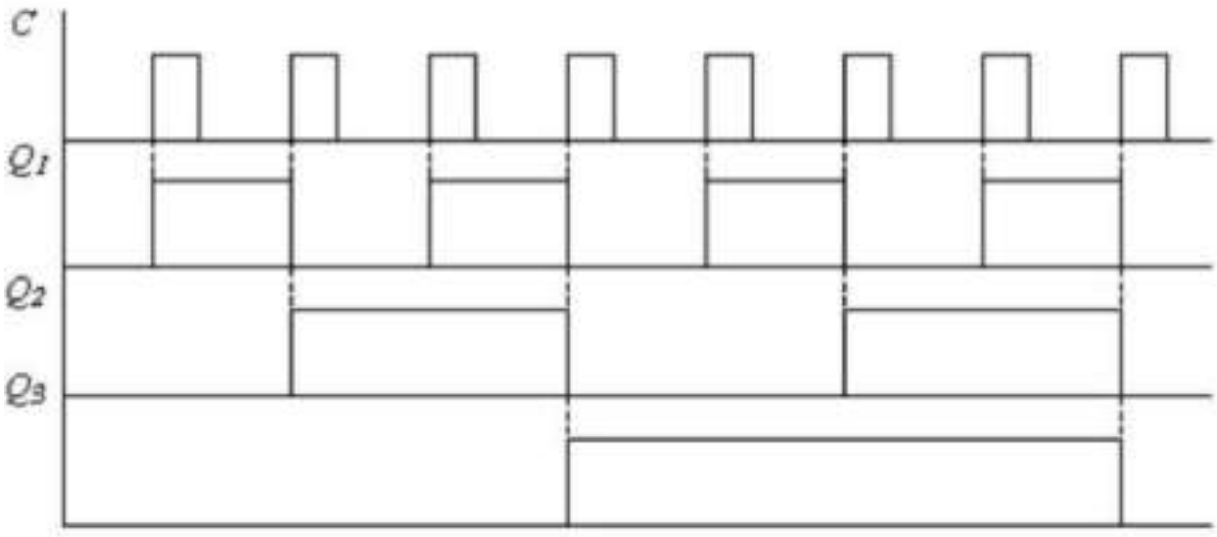


Рис. 9.4. Временные диаграммы работы двоичного счетчика

Каждый поступающий на вход счетчика импульс перебрасывает первый триггер в противоположное состояние (рис. 9.4). Сигнал с инверсного выхода предыдущего триггера является входным сигналом для последующего и, таким образом, комбинация сигналов на выходах Q1, Q2, Q3 будет соответствовать числу поступивших на вход счетчика импульсов, представленному в двоичном коде. Счетчик данного типа называется *асинхронным счетчиком*.

Если на счетный вход каждого последующего триггера счетчика подавать сигнал с прямого выхода предыдущего триггера, то счетчик будет производить операцию вычитания. Счетчики, способные выполнять функции сложения и вычитания, называются *реверсивными*.

Для построения счетчика с требуемым коэффициентом пересчета $К_С$, отличным от величины 2^N (N – число двоичных разрядов счетчика), используется принудительный сброс счетчика в исходное состояние при достижении счетчиком числа $К_С$.

Регистры служат в основном для хранения чисел в двоичном коде при выполнении над ними различных арифметических и логических операций. С помощью регистров выполняются такие действия над числами, как передача их из одного устройства в другое, арифметиче-

ский и логический сдвиг в сторону младших или старших разрядов, преобразование кода из последовательного в параллельный и наоборот и т. д.

Функциональная схема регистра сдвига вместе с его условным графическим обозначением представлена на рис. 9.5.

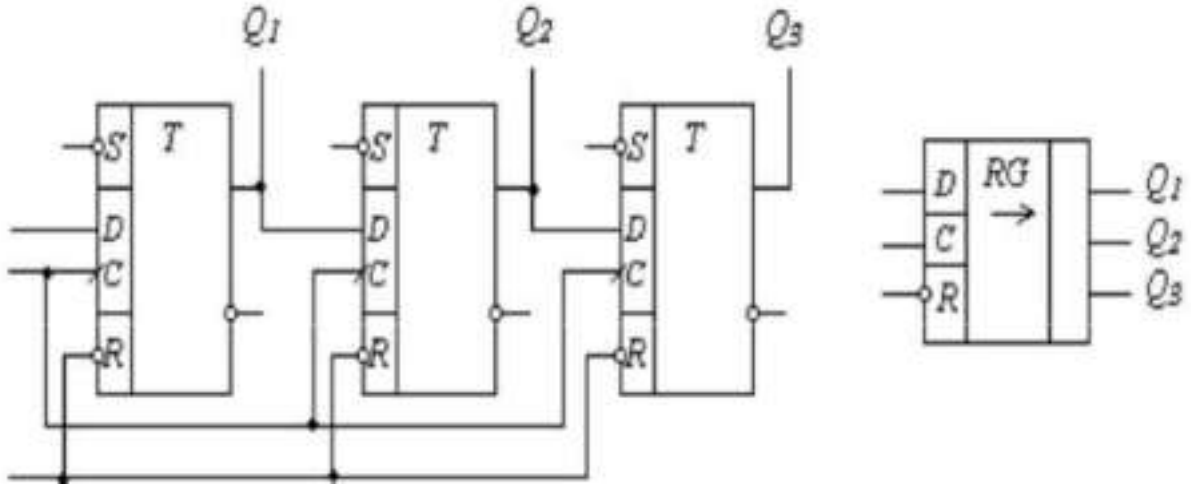


Рис. 9.5. Схема и обозначение регистра сдвига

Последовательный информационный код должен подаваться на вход «D» регистра. Импульс команды сдвига подается одновременно на синхронизирующие входы «С» всех триггеров регистра и переводит каждый триггер в состояние, в котором находился триггер предыдущего разряда. Таким образом, каждый импульс команды сдвига «продвигает» записываемое число на один разряд вправо.

При введении обратной связи в регистр сдвига, последний превращается в замкнутое кольцо, в котором под воздействием тактовых импульсов циркулирует введенная в регистр информация. Такие регистры называют кольцевыми счетчиками. Кодовая единица, введенная в первый триггер, циркулирует в течение всего времени существования тактовых импульсов, подаваемых на входы «С» всех триггеров счетчика. Приходящий тактовый импульс перебрасывает триггер, который был в состоянии «1», в состояние «0». Поскольку выход

«Q» этого триггера связан с входом «D» следующего триггера, то последний устанавливается в состояние «1» и т. д. Количество состояний такого счетчика равно числу триггеров.

Реализация проекта на параметрических элементах

Применение параметрических элементов Quartus II в разработке проектов цифровых схем рассмотрим на простом примере реализации 4-разрядного счетчика с задаваемым коэффициентом пересчета.

Создаем новый проект в Quartus II, затем в графическом редакторе создаем файл *.bdf и сохраняем его в предварительно созданном каталоге. Двойным щелчком правой кнопки мыши открываем меню ввода символов (Enter Symbol), выбираем библиотеку megafunctions и в ее подразделе arithmetic выбираем мегафункцию счетчика lpm_counter (рис. 9.6).

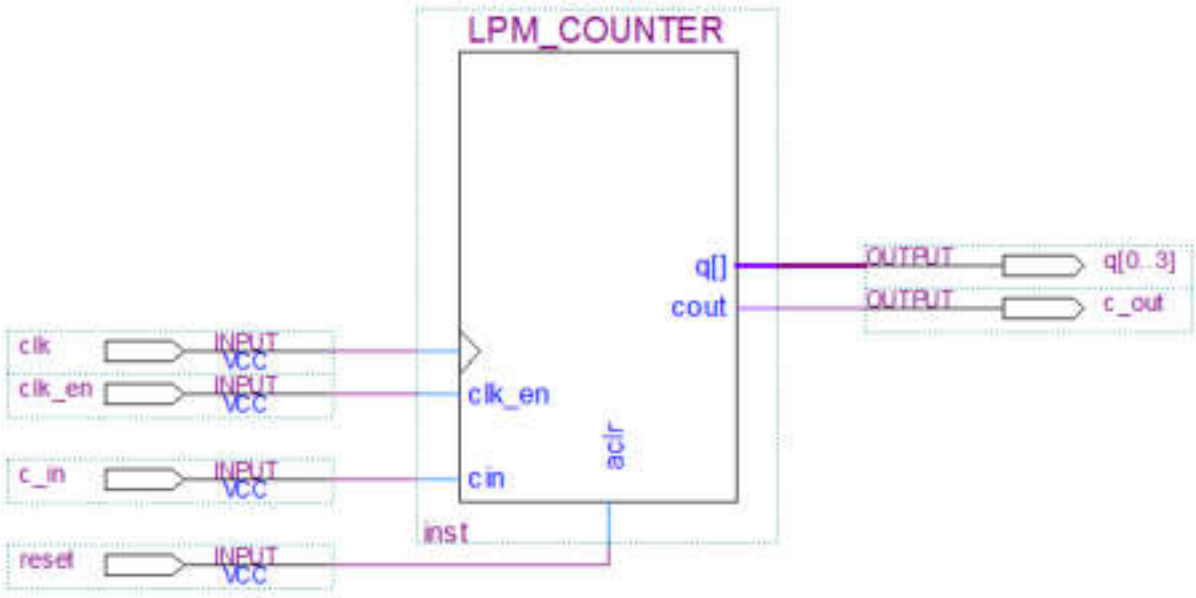


Рис. 9.6. Мегафункция универсального счетчика

При выборе элемента схемы правой кнопкой мыши появляется динамическое меню. Если выбрать пункт Properties, то появляется окно Symbol Properties , в котором можно редактировать использова-

ние портов счетчика (рис. 9.7), а также устанавливать параметры счетчика (рис. 9.8).

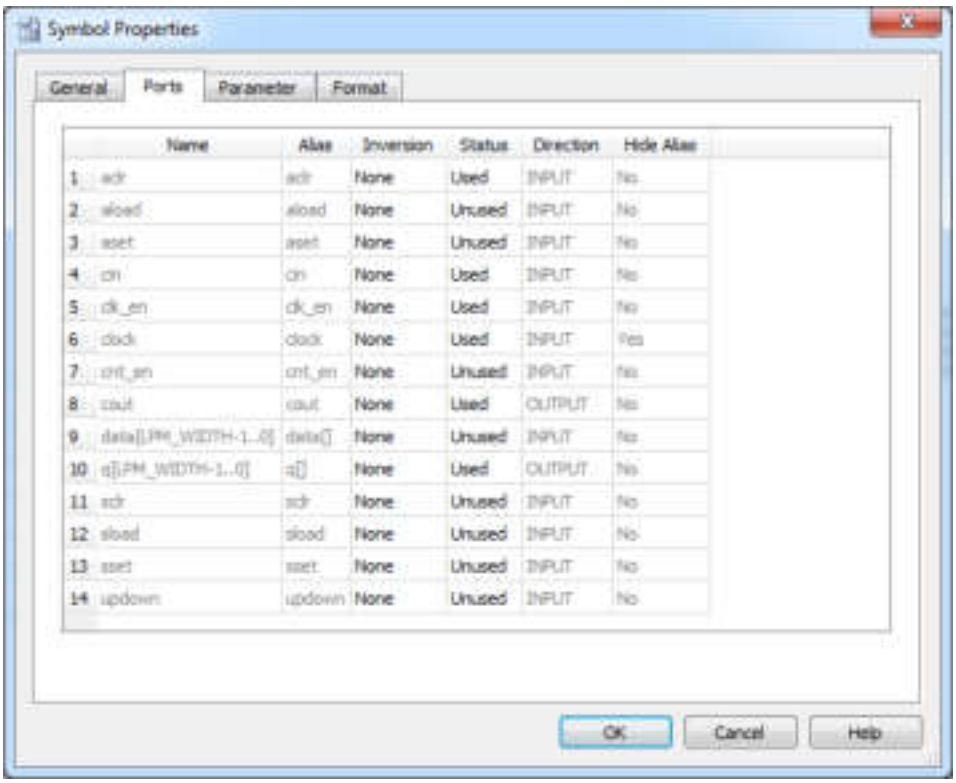


Рис. 9.7. Вкладка редактирования портов универсального счетчика

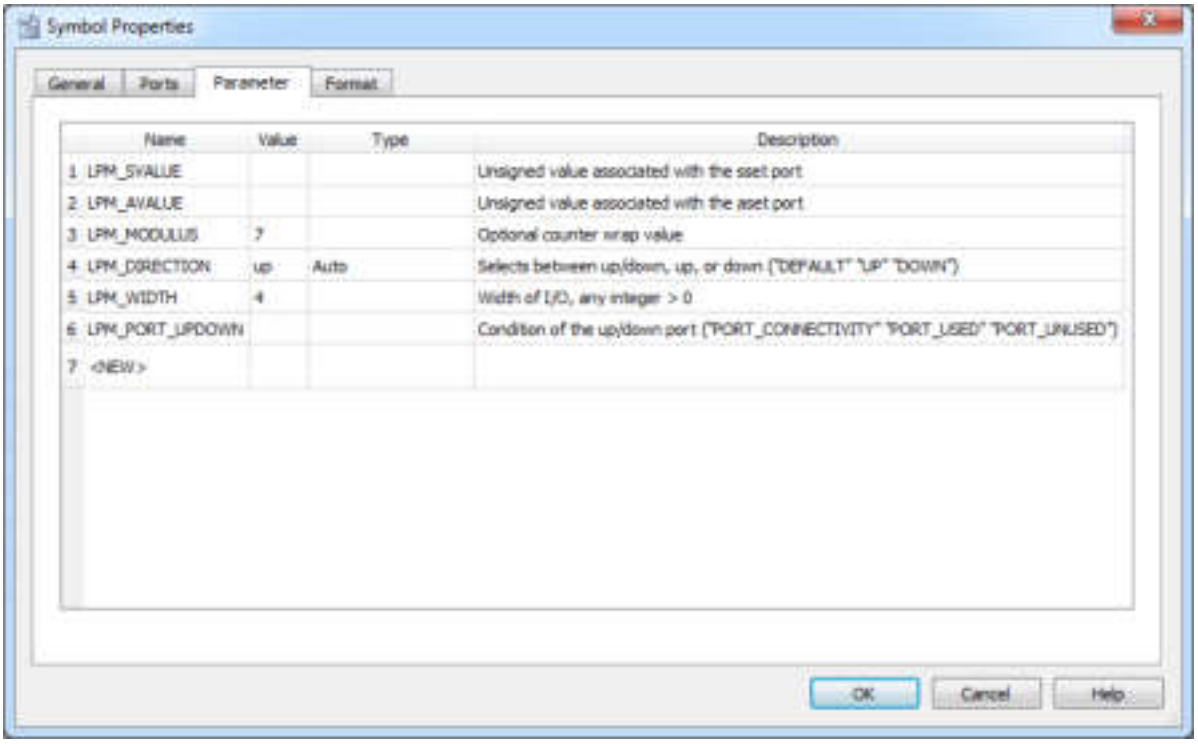


Рис. 9.8. Вкладка редактирования параметров универсального счетчика

Выбрав нужные порты счетчика и задав ему требуемые параметры, можно осуществить моделирование его работы. На рис. 9.9 представлен результат моделирования работы счетчика для случая, когда на его вход «c_in» подается произвольный сигнал.

Рис. 9.9. Диаграммы работы счетчика для произвольного входного сигнала

Если на вход «c_in» постоянно подается логическая «1», то информация на выходе «q» будет изменяться по переднему фронту каждого тактового импульса с учетом значения параметра LPM_MODULUS (рис. 9.10).

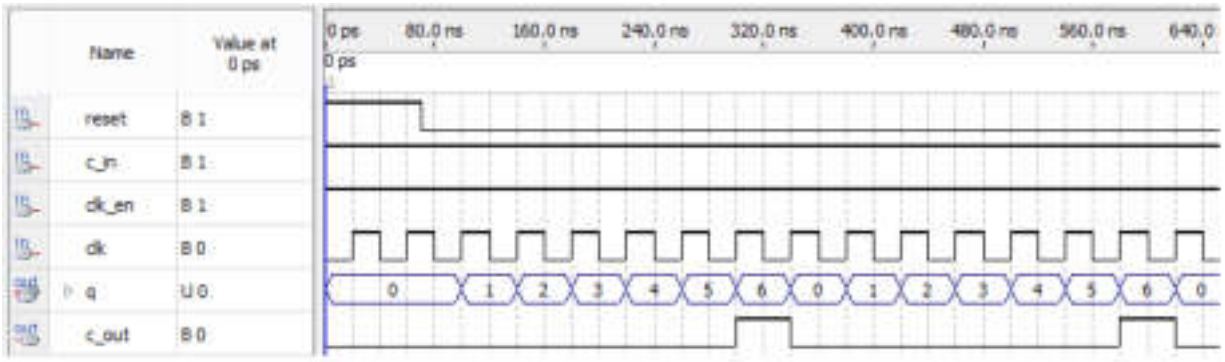


Рис. 9.10. Диаграммы работы счетчика с коэффициентом пересчета $К_С = 7$

Описание некоторых параметрических элементов САПР Quartus II представлено в приложении 1.

Задание к выполнению лабораторной работы:

1. Изучить принцип работы счетчиков импульсов.
2. Создать электрическую схему счетчика при помощи графического редактора САПР Quartus II с учетом варианта задания.
3. Произвести симуляцию работы схемы счетчика и зарисовать диаграммы его работы.

Содержание отчета:

- электрическая схема счётчика (файл *.bdf);
- диаграмма результата временного моделирования (.vwf);
- выводы по работе.

Варианты индивидуальных заданий

Вариант	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
LPM_MODULUS	11	12	13	14	15	6	7	8	9
LPM_DIRECTION	up	down	up	down	up	down	up	down	up
LPM_WIDTH	4	5	4	5	4	5	4	5	4

Контрольные вопросы

1. Объясните понятие «параметрический элемент». Какие параметрические элементы доступны в САПР Quartus II?
2. Объясните принцип работы счетчика построенного на триггерах. Какие типы счетчиков существуют?
3. Объясните назначение пунктов меню Symbol Properties.
4. Чем ограничивается максимальная скорость работы счетчика? Какова максимальная частота работы счетчика, разработанного в ходе выполнения лабораторной работы?

10. Моделирование цифровых схем с использованием параметрических элементов (реверсивный сдвиговый регистр)

Задание к выполнению лабораторной работы

1. На основе параметрического элемента LPM_SHIFTREG создать в Quartus II схему реверсивного сдвигового регистра с учетом варианта задания, указанного преподавателем.
2. Проверить работоспособность созданной схемы, выполнив симуляцию его работы с построением временных диаграмм.
3. На основе счетных триггеров создать в Quartus II схему реверсивного сдвигового регистра и проверить ее работоспособность, выполнив симуляцию с построением временных диаграмм.

В качестве примера рассмотрим выполнение задания на примере создания схемы 6-разрядного сдвигового регистра с возможностью параллельной загрузки (записи) данных.

Создаем проект в Quartus II аналогично предыдущей работе и в библиотеке megafunctions выбираем мегафункцию lpm_shiftreg , как показано на рисунке 10.1.

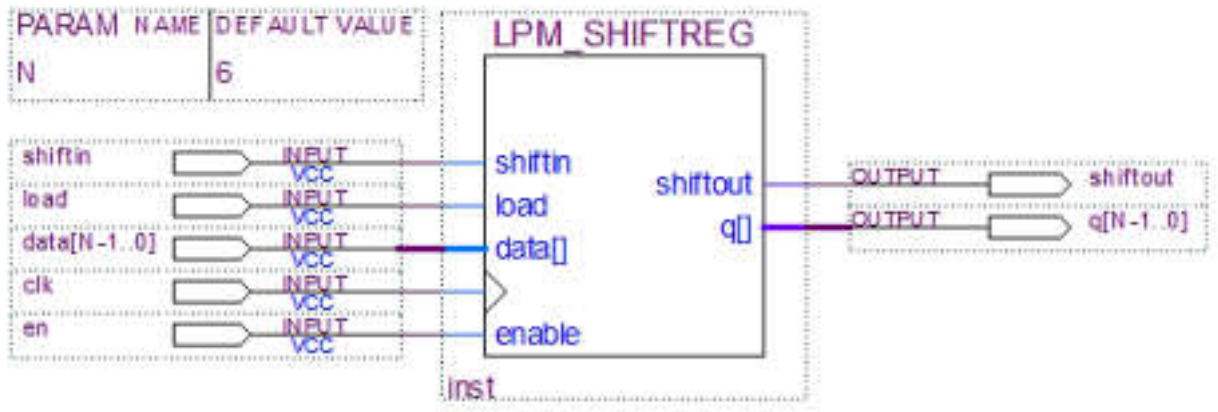


Рис. 10.1. Мегафункция реверсивного сдвигового регистра

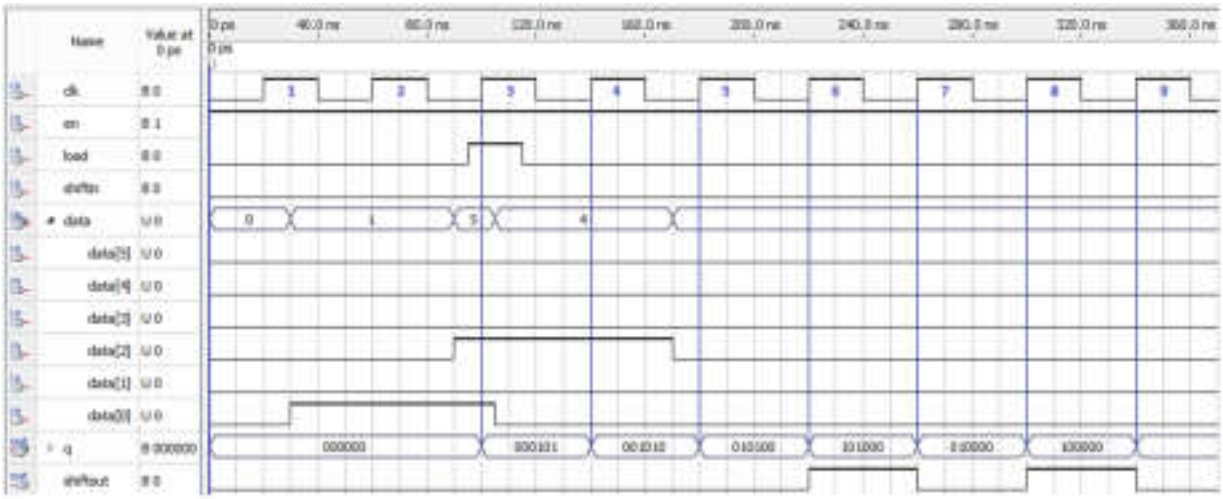


Рис. 10.2. Диаграммы работы сдвигового регистра

Проверка работоспособности схемы сдвигового регистра является частью функциональной верификации, под которой понимают проверку правильности выполнения функций системы. Эту проверку осуществляют подачей на входы соответствующих тестовых сигналов (информационных и управляющих), как показано на рис. 10.2. Затем осуществляют анализ временных диаграмм выходных сигналов и делают вывод о правильности выполнения функций устройства.

Из временных диаграмм рис. 10.2 видно, что запись данных в регистр осуществляется только при наличии активного сигнала load по переднему фронту третьего тактового импульса (на данной временной диаграмме это происходит в момент времени 100 *нс*).

Четвертый и последующие тактовые импульсы (в моменты времени 140 *нс* и 180 *нс*) осуществляют сдвиг записанной информации в сторону старших разрядов, что эквивалентно умножению записанного в регистр числа на два.

Для выполнения третьего пункта задания создаем новый проект в Quartus II и, используя библиотеку примитивов, собираем схему сдвигового 6-разрядного регистра из счетных триггеров (рис. 10.3).

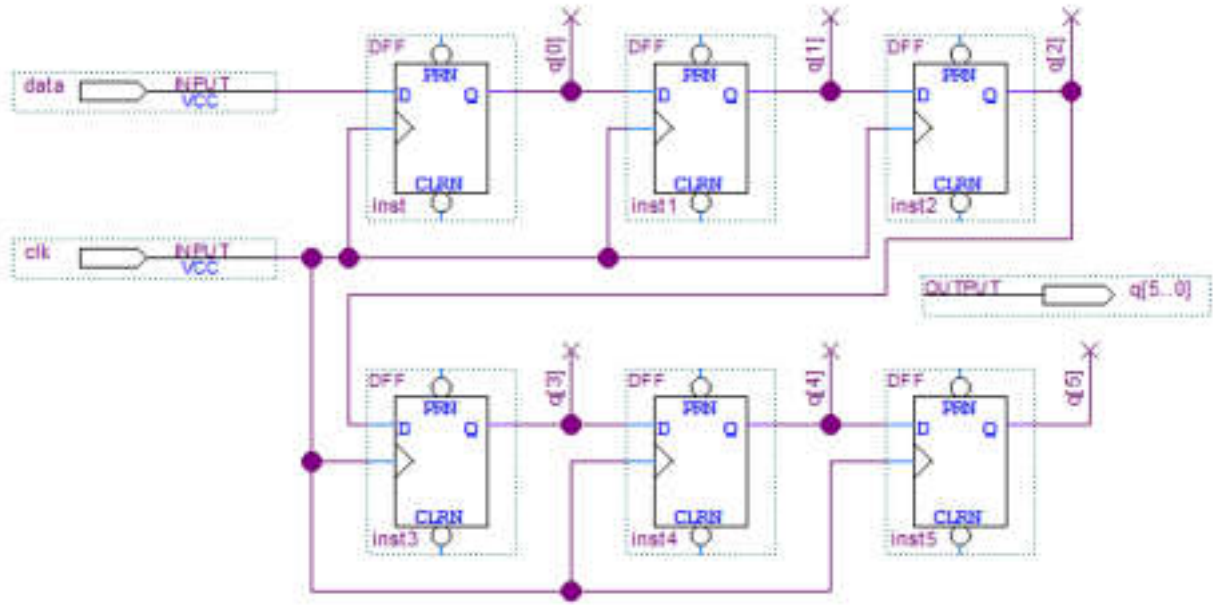


Рис. 10.3. Схема сдвигового регистра на счетных триггерах

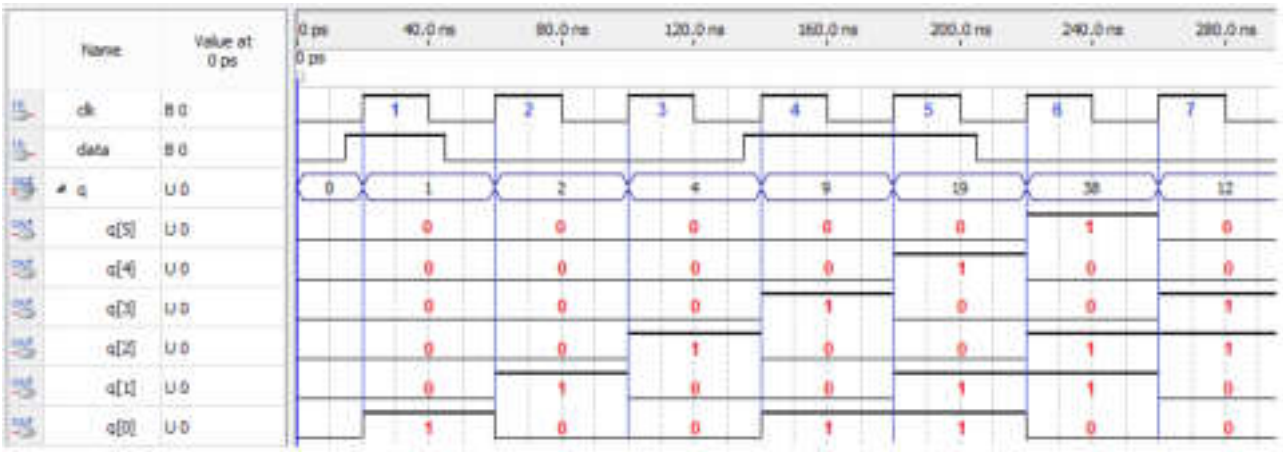


Рис. 10.4. Диаграммы работы сдвигового регистра на счетных триггерах

Работоспособность схемы регистра на счетных триггерах также проверяют подачей на входы соответствующих информационных и управляющих сигналов (рис. 10.4) с последующим анализом временных диаграмм выходных сигналов.

В данном примере на входной порт data последовательно подают двоичный код 10011 (число 19_{10}), начиная со старшего бита.

Из диаграмм видно, что полная запись входных данных в регистр осуществляется передним фронтом пятого тактового импульса.

Шестой тактовый импульс сдвигает записанный двоичный код в сторону старших разрядов, что эквивалентно умножению на 2 (19 · 2 = 38), образуя двоичный код 100110.

Седьмой тактовый импульс также сдвигает двоичный код в сторону старших разрядов и «1» из старшего разряда пропадает, так как не хватает разрядной сетки.

Содержание отчета:

- электрические схемы регистров (файлы *.bdf);
- диаграммы результатов временного моделирования (.vwf);
- выводы по работе.

Варианты индивидуальных заданий

Вариант	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Запись числа	21	22	23	24	25	26	27	28	29
Разрядность	7	8	7	8	7	8	7	8	7

Контрольные вопросы

1. Объясните назначение параметров регистра как параметрического элемента. Какие параметры использованы в лабораторной работе?
2. Объясните принцип работы регистра построенного на триггерах. Какие типы регистров существуют?
3. Каковы особенности реверсивного сдвигового регистра?
4. Чем ограничивается максимальная скорость записи информации в регистр? Какова максимальная частота работы регистров, разработанных в ходе выполнения лабораторной работы?

11. Проектирование схем на основе графа состояний (программируемый преобразователь кода)

Задание к выполнению лабораторной работы:

1. Согласно своему варианту графа состояний автомата разработать функциональную электрическую схему цифрового программируемого устройства преобразования кодов.
2. Создать проект в Quartus II и ввести разработанную схему.
3. Выполнить компиляцию и моделирование ее работы.
4. Проверить полученные результаты, сверив их с таблицей истинности устройства.

Порядок выполнения работы:

1. На основе исходного графа состояний (рис. 11.1) по своему варианту составить таблицу перекодировки состояний устройства.
2. Подставить новые значения состояний в исходный граф.
3. Составить таблицу истинности работы устройства.
4. По таблице истинности разработать схему устройства.
5. Создать папку с названием фамилии выполняющего работу.
6. Запустить Quartus II, создать проект в именной папке, используя в имени файла проекта свою фамилию (например, ivanov.gdf).
7. Выбрать пункт меню Assign/Device... В разделе Device Family указать Cyclone IV E, затем в разделе Devices выбрать тип ПЛИС, применяемой в проекте – EP4CE10F17C8.
8. Ввести схему в графическом редакторе и выполнить компиляцию проекта. После компиляции создается файл отчета *.rpt.
9. Далее необходимо создать файл для построения временных диаграмм. Для этого надо выбрать University Program VWF.
10. В окне «Waveform Editor» в графе Name ввести названия входов и выходов схемы, для которых строятся временные диаграммы. Необходимо отобразить выходы: q0, q1, q2, q3, q[3..0].

11. Ввести конечное время построения диаграммы, равное 4 *мкс*.
12. Подать синхросигнал и установить необходимое состояние входов A и B на временной диаграмме.
13. Выполнить симуляцию до сообщения: «Project simulation was successful» (имитация (симуляция) проекта прошла успешно).
14. В окне «Waveform Editor» появятся временные диаграммы работы устройства. Если последовательность на шине q[3..0] совпадает с заданной, то задание выполнено правильно.

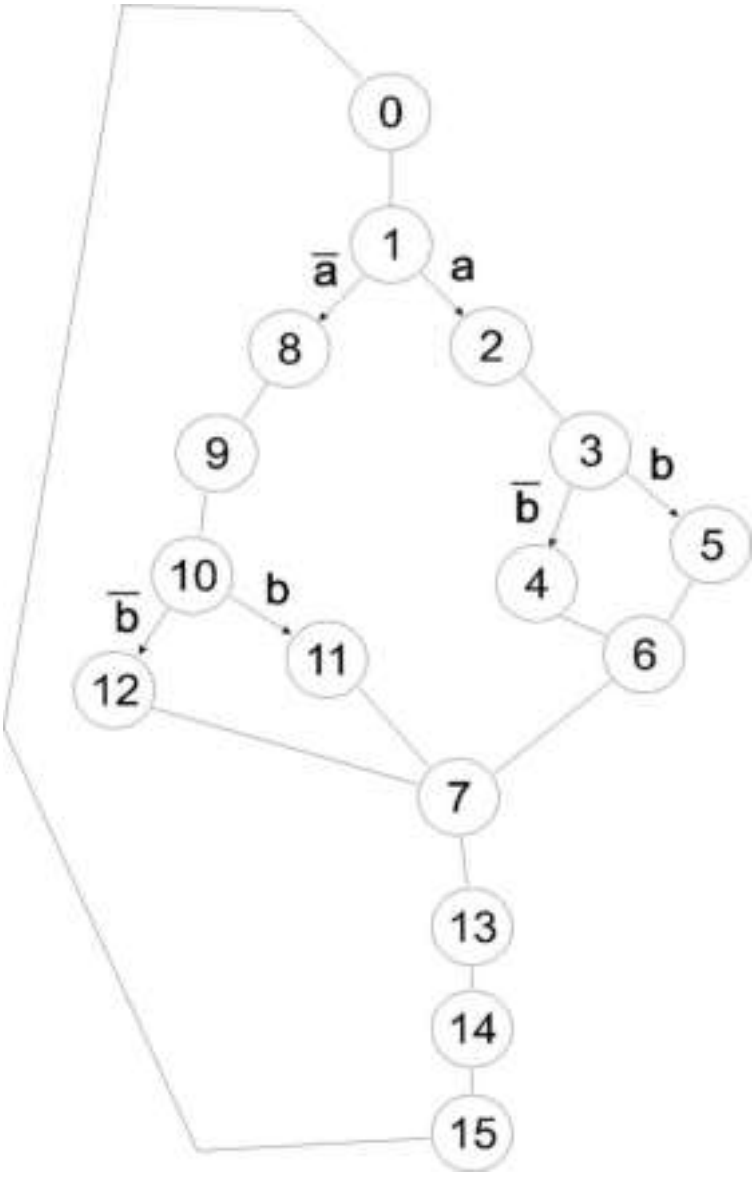


Рис. 11.1. Исходный граф состояния

Таблица 11.1

Варианты состояний графа

Номер вар.	Состояния графа															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
1	5	0	3	11	12	1	9	2	14	8	7	6	4	13	10	15
2	15	5	0	3	11	12	1	9	2	14	8	7	6	4	13	10
3	10	15	5	0	3	11	12	1	9	2	14	8	7	6	4	13
4	13	10	15	5	0	3	11	12	1	9	2	14	8	7	6	4
5	4	13	10	15	5	0	3	11	12	1	9	2	14	8	7	6
6	6	4	13	10	15	5	0	3	11	12	1	9	2	14	8	7
7	7	6	4	13	10	15	5	0	3	11	12	1	9	2	14	8
8	8	7	6	4	13	10	15	5	0	3	11	12	1	9	2	14
9	14	8	7	6	4	13	10	15	5	0	3	11	12	1	9	2
10	2	14	8	7	6	4	13	10	15	5	0	3	11	12	1	9
11	9	2	14	8	7	6	4	13	10	15	5	0	3	11	12	1
12	1	9	2	14	8	7	6	4	13	10	15	5	0	3	11	12
13	12	1	9	2	14	8	7	6	4	13	10	15	5	0	3	11
14	11	12	1	9	2	14	8	7	6	4	13	10	15	5	0	3
15	3	11	12	1	9	2	14	8	7	6	4	13	10	15	5	0
16	0	5	15	10	13	4	6	7	8	14	2	9	1	12	11	3
17	3	0	5	15	10	13	4	6	7	8	14	2	9	1	12	11
18	11	3	0	5	15	10	13	4	6	7	8	14	2	9	1	12
19	12	11	3	0	5	15	10	13	4	6	7	8	14	2	9	1
20	1	12	11	3	0	5	15	10	13	4	6	7	8	14	2	9
21	9	1	12	11	3	0	5	15	10	13	4	6	7	8	14	2
22	2	9	1	12	11	3	0	5	15	10	13	4	6	7	8	14
23	14	2	9	1	12	11	3	0	5	15	10	13	4	6	7	8
24	8	14	2	9	1	12	11	3	0	5	15	10	13	4	6	7
25	7	8	14	2	9	1	12	11	3	0	5	15	10	13	4	6
26	6	7	8	14	2	9	1	12	11	3	0	5	15	10	13	4
27	4	6	7	8	14	2	9	1	12	11	3	0	5	15	10	13
28	13	4	6	7	8	14	2	9	1	12	11	3	0	5	15	10
29	10	13	4	6	7	8	14	2	9	1	12	11	3	0	5	15

Пример выполнения задания

1. Составляем таблицу перекодировки состояний автомата и их двоичный код (таблица 11.2).
2. Подставляем новые значения из таблицы перекодировки в граф состояний (рис. 11.2).
3. Составляем таблицу истинности автомата (таблица 11.3).
4. После получения таблицы истинности автомата создается функциональная схема в САПР Quartus II без минимизации (рис. 11.3).
5. Временные диаграммы для этого варианта после компиляции проекта приведены на рис. 11.4.

Таблица 11.2

Таблица перекодировки состояний автомата и их двоичный код

Номер состояния	Номер состояния из таблицы 11.1	Двоичный код q3,q2,q1,q0
0	0	0000
1	3	0011
2	11	1011
3	12	1100
4	1	0001
5	9	1001
6	2	0010
7	14	1110
8	8	1000
9	7	0111
10	6	0110
11	4	0100
12	13	1101
13	10	1010
14	15	1111
15	5	0101

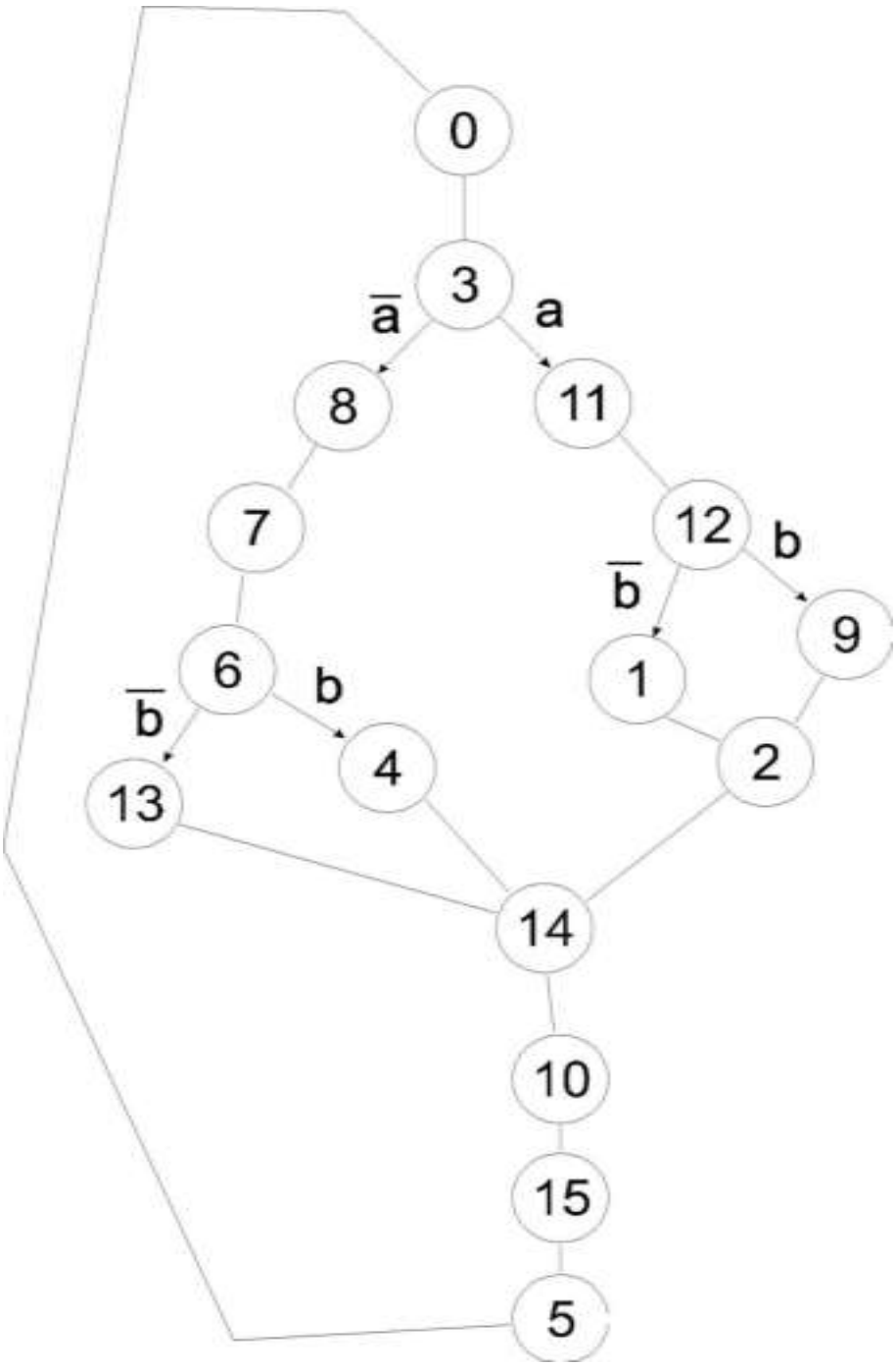


Рис. 11.2. Граф состояния, полученный с учетом таблицы перекодировки

Таблица 11.3

Таблица истинности автомата

старое состояние		условие	новое состояние	
№	Код		№	Код
0	0000	-	3	0011
3	0011	A=0	8	1000
3	0011	A=1	11	1011
8	1000	-	7	0111
7	0111	-	6	0110
6	0110	B=0	13	1101
6	0110	B=1	4	0100
13	1101	-	14	1110
14	1110	-	10	1010
10	1010	-	15	1111
15	1111	-	5	0101
5	0101	-	0	0000
11	1011	-	12	1100
12	1100	B=0	1	0001
12	1100	B=1	9	1001
1	0001	-	2	0010
2	0010	-	14	1110
4	0100	-	14	1110
9	1001	-	2	0010

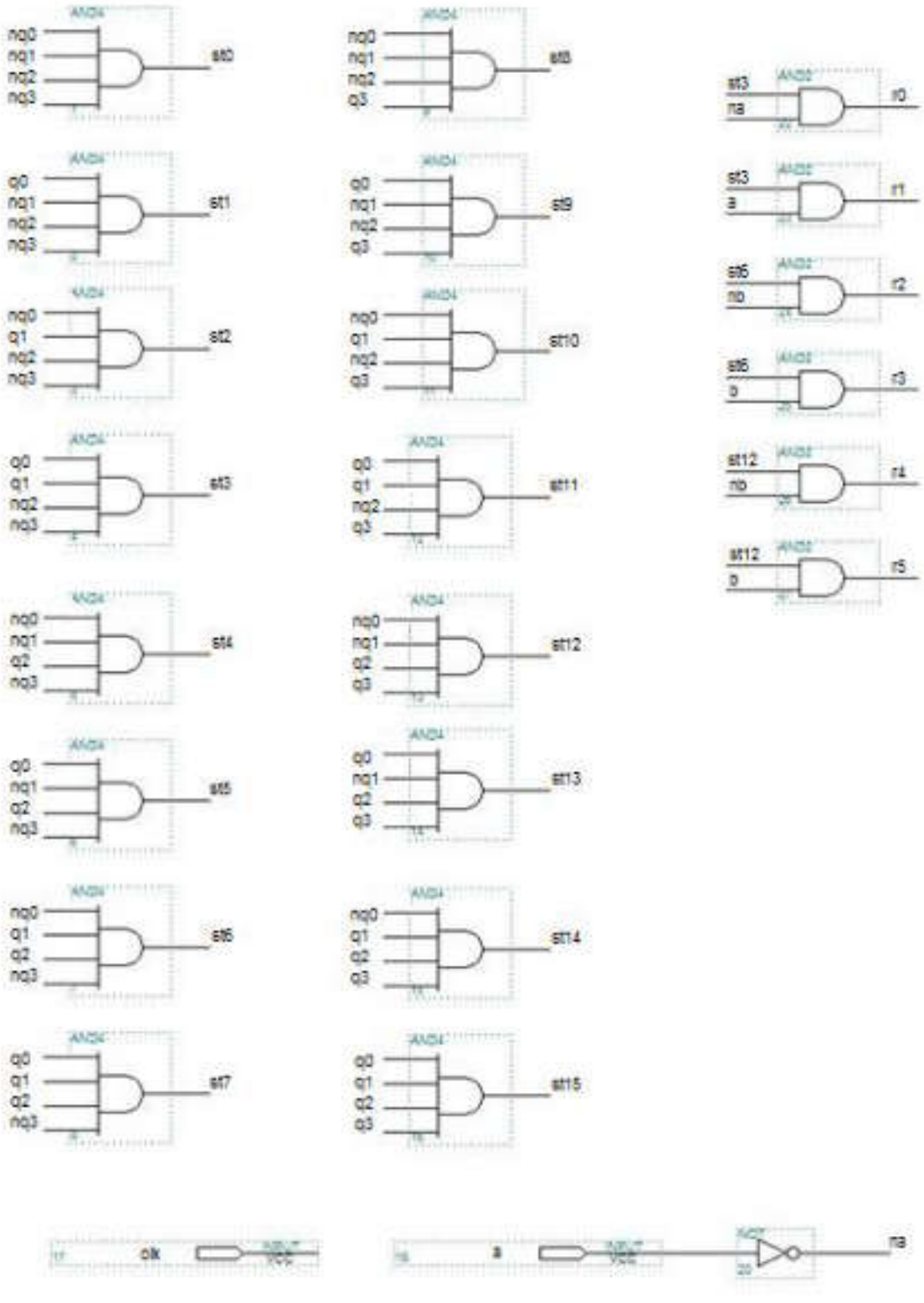


Рис. 11.3 (начало). Функциональная схема без минимизации

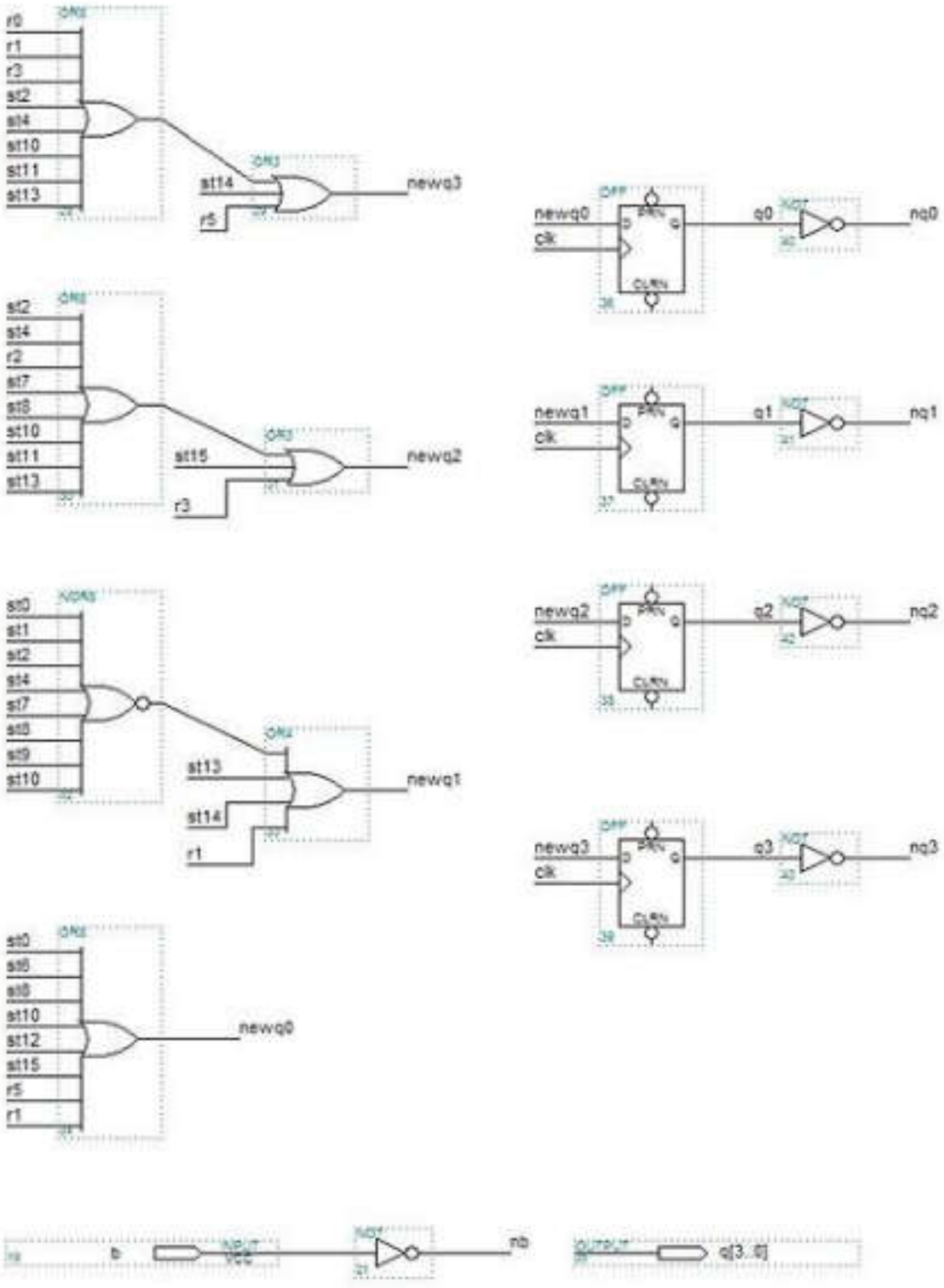


Рис. 11.3 (окончание). Функциональная схема без минимизации

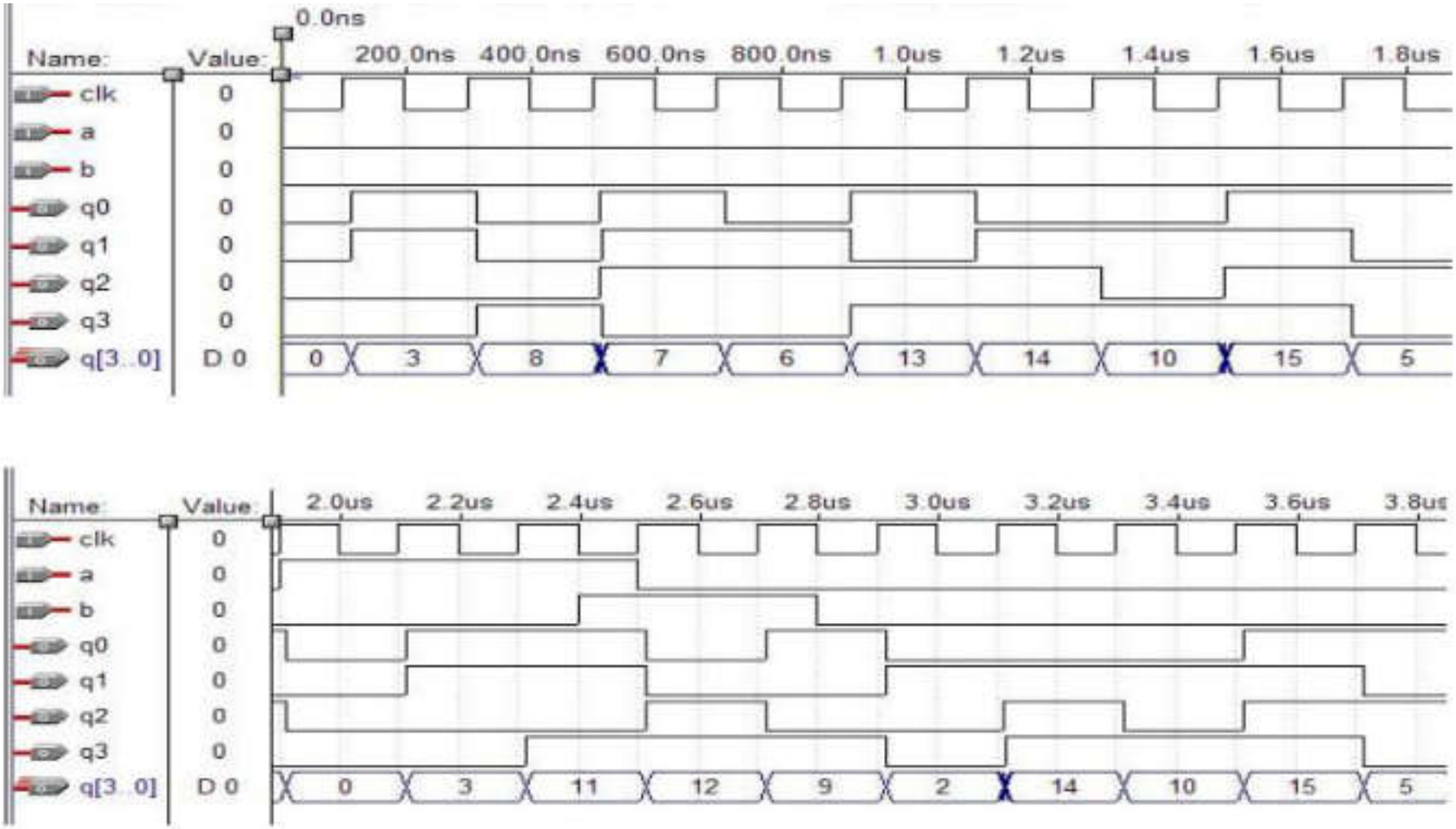


Рис. 11.4. Временные диаграммы работы

12. Синтез нерекурсивного КИХ-фильтра пятого порядка на ПЛИС

Краткая теория

Фильтры с конечной импульсной характеристикой (КИХ) часто используют в цифровой обработке сигналов. Это обусловлено тем, что его выходной сигнал конечен (успокаивается до нуля через некоторое время) и его фазовая характеристика линейна. Выходной сигнал КИХ-фильтра определяется с помощью следующего уравнения:

$$y[n] = \sum_{i=0}^{N} b_i x[n-i].$$

Здесь $y[n]$ – выходной сигнал фильтра; x – входной сигнал; b_i – коэффициенты фильтра.

N – порядок фильтра. Чем выше N, тем более сложным будет фильтр.

На рис. 12.1 представлен пример схемы КИХ-фильтра пятого порядка. Входной сигнал Xin поступает на блок MCM, где осуществляется его умножение на постоянные коэффициенты. Триггеры D1…D4 задерживают сигнал на один такт. Выходной сигнал формируется как сумма произведений постоянных коэффициентов на входной сигнал с временной задержкой на различное число тактовых периодов.

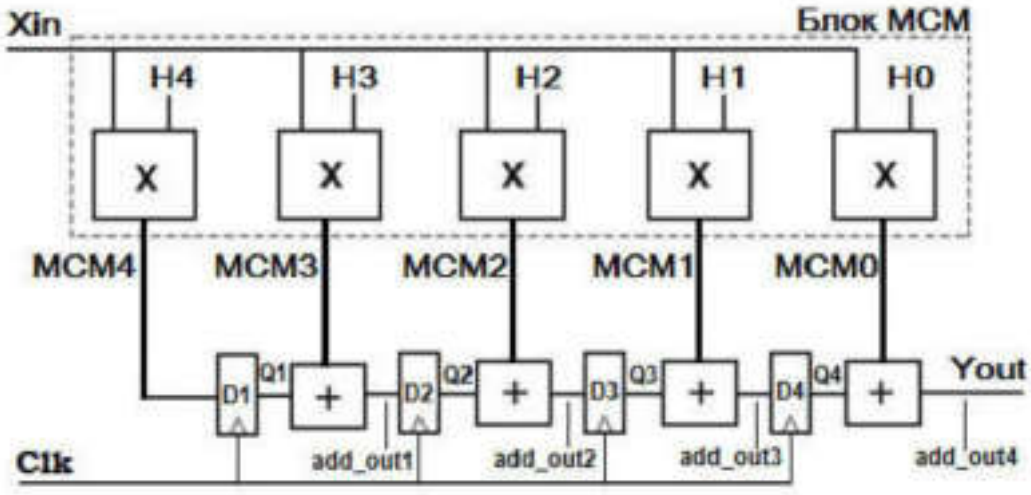


Рис. 12.1. Структурная схема КИХ-фильтра пятого порядка

Далее рассмотрим пример проекта КИХ-фильтра пятого порядка, который выполнен на языке VHDL в САПР Quartus II Web Edition версии 13.0 и отлажен при помощи симулятора ModelSim ASE 10.1d и программного пакета Scilab версии 5.5.2.

Рассмотрим особенности структуры проекта, а также опишем его основные параметры и используемые сигналы.

Сигнал Xin определен как signed (знаковый) разрядностью 8 бит, а сигнал Yout также типа signed, но разрядностью 16 бит.

Проект содержит два файла: основной файл fir_5tap.vhd со всеми умножителями и сумматорами, а также файл DFFx.vhd для определения операций триггерных задержек.

Листинг кода основного файла fir_5tap.vhd приведен ниже:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;          --описаны типы SIGNED

entity fir_5tap is
port( Clk : in std_logic;          -- тактовый сигнал
      Xin : in signed(7 downto 0);  -- 8-битный входной сигнал
      Yout : out signed(15 downto 0)  -- 16-битный выходной сигнал
      );
end fir_5tap;

architecture Behavioral of fir_5tap is
component DFFx is                  -- компонент задержки
  port(
      Q : out signed(15 downto 0);  -- выход на сумматор
      Clk :in std_logic;            -- тактовый вход
      D :in  signed(15   downto 0)  -- вход данных из блока MCM
      );
end component;
```

```
signal H0,H1,H2,H3,H4 : signed(7 downto 0) := (others => '0');
signal
        MCM0,MCM1,MCM2,MCM3,MCM4,add_out1,add_out2,add_out
        3,
       add_out4 : signed(15 downto 0) := (others => '0');
signal Q1,Q2,Q3,Q4 : signed(15 downto 0) := (others => '0');

begin ---------------------------- инициализация коэффициентов фильтра
H0 <= to_signed(28,8);
H1 <= to_signed(39,8);                -- преобразование в тип SIGNED
H2 <= to_signed(46,8);
H3 <= to_signed(39,8);
H4 <= to_signed(28,8);

---------- умножение входного сигнала на постоянные коэффициенты
MCM4 <= H4*Xin;
MCM3 <= H3*Xin;
MCM2 <= H2*Xin;
MCM1 <= H1*Xin;
MCM0 <= H0*Xin;

------------------------------------------------- формирование сумматоров
add_out1 <= Q1 + MCM3;
add_out2 <= Q2 + MCM2;
add_out3 <= Q3 + MCM1;
add_out4 <= Q4 + MCM0;

---------------------- триггерные операции (для формирования задержки)
dff1 : DFFx port map(Q1,Clk,MCM4);
dff2 : DFFx port map(Q2,Clk,add_out1);
dff3 : DFFx port map(Q3,Clk,add_out2);
dff4 : DFFx port map(Q4,Clk,add_out3);
```

```
-------- формирование выходного сигнала по переднему фронту тактов
process(Clk)
begin
  if(rising_edge(Clk)) then
       Yout <= add_out4;
  end if;
end process;
end Behavioral;
```

VHDL код компонента триггерных задержек DFFx приведен ниже:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;          --описаны типы SIGNED
entity DFFx is
  port(
       Q: out signed(15 downto 0);          -- выход на сумматор
       Clk : in std_logic;                  -- тактовый вход
       D: in  signed(15 downto 0)           -- вход данных от блока MCM
  );
end DFFx;
architecture Behavioral of DFFx is
signal qt : signed(15 downto 0) := (others => '0');
  begin
  Q <= qt;
process(Clk)
  begin
       if ( rising_edge(Clk) ) then
         qt <= D;
       end if;
end process;
end Behavioral;
```

Далее приводится код тестбенча для тестирования проекта. Процесс стимуляции основан на обработке входного тестового сигнала линейной частотной модуляции (ЛЧМ), сформированного в программном пакете Scilab. Вид сигнала ЛЧМ показан на рисунке 12.2.

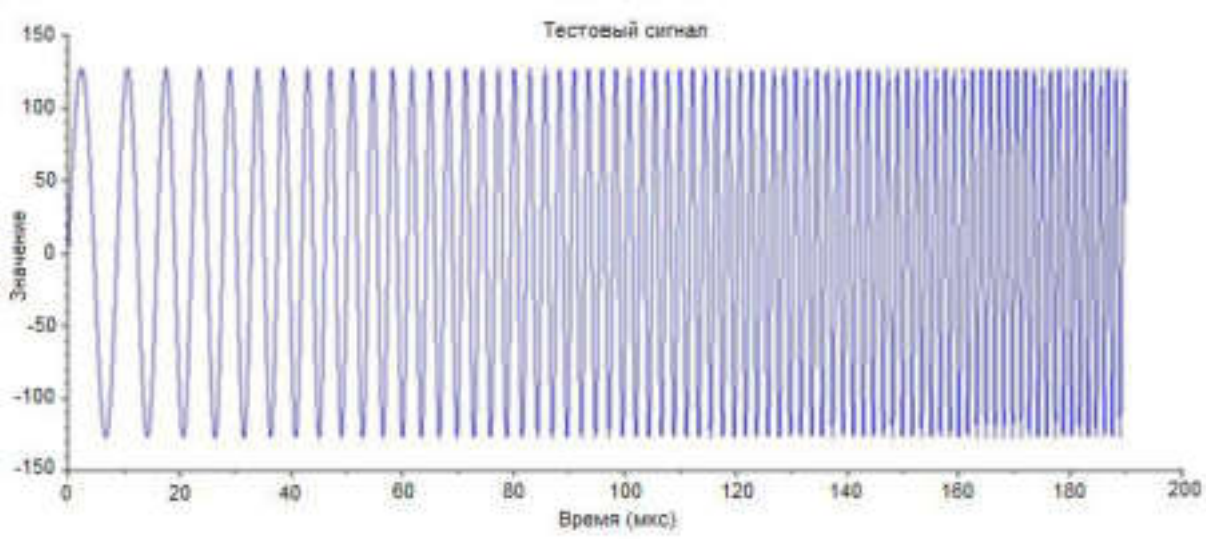


Рис.12.2. Сигнал линейной частотной модуляции

Этот тестовый сигнал записан во внешний текстовый файл var_sin.txt, который должен быть расположен в каталоге основных файлов проекта.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;                -- описаны типы SIGNED
USE std.textio.ALL;                      -- описаны файловые функции

ENTITY fir_5tap_tb IS                    -- это имя файла тестбенча
END fir_5tap_tb;

ARCHITECTURE behavior OF fir_5tap_tb IS
  signal Clk : std_logic := '0';
  signal Xin : signed(7 downto 0) := (others => '0');
  signal Yout : signed(15 downto 0) := (others => '0');
  constant Clk_period : time := 10 ns;
```

```
BEGIN  ---------------------------- инициализация модуля тестов (UUT)
  uut: entity work.fir_5tap PORT MAP (
       Clk => Clk,
       Xin => Xin,
       Yout => Yout  );
---------------------------------------- определения процесса тактирования
Clk_process:process
begin
  Clk <= '0';
  wait for Clk_period/2;
  Clk <= '1';
  wait for Clk_period/2;
end process;
------------------------------------------------------ процесс стимулирования
stim_proc: process
  file infile : text;
  variable s : line;
  variable s_data : integer range -128 to 127;
begin
  file_open(infile,"var_sin.txt",READ_MODE);
  wait for Clk_period*1;
  while not endfile(infile) loop
       readline(infile,s);
       read(s,s_data);
       Xin <= to_signed(s_data,8);
       wait for Clk_period*1;
  end loop;
file_close(infile);
wait;
end process;
END;
```

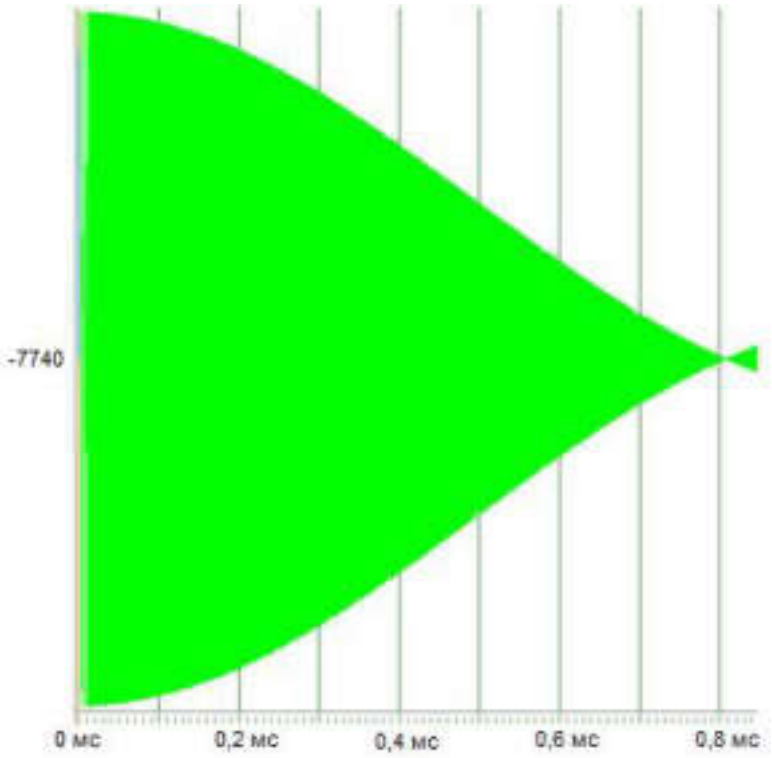


Рис. 12.3. Результаты моделирования АЧХ КИХ-фильтра в программе ModelSim

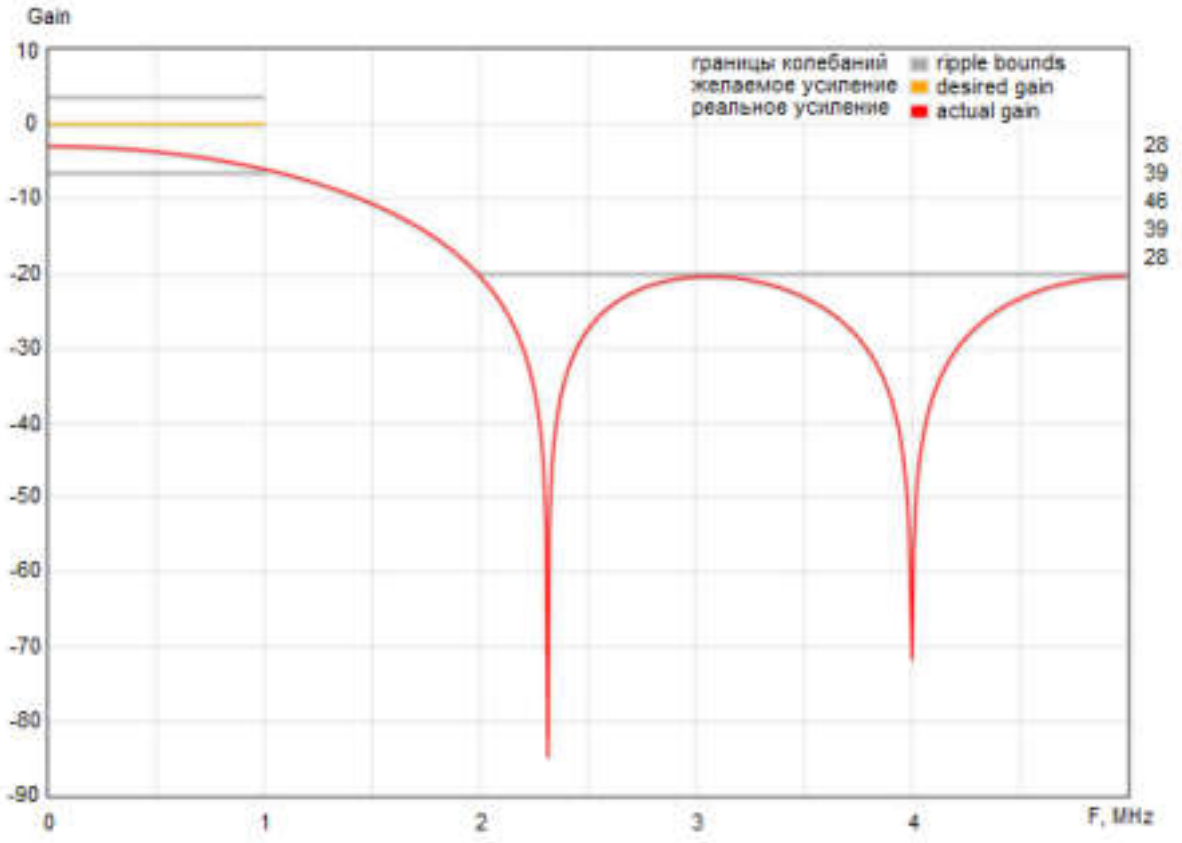


Рис. 12.4. АЧХ КИХ-фильтра с коэффициентами (t-filter.engineerjs.com)

Порядок выполнения работы

1. Скопировать файлы проекта в отдельную папку на диске D. Путь к файлам проекта рекомендуется делать коротким. Не допускается использовать в имени пути пробелы и кириллицу.
2. Изменить в главном файле fir_5tap.vhd коэффициенты H0…H4, согласно варианту задания, и выполнить его сохранение.
3. Запустить программу ModelSim ASE 10.1d и создать новый проект (File -> New -> Project…), указав в диалоговом окне Create Project имя проекта fir_5tap и путь к каталогу с его файлами. Остальные параметры оставить без изменения.
4. В появившемся окне Add items to the Project выбрать Add Existing File и добавить три файла в проект: fir_5tap.vhd, DFFx.vhd и fir_5tap_tb.vhd, после этого нажать Close.
5. Вопросительные знаки в колонке Status означают, что над файлами проекта не осуществлялось никаких действий. Вначале необходимо выполнить компиляцию файлов проекта. Для этого нужно выделить файл и в динамическом меню выполнить: Compile -> Compile All, после чего вопросительные знаки в колонке Status должны исчезнуть.
6. Выделить файл fir_5tap_tb.vhd и через главное меню выполнить команду Simulate -> Start Simulation.
7. В окне Start Simulation раскрыть work и выделить fir_5tap_tb.vhd, в подокне Resolution указать *ns* и нажать OK.
8. Раскрыть вкладку Wave и перенести туда сигналы Clk, Xin и Yout, используя в динамическом меню команду Add Wave.
9. Выделить в окне Wave сигналы Xin, Yout и изменить их представление, используя в динамическом меню команду Radix -> Decimal.
10. Запустить симуляцию: Simulate -> Run -> Run 100, перейти в командное окно Transcript, набрать команду run @1 ms и нажать Enter.

11. Выделить в окне Wave сигнал Yout, изменить его формат, используя в динамическом меню команду Format -> Analog и изменением масштаба («+» «-») добиться приемлемой картинки. Если частоты среза явно не видны, то следует запустить Scilab, откорректировать в файле var_sin.sce начальную частоту Fo и скорость её изменения Vf, а затем выполнить запуск на исполнение. Полученный новый файл var_sin.txt перезаписать в каталог проекта.

Варианты заданий

Номер варианта	1	2	3	4	5	6	7	8	9	10
H0	-27	-64	36	98	75	47	14	-45	-85	49
H1	-13	-39	58	61	37	21	35	-22	-52	18
H2	13	-11	95	29	16	-21	84	22	-31	-18
H3	27	11	58	61	-16	-47	35	45	31	-49
H4	42	39	36	98	-37	-72	14	91	52	-74

Содержание отчета

- структурная схема исследуемого КИХ-фильтра 5-го порядка;
- листинг файла fir_5tap_tb.vhd с измененными коэффициентами;
- листинг файла var_sin.sce с частотой Fo и скоростью Vf;
- диаграмма выходного сигнала Yout фильтра (имитация его АЧХ);
- выводы по работе на основе диаграммы выходного сигнала.

Контрольные вопросы

1. Назначение постоянных коэффициентов фильтра.
2. Принцип работы КИХ-фильтра.
3. Особенности цифровых фильтров по сравнению с аналоговыми.
4. Ограничение полосы рабочих частот в цифровых фильтрах.
5. Преимущества аппаратной реализации цифровых фильтров по сравнению с программной реализацией.

13. Управление обработкой данных с помощью процессора Nios II

Краткая теория

Процессор Nios II – это ядро универсального RISC процессора, которое обеспечивает следующие возможности:

- Полный набор 32-битных инструкций
- 32 универсальных регистра
- Опциональный набор теневых регистров
- 32 источника прерываний
- Внешний контроллер прерываний для увеличения их источников
- Одиночную инструкцию умножения 32×32 и деления
- Специальные инструкции для 64-битных и 128-битных вычислений над результатами умножения
- Инструкции для операций с плавающей точкой
- Одиночную инструкцию для многорегистрового сдвига
- Доступ к периферии и интерфейсам с внешней памятью
- Аппаратный отладочный модуль, позволяющий запускать, останавливать процессор и пошагово выполнять программы
- Опциональный диспетчер памяти (MMU) для поддержки операционных систем, которым требуется MMU
- Опциональный элемент защиты памяти (MPU)
- Программную среду разработки, основанную на инструменте GNU C/C++ и инструменте разработки программы Eclipse
- Интеграцию со встроенным логическим анализатором Altera SignalTap® II с возможностью анализа в реальном времени инструкций и данных вместе с другими сигналами в проекте FPGA.
- Структуру системы команд (ISA), совместимую со всеми процессорными системами Nios II.
- Рабочие характеристики до 250 DMIPS.

Система с процессором Nios II – это эквивалент микроконтроллеру или «компьютеру в чипе», который содержит процессор и комбинацию периферии и памяти на одном чипе. Система с процессором Nios II состоит из ядра процессора Nios II, набора периферии на чипе, памяти на чипе и интерфейса с внешней памятью. Аналогично семейству микроконтроллеров система с процессором Nios II использует постоянный набор инструкций и модель программирования.

Работа с процессором Nios II похоже на работу с любым другим микроконтроллером. Пакет разработки Nios II EDS содержит два близких инструмента разработки программы:

- инструменты разработки программы под Nios II;
- инструменты разработки программы Eclipse для Nios II.

Оба инструмента основываются на компиляторе GNU C/C++. Инструменты разработки программы Eclipse для Nios II предлагают устоявшуюся среду для программной разработки. Используя инструменты разработки Eclipse для Nios II, можно непосредственно начать разработку и симуляцию приложений программы Nios II.

Nios II EDS содержит инструменты разработки программ под Nios II. Эти инструменты имеют интерфейс командной строки.

Можно использовать специальные выводы и логические ресурсы на чипе под функции, не имеющие отношения к процессору. Эти ресурсы предоставляют специальные вентили и регистры в качестве связующей логики для проекта печатной платы. В большинстве проектов система с процессором Nios II занимает около 5% ресурсов ПЛИС, оставляя остальные ресурсы для реализации других функций.

Процессор Nios II – это процессор с конфигурируемым программным ядром (в микроконтроллерах с фиксированным ядром). В этом контексте возможность конфигурации позволяет добавлять или удалять средства в базисе системы в системе для решения конкретных задач. Программное ядро позволяет процессорному ядру не ограни-

чиваться одним кристаллом, а размещаться в любых чипах семейств Altera FPGA.

Гибкий набор периферии – это самое главное отличие процессора Nios II от обычных микроконтроллеров. Процессор Nios II реализуется в программируемой логике, поэтому можно создать нестандартную процессорную систему с точным требуемым набором периферии. Следствием гибкости периферии является гибкость адресного пространства.

Существуют два общих класса периферии: стандартная периферия и собственная периферия.

К стандартной периферии относятся: таймеры, последовательные интерфейсы, стандартный ввод/вывод, контроллеры SDRAM и прочие интерфейсы с памятью.

Можно создавать и свою собственную периферию и интегрировать ее в Nios II. Для систем, расходующих большое количество циклов для выполнения части кода, обычной практикой является создание собственной периферии, которая реализует этот код аппаратно.

Также как и собственную периферию, можно создавать и собственные инструкции, улучшающие характеристики системы пополнением процессора дополнительными устройствами. Собственная логика интегрируется в АЛУ процессора Nios II. Также как и обычные инструкции Nios II, логика собственных инструкций основана на работе с двумя регистрами и опционально возвращает результат в назначенный регистр.

В отношении программы собственные инструкции применяются как ассемблерные макросы или С-функции, т. е. программисту не нужно понимать язык ассемблера для использования собственных инструкций Nios II.

Инструмент разработки Altera’s SoPC Builder (Qsys) полностью автоматизирует процесс конфигурирования средств Nios II и генера-

цию аппаратной части проекта. Графическая оболочка (GUI) SoPC Builder позволяет конфигурировать системы Nios II с любым количеством периферии и интерфейсов с памятью. Инструмент SoPC Builder может также импортировать HDL файлы проекта, предлагая простой механизм для интегрирования собственной логики в систему Nios II.

Инструкция по работе с Nios II

1. Создайте новый проект для ПЛИС и добавьте к нему схему (bdf), как entity top level. После этого нужно добавить процессор Nios и его окружение. Для этого откройте оболочку Qsys: Tools → Qsys.

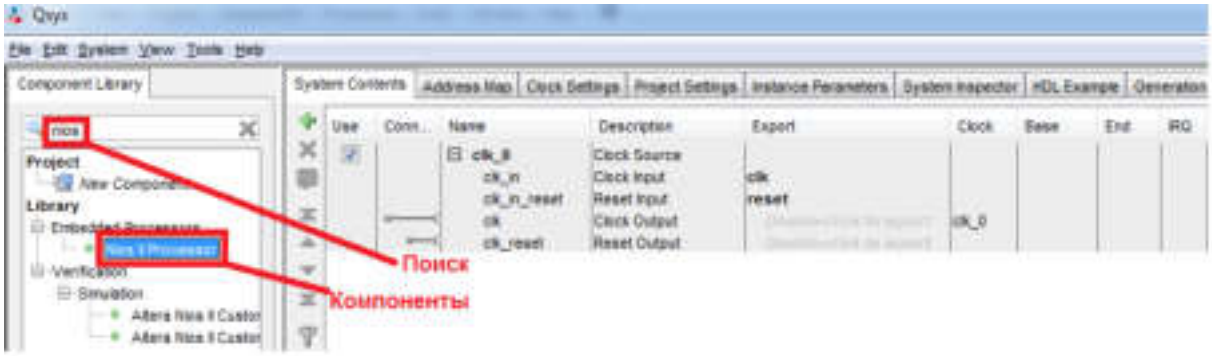


Рис. 13.1. Выбор встроенного процессора Nios

В окне компонентов, используя возможность поиска, находим Nios II Processor, как показано на рисунке 13.1, и кликаем мышью по нему дважды для выбора. В следующем окне нужно выбрать самую простую конфигурацию процессора (рис. 13.2) и нажать Finish.

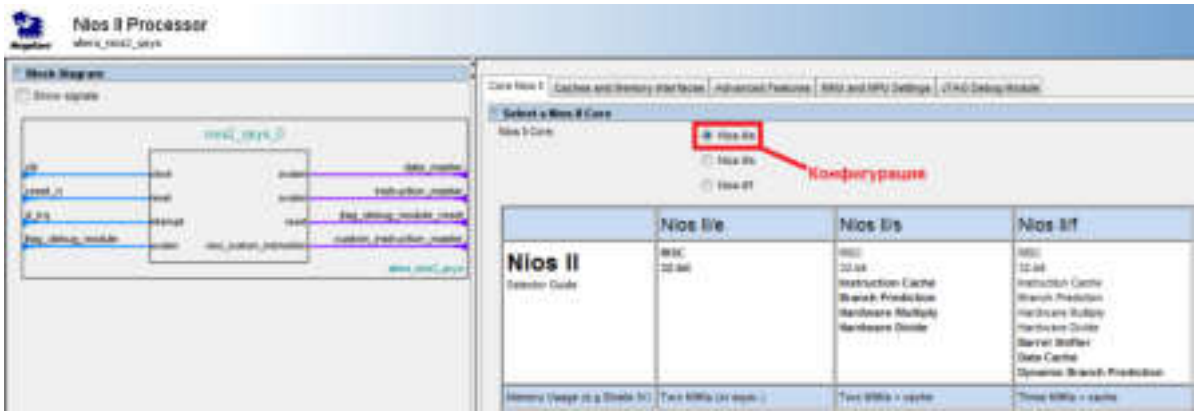


Рис. 13.2. Выбор простой конфигурации Nios

2. Далее набираем в окне поиске on-chip и добавляем в схему память On-Chip Memory (RAM or ROM), как показано на рисунке 13.3.

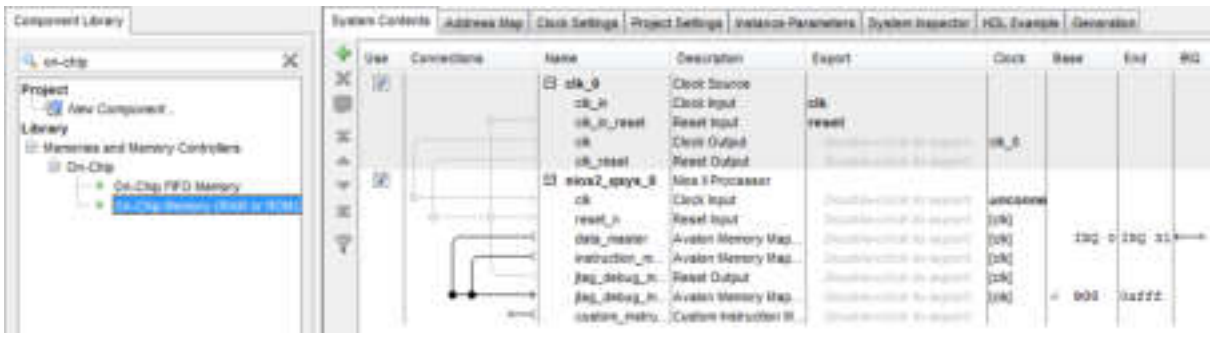


Рис. 13.3. Выбор компонента памяти On-Chip Memory для Nios

Выбираем размер памяти 20480 байта, как показано на рисунке 13.4, и нажимаем кнопку Finish.

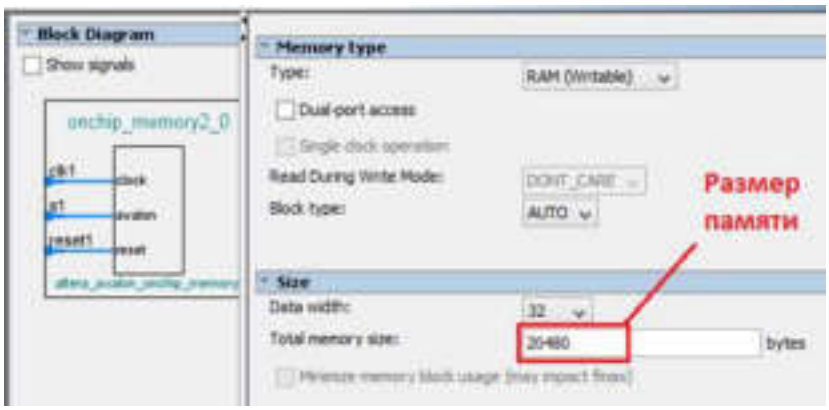


Рис. 13.4. Выбор размера памяти On-Chip Memory для Nios

3. Затем аналогично добавляем параллельный порт ввода-вывода PIO. Делаем его 4-разрядным и настраиваем на вывод, как показано на рисунке 13.5, (предполагается управление 4-мя светодиодами).

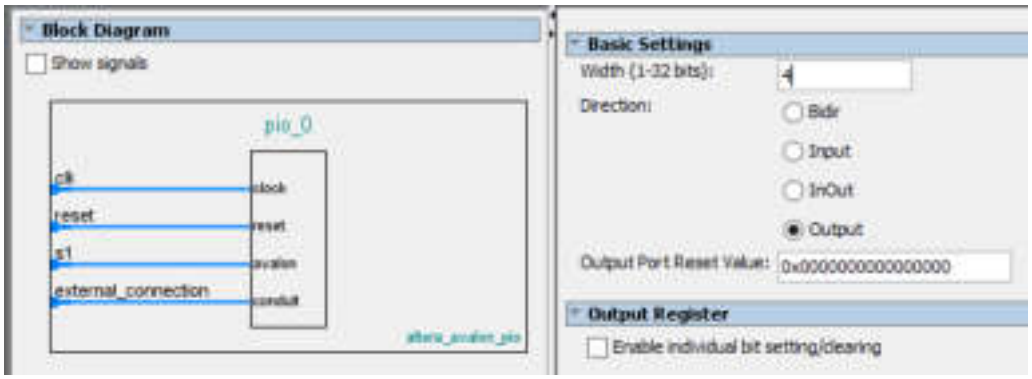


Рис. 13.5. Выбор разрядности порта ввода-вывода

4. Последним компонентом добавляем JTAG с настройками по умолчанию. Рекомендуется переименовать модули для упрощения их восприятия. В итоге должна получиться схема соединений портов компонентов, показанная на рисунке 13.6.

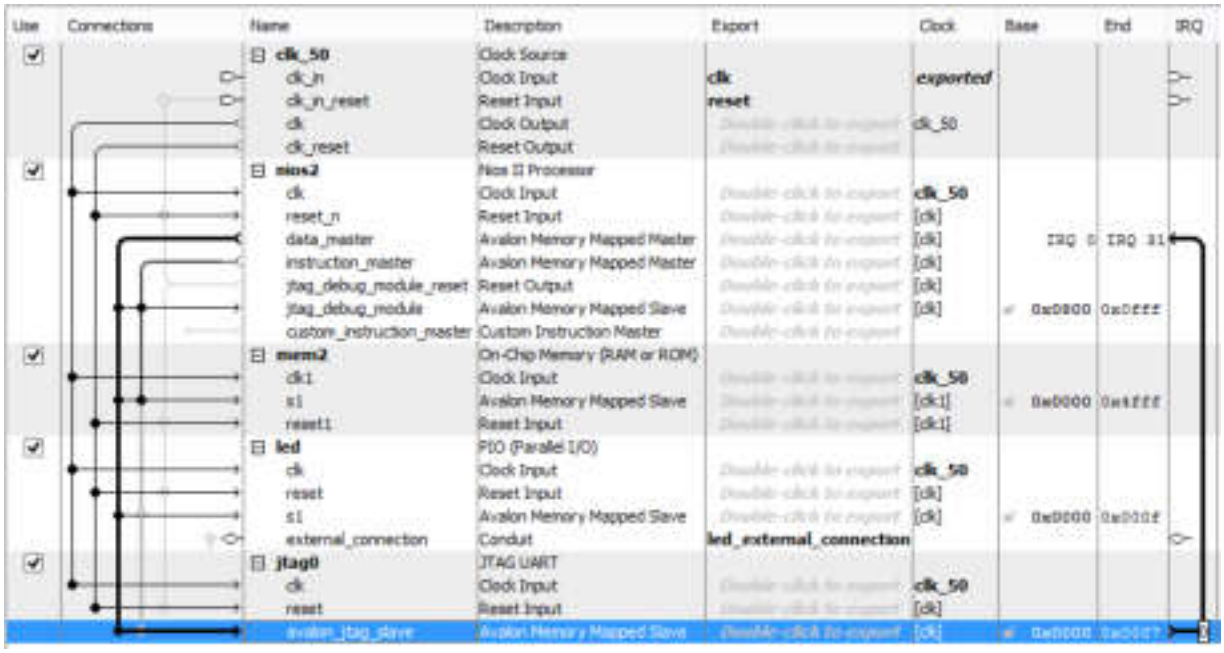


Рис. 13.6. Схема соединений портов компонентов

5. Заходим в настройки Nios II. Для этого дважды кликаем по добавленному элементу и в окнах векторов сброса и исключения устанавливаем память RAM, как показано на рисунке 13.7.

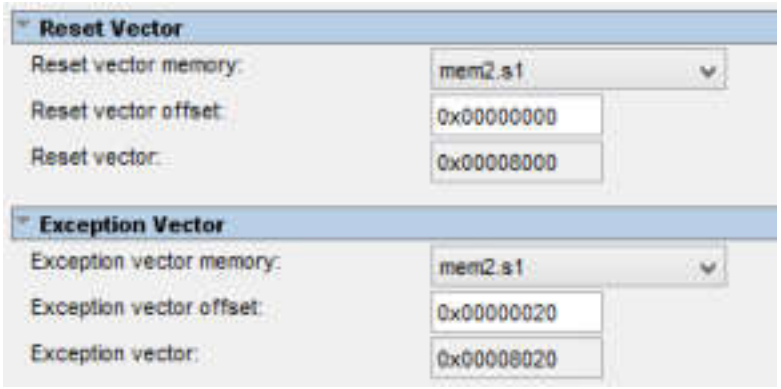


Рис. 13.7. Установка векторов сброса и исключения

6. Для назначения адресов устройствам выполним из меню System → Assign Base Addresses. Будут назначены уникальные адреса для устройств. С помощью этих адресов к устройствам можно получить доступ из программного кода на C/C++. Если при разработке системы в программе Qsys сделано все правильно, то внизу ее окна не должно быть ошибок и предупреждений, как показано на рисунке 13.8.

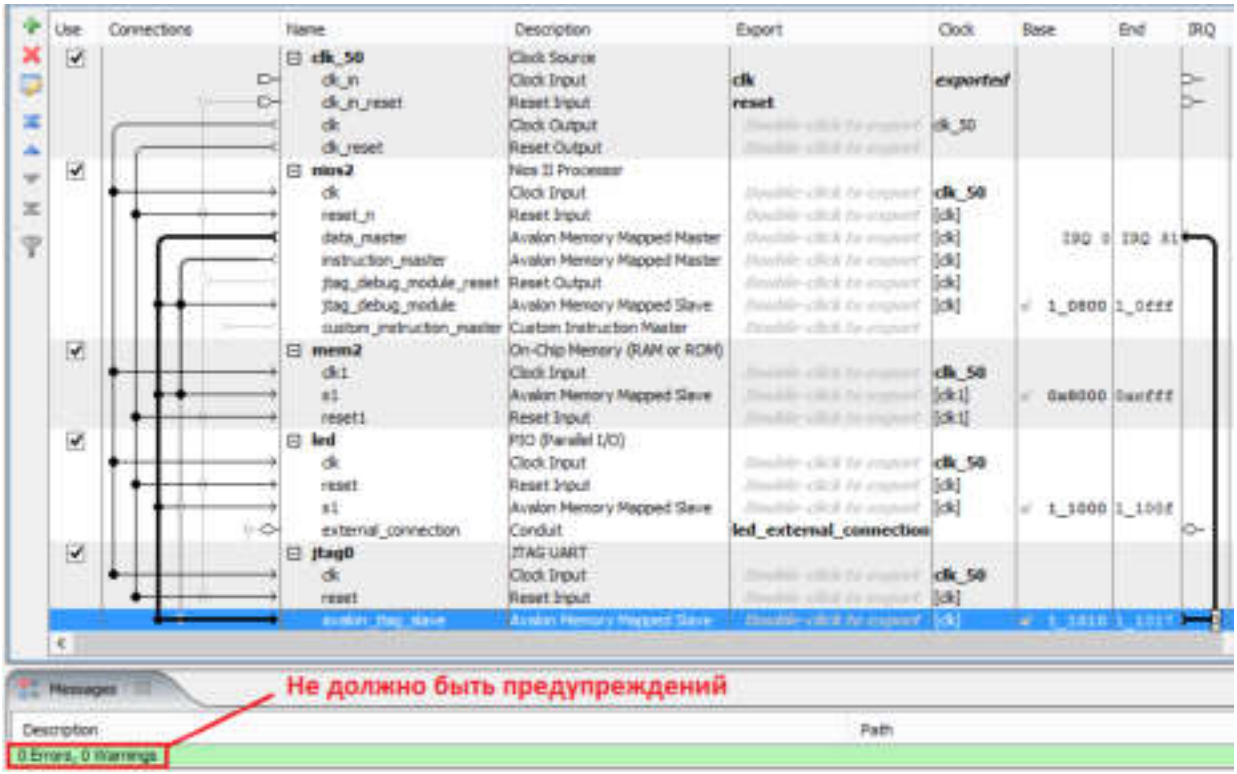


Рис. 13.8. Отсутствие ошибок и предупреждений

7. Чтобы осуществить симуляцию и синтез проекта, нужно перейти на вкладку «Generation» и указать программе Qsys параметры симуляции и синтеза, которые показаны на рисунке 13.9.

После этого необходимо сохранить систему под именем nios2.qsys, а затем нажать кнопку Generate, чтобы запустить генерацию процессора Nios. Если система сгенерирована успешно, то мы получим сообщение, показанное на рисунке 13.10.

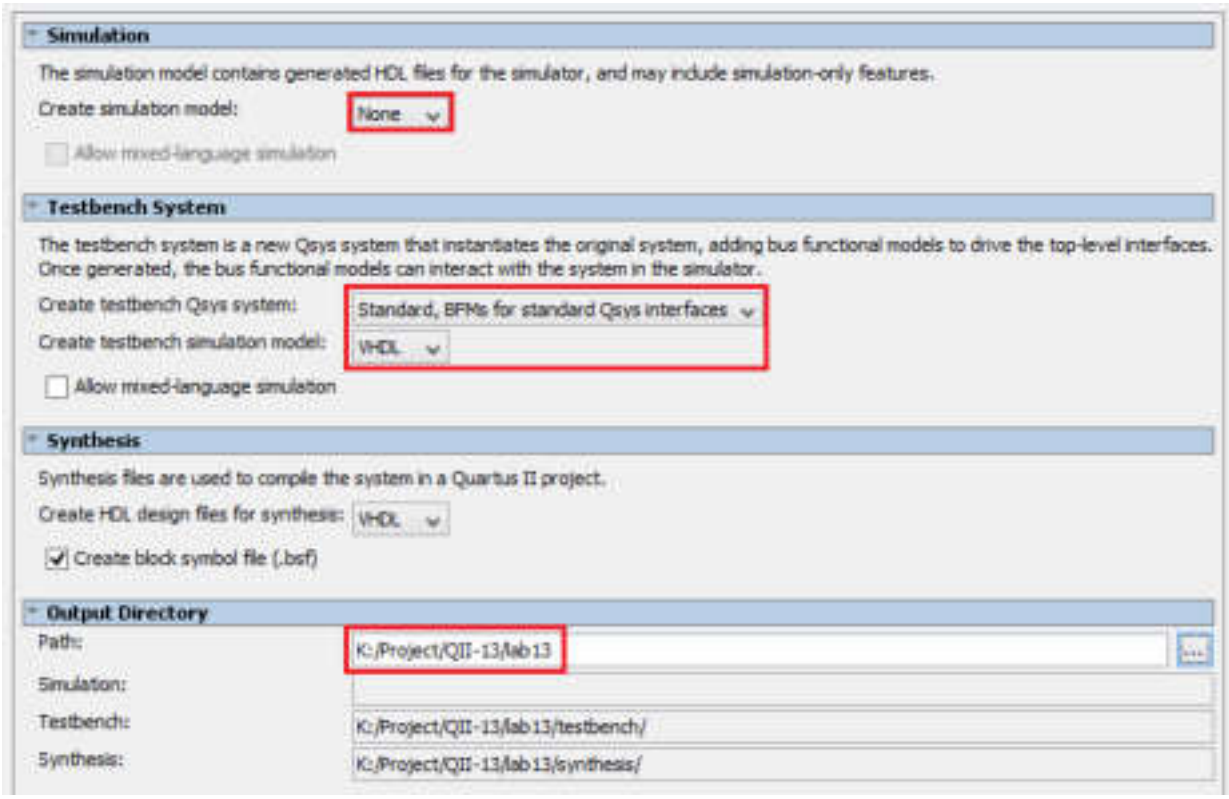


Рис. 13.9. Параметры симуляции и синтеза

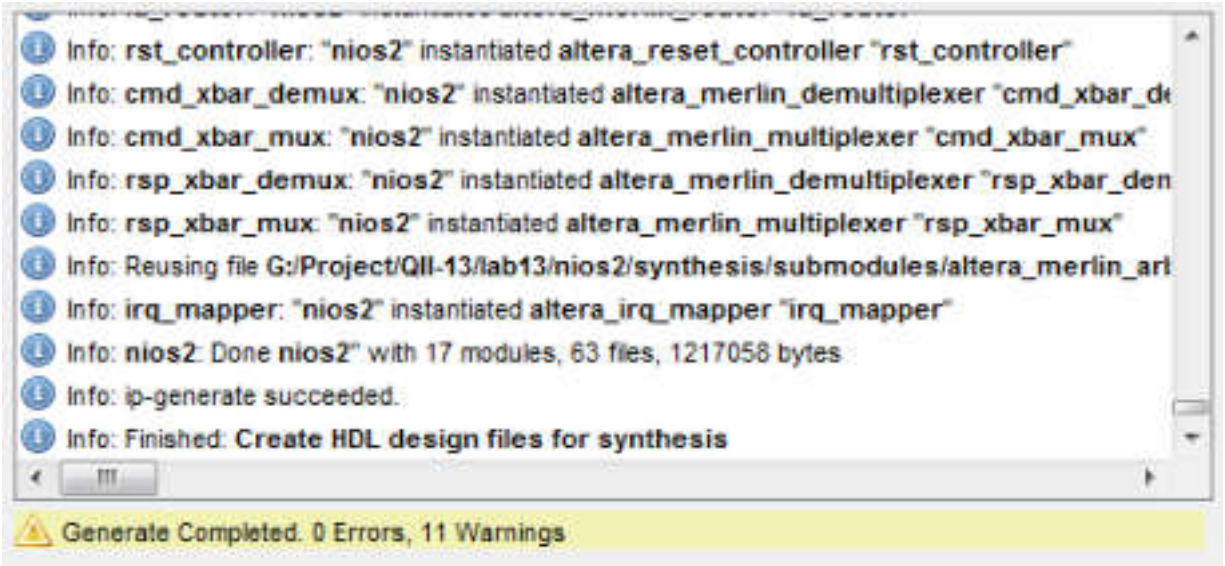


Рис. 13.10. Отсутствие ошибок после генерации системы

По окончании генерации системы создается файл .sopsinfo, который содержит всю информацию о компонентах, входящих в систему, включая их базовые адреса.

8. Далее необходимо написать программное обеспечение для проектируемой системы с Nios. Это делается при помощи специально встроенной в пакет программной среды Eclipse и компилятора.

Выполняем команду Tools → Nios II Software Build Tools for Eclipse. Если Eclipse используется первый раз, то необходимо указать путь для рабочей страницы work space (например, папку проекта).

Для использования файла .sopsinfo нужно создать пакет поддержки платы «board support package (BSP)» и модифицировать его специально для симуляции. Выполняем File → New → Nios II Board Support Package и заполняем все поля, как показано на рисунке 13.11.

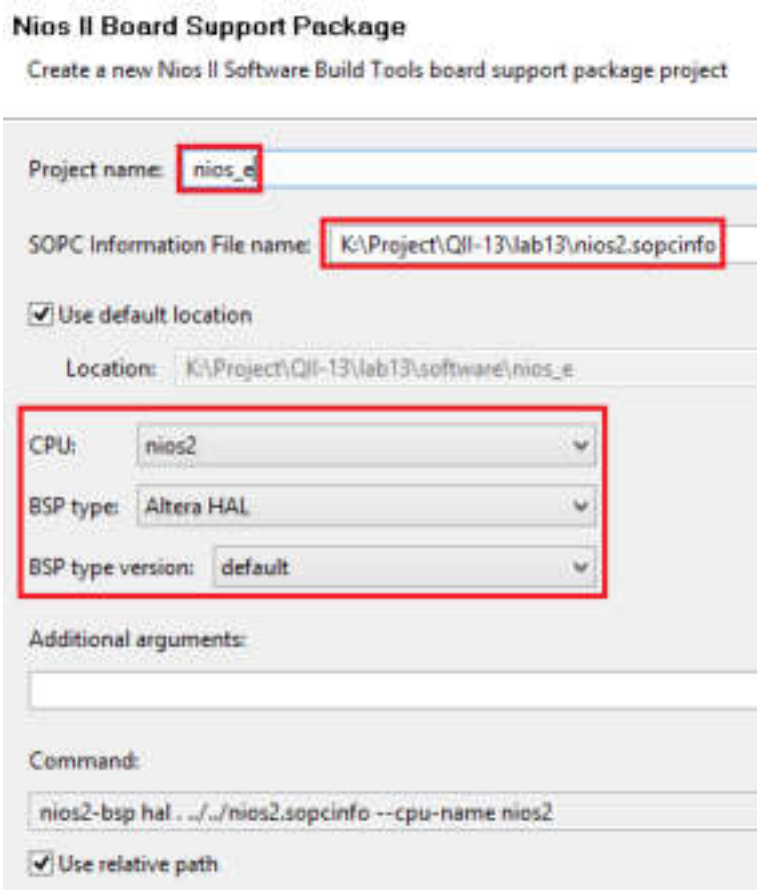


Рис. 13.11. Заполнение полей пакета поддержки платы системы

После нажатия Finish будет создана папка nios_e, в которой находятся файлы system.h, io.h и папка drivers. Эти файлы и папка создаются на основе информации, содержащейся в файле .sopsfile.

9. Чтобы модифицировать BSP, кликните ПКМ по nios_e и выберите пункт Nios II → BSP Editor... В окне редактора нужно установить галочки, как показано на рисунке 13.12, и нажать кнопку Generate.

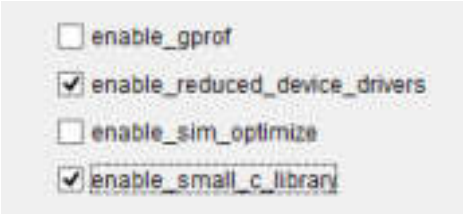


Рис. 13.12. Разрешение использования драйверов и библиотек

Для создания приложения C/C++ вначале выберите пункт File → New → Nios II Application, введите путь к папке BSP и имя проекта, как показано на рисунке 13.13.

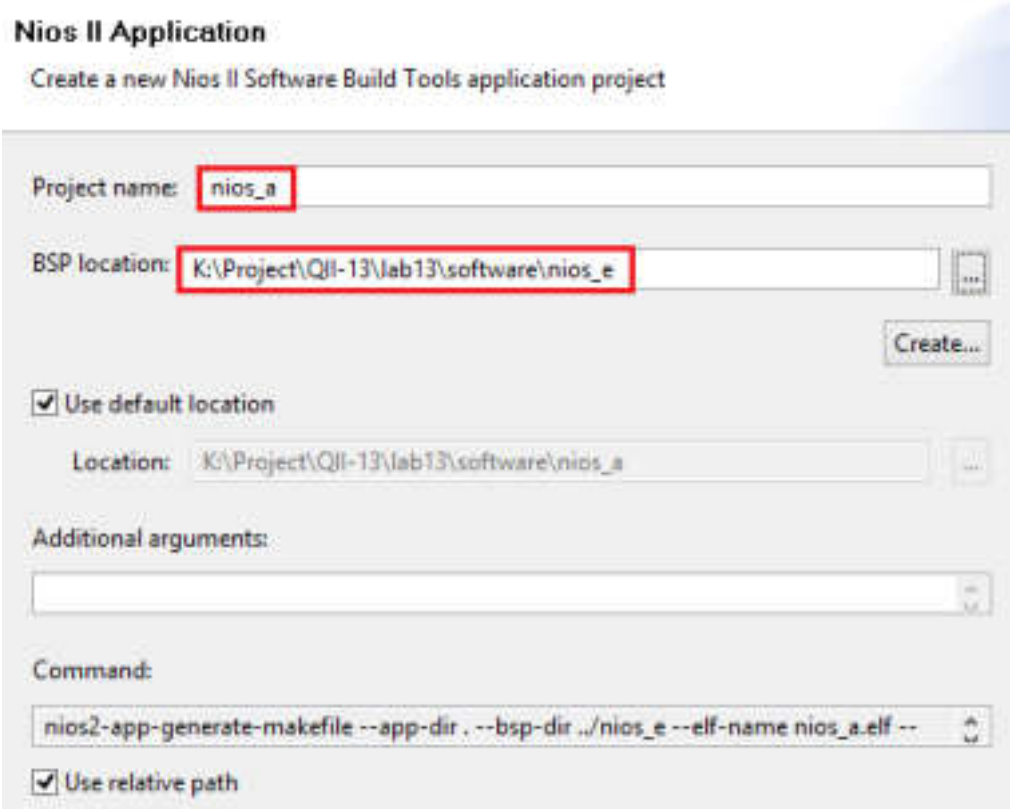


Рис. 13.13. Задание параметров приложения

Чтобы создать код приложения, вначале нужно кликнуть ПКМ по nios_a, выбрав пункт New → Source File, заполнить окна, как показано на рисунке 13.14, и нажать кнопку Finish.

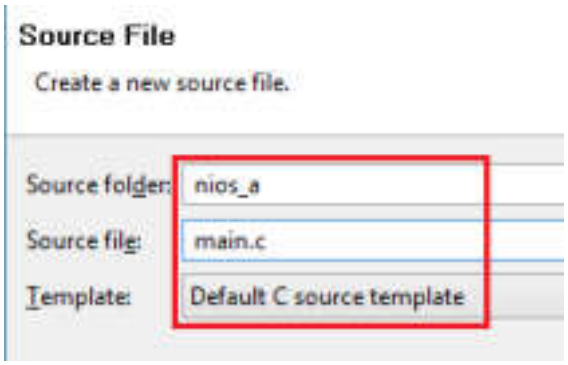


Рис. 13.14. Указание источников и шаблона приложения

Появится шаблон `main.c` для ввода кода. Ниже представлен пример кода для некоторого условного варианта задания.

```
#include "io.h"
#include "alt_types.h"
#include "system.h"

int main() {
     alt_u8 leds = 0x0B;
     printf("Ivanov\n");
     while(1) {
          leds = ~leds;
          IOWR(LED_BASE, 0, leds);
     }
     return 0;
}
```

Модифицируйте код для своего варианта задания, а затем выполните команду Project → Build All, в результате которой будет создан бинарный файл с прошивкой.

Для симуляции проекта нужно кликнуть ПКМ по nios_a, выбрав пункт Run as → Nios II Modelsim. Затем нужно установить параметры конфигурации программы Modelsim: указать место расположения папки «win32aloem» и файла тестбенча для симуляции проекта в программе Modelsim. После этого последовательно нажать на кнопки «Apply» и «Run», как показано на рисунке 13.15.

В результате этих манипуляций запустится процесс подготовки к началу симуляции. Необходимо подождать некоторое время, пока загрузятся все необходимые компоненты для симуляции.

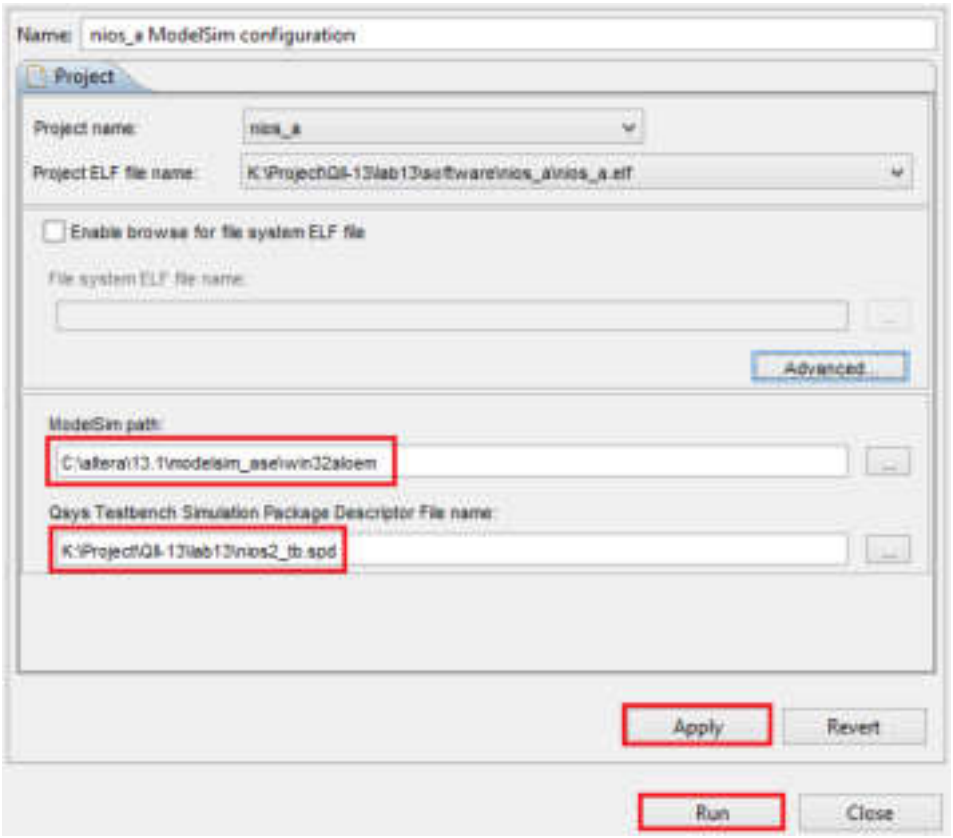


Рис. 13.15. Окно параметров конфигурации программы Modelsim

Затем в программе Modelsim нужно добавить в окно «Wave» сигналы для симуляции, перейти в окно «Transcript» и выполнить в нем команду «run 1 ms». После окончания симуляции должен получиться результат аналогичный представленному на рисунке 13.16.

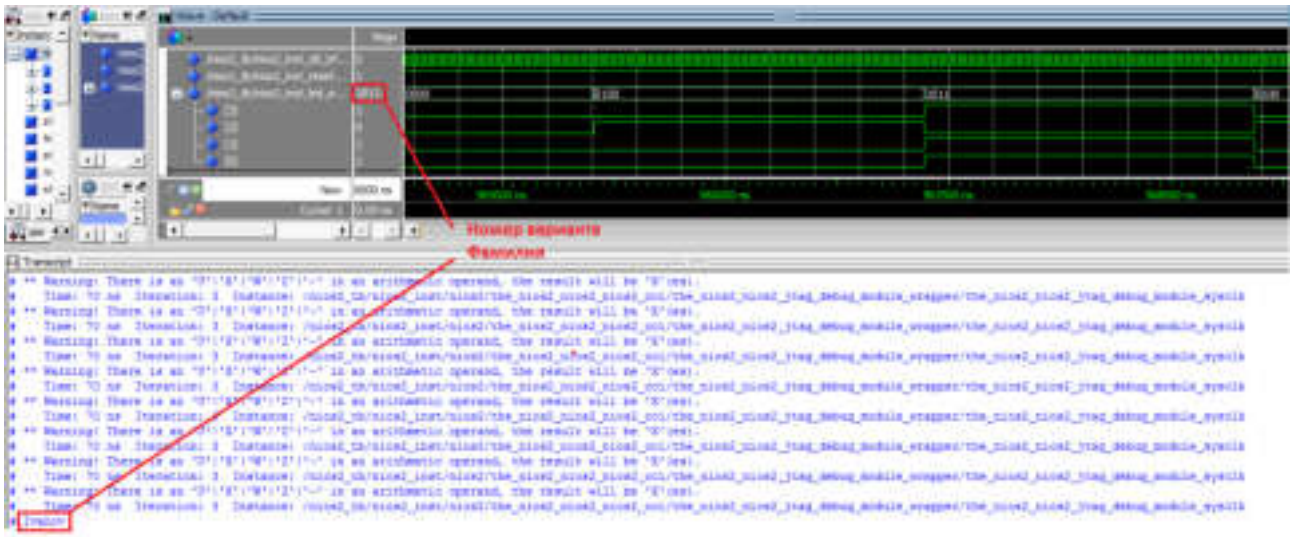


Рис. 13.16. Результат симуляции примера системы в программе Modelsim

Приложение 1

Параметрические элементы САПР Quartus II

Counter

Входные выводы	
Имя вывода	Описание
data [ ]	Параллельный вход данных счетчика
clock	Вход счетных импульсов
clk_en	Разрешение синхронизации
cnt_en	Разрешение счета
updown	Управление направлением счета (1 = сложение, 0 = вычитание)
aclr	Асинхронный сброс входов
aset	Асинхронная установка входов
aload	Асинхронная загрузка входов. Установка счетчика в значение data[ ].
sclr	Синхронный сброс входов. Сброс счетчика следующим тактовым импульсом
sset	Синхронная установка входов. Установка счета следующим тактовым импульсом
sload	Синхронная загрузка входов. Загрузка в счетчик значения data[ ] следующим тактовым импульсом

Выходные выводы	
Имя вывода	Описание
q [ ]	Выход счетчика
eq [15..0]	Декодированный выход счетчика. Высокий активный уровень появляется в момент, когда счетчик достигает заданного значения
cout	Перенос в старший разряд
Параметры	
Параметр	Описание
LPM_WIDTH	Разрядность счетчика или входных значений data[ ] и выходных q[ ]
LPM_DIRECTION	Может принимать значения UP, DOWN или UNUSED. Если этот параметр используется, то вход updown не должен быть подключен. Если вход updown не подключен, то значение LPM_DIRECTION по умолчанию – UP
LPM_MODULS	Максимальный счет, плюс один. Число уникальных состояний в цикле счетчика. Если введенное значение больше, чем LPM_MODULUS параметр, поведение счетчика не определено
LPM_AVALUE	Постоянное значение, которое загружается, когда aset высок. Если введенное значение больше чем <modulus>, поведение счетчика неопределено, где <modulus> - LPM_MODULUS. Параметр ограничен значением в 32 бита
LPM_SVALUE	Постоянное значение, которое загружается по переднему фронту тактовых импульсов, когда sset или sconst высок. Должен использоваться, если используется sconst
LPM_HINT	Позволяет определять специфические Altera-параметры в файлах проекта VHDL
LPM_TYPE	Идентифицирует LPM имя в файлах проекта VHDL

Multiplier

Входные выводы	
Имя вывода	Описание
dataa[]	Множимое
datab[]	Множитель
sum[]	Частичная сумма
clock	Вход тактовых импульсов
clken	Разрешение использования тактового входа
aclr	Асинхронный сброс
Выходные выводы	
Имя вывода	Описание
result[]	result = dataa[] * datab[] + sum. Произведение LSB выровнено с суммой LSB
Параметры	
Параметр	Описание
LPM_WIDTHA	Разрядность dataa[ ]
LPM_WIDTHB	Разрядность datab[ ]
LPM_WIDTHP	Разрядность result[ ]
LPM_WIDTHS	Разрядность sum[ ]. Обязателен, даже если порт суммы не используется
LPM_ REPRESENTATION	Тип выполняемого сравнения SIGNED, UNSIGNED, UNUSED. Если значение не указанно, то по умолчанию устанавливается UNSIGNED
LPM_HINT	Позволяет определять специфические Altera-параметры в файлах проекта VHDL
LPM_TYPE	Идентифицирует LPM имя файла проекта VHDL

INPUT_A_IS_CONSTANT	Altera параметр. Принимает значения YES, NO и UNUSED. Если dataa [] связан с постоянным значением, то INPUT_A_IS_CONSTANT равный YES оптимизирует multiplier по использованию ресурсов и скорости. Если опущено, значение по умолчанию - NO
INPUT_B_IS_CONSTANT	Altera параметр. Принимает значения YES, NO и UNUSED. Если datab [] связан с постоянным значением, то INPUT_B_IS_CONSTANT равный YES оптимизирует multiplier по использованию ресурсов и скорости. Значение по умолчанию - NO
USE_EAB	Altera параметр. Принимает значения "ON", "OFF", и "UNUSED". Устанавливая параметр USE_EAB "ON" позволяет Quartus II использовать блоки дополнительных атрибутов, чтобы использовать 4×4 или (8×значение константы) стандартные блоки в ACEX1K и FLEX10K устройствах
LATENCY	Altera параметр. То же, что и LPM_PIPELINE. Параметр обеспечивает совместимости с MAX+PLUS II проектами версии ниже 7.0. Для всех новых проектов используется параметр LPM_PIPELINE
MAXIMIZE_SPEED	Altera-параметр. Значения от 0 до 10. Если параметр используется, то Quartus II оптимизирует функцию lpm_mult для скорости, а не для экономии ресурсов, и отменяет установку опции Optimize в окне Global Project Logic Synthesis. Если MAXIMIZE_SPEED не использован, значение опции Optimize используется вместо него. Если MAXIMIZE_SPEED = 6 или выше, компилятор оптимизирует мегафункции lpm_mult для высокой скорости; если установлено - 5 или меньше, компилятор оптимизирует для экономии ресурсов

Comparator

Входные выводы	
Имя вывода	Описание
dataa[]	datab[] сравнивается с этим значением
datab[]	Значение, с которым сравнивается dataa[]
clock	Вход тактовых импульсов
clken	Разрешение использования тактового входа
aclr	Асинхронный сброс
Выходные выводы	
Имя вывода	Описание
alb	High (1), если dataa[] < datab[]
aeb	High (1), если dataa[] == datab[]
agb	High (1), если dataa[] > datab[]
ageb	High (1), если dataa[] >= datab[]
aneb	High (1), если dataa[] != datab[]
aleb	High (1), если dataa[] <= datab[]
Параметры	
Параметр	Описание
LPM_WIDTH	Разрядность входов dataa[] и datab[]
LPM_REPRESENTATION	Тип выполняемого сравнения SIGNED, UNSIGNED, UNUSED. Если значение не указанно, то по умолчанию устанавливается UNSIGNED
LPM_PIPELINE	
LPM_HINT	Позволяет определять специфические Altera-параметры в файлах проекта VHDL
LPM_TYPE	Идентифицирует LPM имя файла проекта VHDL
CHAIN_SIZE	
ONE_INPUT_IS_CONSTANT	Специфический Altera-параметр. Принимает значения YES, NO или UNUSED. Обеспечивает большую оптимизацию, если один из входов постоянен. По умолчанию - NO

Adder Subtractor

Входные выводы	
Имя вывода	Описание
dataa[]	Первое слагаемое/ Уменьшаемое
datab[]	Слагаемое/ Вычитаемое
add_sub	Если «1» (high), операция = dataa[]+datab[]+cin. Если «0» (low), операция = dataa[]-datab[] +cin-1
clock	Вход тактовых импульсов
clken	Разрешение использования тактового входа
aclr	Асинхронный сброс
Выходные выводы	
Имя вывода	Описание
result[]	dataa[] +datab[] +cin или dataa[] -datab[] +cin-1
cout	Обнаруживает переполнения в операциях UNSIGNED
overflow	Результат превышает доступную точность

Параметры	
Параметр	Описание
LPM_WIDTH	Разрядность входов dataa[], datab[], result[]
LPM_DIRECTION	Значения - ADD, SUB и UNUSED. Если не указано, значение по умолчанию DEFAULT, в этом случае используется значение add_sub порта. Add_sub порт не может использоваться, если используется LPM_DIRECTION
LPM_REPRESENTATION	Тип выполняемого сравнения SIGNED, UNSIGNED, UNUSED. Если значение не указанно, то по умолчанию устанавливается UNSIGNED
LPM_HINT	Позволяет определять специфические Altera-параметры в файлах проекта VHDL
LPM_TYPE	Идентифицирует LPM имя файлах проекта VHDL

ONE_INPUT_IS_CONSTANT	Altera параметр. Принимает значения YES, NO и UNUSED. Обеспечивает большую оптимизацию, если один вход постоянный. Если не указано, значение по умолчанию - NO
MAXIMIZE_SPEED	Altera-параметр. Возможные значения от 0 до 10. Если параметр используется то MAX+PLUS II пытается оптимизировать данную функцию lpm_mult для скорости, а не для уменьшения занимаемой области, и отменяет установку опции Optimize в диалоговом окне Global Project Logic Synthesis (меню Assign). Если MAXIMIZE_SPEED не использован, значение опции Optimize используется вместо него. Если установлено MAXIMIZE_SPEED – 6 или выше, компилятор оптимизирует мегафункции lpm_mult для более высокой скорости; если установлено - 5 или меньше, компилятор оптимизирует для уменьшения занимаемой области.

Absolute Value

Входные выводы	
Имя вывода	Описание
data []	Число со знаком
Выходные выводы	
Имя вывода	Описание
result[]	Абсолютное значение data [] .
overflow	
Параметры	
Параметр	Описание
LPM_WIDTHA	Разрядность data [] и result[]
LPM_HINT	Позволяет определять специфические Altera-параметры в файлах проекта VHDL.
LPM_TYPE	Идентифицирует LPM имя файлах проекта VHDL

Divider

Входные выводы	
Имя вывода	Имя вывода
numer[]	Числитель
denom[]	Знаменатель
clock	Вход тактовых импульсов
clken	Разрешение использования тактового входа
aclr	Асинхронный сброс
Выходные выводы	
Имя вывода	Описание
quotient[]	Частное
remain[]	Остаток
Параметры	
Параметр	Описание
LPM_WIDTHN	Разрядность numer[] и quotient[]
LPM_WIDTHD	Разрядность denom[] и remain[]
LPM_NREPRESENTATION	Определяет параметр числителя SIGNED или UNSIGNED. Сейчас поддерживается только UNSIGNED
LPM_DREPRESENTATION	Определяет параметр знаменателя SIGNED или UNSIGNED. Сейчас поддерживается только UNSIGNED
LPM_HINT	Позволяет определять специфические Altera-параметры в файлах проекта VHDL
LPM_TYPE	Идентифицирует LPM имя файла проекта VHDL

Приложение 2

Аббревиатура

АЧХ – амплитудно-частотная характеристика
АЦП – аналого-цифровой преобразователь
БПФ – быстрое преобразование Фурье
ВЧ – высокая частота (высокочастотный)
ГОС ВО – государственный образовательный стандарт высшего образования
КИХ – конечная импульсная характеристика
ЛАЧХ – логарифмическая амплитудно-частотная характеристика
ЛДС – линейная дискретная система
НЧ – низкая частота (низкочастотный)
П – полосовой
ПКс – профессиональная компетенция (собственная)
ПЛИС – программируемая логическая интегральная схема
ПФ – полосовой фильтр
Р – режекторный
РФ – режекторный фильтр
ЦАП – цифро-аналоговый преобразователь
ФВЧ – фильтр высокой частоты
ФНЧ – фильтр низкой частоты

Библиографический список

1. Кузяков, О.Н. Проектирование систем на микропроцессорах и микроконтроллерах [Электронный ресурс]: учебное пособие / О.Н. Кузяков. — Электрон. дан. — Тюмень: ТюмГНГУ, 2014. — 104 с. — Режим доступа: https://e.lanbook.com/book/64535. (дата обращения 12.01.2020)
2. Мурсаев, А.Х. Практикум по проектированию на языках VerilogHDL и SystemVerilog [Электронный ресурс]: учебное пособие / А.Х. Мурсаев, О.И. Буренева. — Электрон. дан. — Санкт-Петербург: Лань, 2018. — 120 с. — Режим доступа: https://e.lanbook.com/book/103142 (дата обращения 12.01.2020)
3. Чан, Танг Т. Высокоскоростная цифровая обработка сигналов и проектирование аналоговых систем: пер. с англ. / Танг Т. Чан; под ред. Г. А. Егорочкина. – М.: Техносфера, 2013. – 188 с.
4. Кестер, У. Проектирование систем цифровой и смешанной обработки сигналов / пер. с англ. под ред. А. А. Власенко. – Москва: Техносфера, 2010. – 326 с.
5. Муханин, Л.Г. Схемотехника измерительных устройств [Электронный ресурс]: учебное пособие / Л.Г. Муханин. — Электрон. дан. — Санкт-Петербург: Лань, 2018. — 284 с. — Режим доступа: https://e.lanbook.com/book/98243.
6. Муромцев, Д.Ю. Проектирование функциональных узлов и модулей радиоэлектронных средств [Электронный ресурс]: учебное пособие / Д.Ю. Муромцев, И.В. Тюрин, О.А. Белоусов и др. — Электрон. дан. — Санкт-Петербург: Лань, 2018. — 252 с. — Режим доступа: https://e.lanbook.com/book/109513 (дата обращения 12.01.2020)
7. Строгонов, А.В. Цифровая обработка сигналов в базисе программируемых логических интегральных схем [Электронный ресурс]: учебное пособие / А.В. Строгонов. — Электрон. дан. — Санкт-Петербург: Лань, 2018. — 312 с. — Режим доступа: https://e.lanbook.com/book/104960 (дата обращения 12.01.2020).
8. Mentor Graphics. ModelSim Tutorial. Software Version 10.1d. – Mentor Graphics Corporation, 2012. – 81 p.