ПРАКТИКА ИНЖЕНЕРНОЙ ЭЛЕКТРОНИКИ

Крид Хадлстон

ПРОЕКТИРОВАНИЕ ИНТЕЛЛЕКТУАЛЬНЫХ ДАТЧИКОВ С ПОМОЩЬЮ Microchip dsPIC





WWW.MK-PRESS.COM

МК-ПРЕСС

Intelligent Sensor Design Using the Microchip dsPIC®

Creed Huddleston



ELSEVIER

Elsevier Science
30 Corporate Drive, Suite 400,
Burlington, MA 01803

Крид Хадлстон

Проектирование интеллектуальных датчиков с помощью Microchip dsPIC®

Перевод с английского: В. И. Кириченко, В. В. Литвин

"МК-Пресс"
Киев, 2008

ББК 32.973-04
УДК 004.312
X14

Хадлстон К.

X14 Проектирование интеллектуальных датчиков с помощью Microchip dsPIC – К.: "МК-Пресс", 2008. – 320с., ил.

ISBN 978-966-8806-38-4 (рус.)
ISBN 978-0-7506-7755-4 (англ.)

На страницах этой книги раскрыты способы применения популярных цифровых контроллеров сигналов Microchip dsPIC, в которых вычислительный потенциал мощных цифровых процессоров сигналов удачно объединен с возможностями микроконтроллеров PIC. Рассматриваются вопросы не только программирования, но и проектирования электронного оборудования. Таким образом, читатель получает полное представление о процессе создания интерфейса для трех конкретных типов датчиков: температуры, давления/нагрузки и расхода. Эта практичная, легкая в восприятии книга раскрывает реальные проблемы, возникающие в повседневной работе разработчиков, и показывает решения, позволяющие реализовать все сильные стороны такого мощного средства, как интеллектуальные датчики.

ББК 32.973-04

Главный редактор: Ю. А. Шпак

Подписано в печать 01.11.2007. Формат 60 × 84 1/16. Бумага офсетная. Печать офсетная.
Усл. печ. л. 18,6. Уч.-изд. л. 15,8. Тираж 1500 экз. Заказ № 1059

СПД Савченко Л.А., Украина, г. Киев, тел./факс: (044) 517-73-77; e-mail: info@mk-press.com.
Свидетельство о внесении субъекта издательского дела в Государственный реестр издателей, производителей и распространителей издательской продукции:
серия ДК №51582 от 28.11.2003.

Отпечатано в типографии ЧП Швец С.М. (свидетельство ДК №867 от 22.03.2002).
32300, Хмельницкая обл., г. Каменец-Подольский, ул. Пятницкая, 9а.
Тел.: (03849) 2-72-01, 2-20-79.

Никакая часть настоящего издания ни в каких целях не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами, будь то электронные или механические, включая фотокопирование и запись на магнитный носитель, если на это нет письменного разрешения издательства Elsevier Science.

Authorized translation from the English language edition published by Elsevier Science, Copyright © 2007. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without prior written permission of the publisher.

Russian language edition published by MK-Press according to the Agreement with Elsevier Science, Copyright © 2008.

ISBN 978-966-8806-38-4 (рус.)
ISBN 978-0-7506-7755-4 (англ.)

© "МК-Пресс", 2008
© Elsevier Science, 2007

Эта книга с любовью посвящается моей потрясающей жене Лизе и чудесным детям Кейт, Бет и Дэну. Мы — по-настоящему счастливая семья. Это факт, над которым я часто размышляю.

Книга посвящается также моей сестре Саре — обладательнице неповторимого смеха, которая, наверное, будет удивлена (надеюсь, — приятно) увидеть здесь свое имя.

Краткое оглавление

Глава 1. Что такое интеллектуальные датчики, и зачем они нужны? 21

Глава 2. Интуитивная цифровая обработка сигналов 42

Глава 3. Возможности цифрового контроллера сигналов dsPIC 76

Глава 4. Умение быть хорошим коммуникатором 131

Глава 5. Базовые средства разработки для ЦКС dsPIC 173

Глава 6. Реальный пример: датчик температуры 200

Глава 7. Реальный пример: датчики давления и нагрузки 250

Глава 8. Реальный пример: датчики потока 269

Глава 9. Взгляд в будущее 292

Приложение А. Содержимое прилагаемого к книге компакт-диска 304

Приложение Б. Инициализация ЦКС dsPIC и код запуска системы 306

Приложение В. Управляемый по прерываниям последовательный ввод-вывод 308

Содержание

Введение **14**

Благодарности **17**

Об авторе **20**

Глава 1. Что такое интеллектуальные датчики, и зачем они нужны? 21

1.1. Традиционные датчики несовершенны 24
1.2. Начнем сначала — оцифровка выходного сигнала датчика .. 28
1.3. Следующий шаг — добавим немного интеллекта 29
1.4. И в завершение — быстрый и надежный обмен данными 30
1.5. Совместите все это вместе, и получится интеллектуальный датчик 30
1.6. Почему бы не сделать все датчики интеллектуальными? 33
Затраты на доработку и производство превышают выгоду, оцененную потребителем *33*
Отсутствие необходимой инфраструктуры *34*
Реализации дополнительной электронной схемы препятствуют внешние условия *34*
1.7. Реальные примеры интеллектуальных датчиков 35
Многоканальный цифровой датчик температуры *35*
Датчики потока *37*
Датчик положения в рулевой системе с управлением "по проводу" *37*
1.8. Обзор остальной части книги 39

Глава 2. Интуитивная цифровая обработка сигналов **42**

2.1. Фундаментальные понятия обработки сигналов 42
Конкретизация понятий "сигнал" и "шум" *43*
Наблюдение сигналов в частотной области *46*
Очистка сигнала — введение в фильтры *51*
Дискретизация аналогового сигнала 52
Фильтры нижних частот 57
Фильтры верхних частот 58
Полосовые фильтры 59
Заграждающий фильтр 61

Реализации цифровых фильтров 62
Медианные фильтры 64
2.2. Дискретизация сигналов 66
Влияние оцифровки на дискретный сигнал 66
Влияние ограничений в размере регистра 68
Дискретизация с запасом по частоте 69
2.3. Как анализировать прикладную задачу, в которой применен датчик 70
2.4. Обобщенная структура процесса обработки сигнала от датчика 71
Формирование и обнаружение сигнала 73
Фильтрация перед анализом 73
Линеаризация сигнала 73
Анализ параметра 74
Фильтрация после анализа 74
Обнаружение ошибок и их обработка 74
Обмен данными 74
2.5. Резюме 75

Глава 3. Возможности цифрового контроллера сигналов dsPIC 76

3.1. Архитектура обработки данных в ЦКС dsPIC 79
Память ЦКС dsPIC 79
Распределения памяти в пространстве данных 81
Регистры специального назначения 81
Статическое ОЗУ 82
Память пространства программ, распределенная под память пространства данных 82
Распределение памяти пространства программ 83
Доступ к конфигурационной памяти из пространства пользовательской памяти 85
Отображение памяти программ в пространстве данных 85
Цифровой процессор сигналов 86
Представление числовых данных 86
Аппаратный умножитель 90
Сдвоенный 40-разрядный накапливающий сумматор 91
40-разрядная схема циклического сдвига 94
Режимы адресации и модуль формирования адреса 94
3.2. Структура прерываний 95
Теневые регистры 97
3.3. Встроенные периферийные устройства 98

Периферийные модули сбора данных 98
Аналого-цифровые преобразователи 99
Шаг 1. Конфигурация тракта сигнала 103
Шаг 1а. Конфигурация выводов порта ввода-вывода 104
Шаг 1б. Выбор источников опорного напряжения 106
Шаг 1в. Выбор аналоговых входов для оцифровки 108
Шаг 1г. Определение тактовой частоты АЦП 112
Шаг 1д. Выбор инициаторов дискретизации и преобразования 115
Контрольный перечень этапов для использования модуля АЦП 118
Модуль таймеров/счетчиков *118*
Общие характеристики таймеров/счетчиков 119
Таймер/счетчик типа А 121
Инициализация режима таймера 123
Инициализация режима синхронного счетчика 125
Режим асинхронного счетчика 125
Таймеры/счетчики типа В 126
Инициализация 16-разрядного таймера и 16-разрядного синхронного счетчика 128
Инициализация 32-разрядного таймера или синхронного счетчика 128
Таймеры/счетчики типа С 128
3.4. РЕЗЮМЕ 129

Глава 4. Умение быть хорошим коммуникатором 131

4.1. ВИДЫ СВЯЗИ 131
Определение характеристик канала связи *132*
Пропускная способность канала передачи данных *133*
Сравнение двухточечных и многоузловых сетей *134*
Физические свойства канала передачи данных *136*
Асинхронная и синхронная передача данных *136*
Аппаратное обнаружение ошибок *139*
4.2. КОММУНИКАЦИОННЫЕ ИНТЕРФЕЙСЫ, ДОСТУПНЫЕ В DSPIC30F 139
Порт интерфейса SPI *139*
Приемопередатчик UART *141*
Базовая структура работы интерфейса UART 144
Интерфейс CAN *153*
Введение в архитектуру CAN 153
Форматы данных CAN 156
Арбитраж шины 159
Фильтры принятия сообщений 161
Базовая структура взаимодействия с интерфейсом CAN 162

4.3. Протоколы высокого уровня 162
Общий протокол обработки сообщений *164*
Протоколы со специальными командами *166*
4.4. Резюме 172

Глава 5. Базовые средства разработки для ЦКС dsPIC 173

5.1. Среда тестирования приложений 173
5.2. Обзор микропрограммной структуры 174
Поток данных через приложение *178*
Системный поток задач *180*
Инициализация программной среды 183
Инициализация системных аппаратных средств и программных конечных автоматов 186
Конфигурирование прерываний dsPIC 188
5.3. Реализация структурных модулей 189
5.4. Резюме 199

Глава 6. Реальный пример: датчик температуры 200

6.1. Типы датчиков температуры 201
Термопары *202*
Термопреобразователи сопротивления *203*
Терморезисторы *205*
Кремниевые датчики *205*
Инфракрасные датчики *206*
6.2. Ключевые аспекты измерения температуры 207
Диапазон измерений *208*
Разрешающая способность измерений *209*
Точность измерений *214*
Специфика *215*
Характеристики сигнала 215
Уровни сигналов 215
Частотный спектр сигнала 216
Компенсация холодного спая 217
Линеаризация 218
Калибровка 219
Источники электрического шума 222
Питание от сети переменного тока 222
Сбойные ситуации 225
Разомкнутая термопара 225
Реверсированная термопара 226

6.3. Разработка приложения ... 226
Спецификация системы ... *227*
Формирование сигналов от датчика ... *228*
Дифференциальный усилитель ... 229
Фильтр защиты от наложения спектров ... 230
Анализ цифровой фильтрации ... 231
Алгоритмы анализа данных ... 233
Протокол обмена данными ... 233
6.4. Аппаратная реализация ... 240
Аналоговый усилитель и фильтр защиты от наложения спектров 240
Компенсация холодного спая ... *241*
Изоляция сигнала ... *242*
6.5. Микропрограммная реализация ... 244
Дискретизация сигнала ... *244*
Реализация цифрового фильтра ... *245*
Реализация анализа данных ... *247*
Реализация обработки ошибок ... *247*
Реализация протокола обмена данными ... *248*
6.6. Резюме ... 249

Глава 7. Реальный пример: датчики давления и нагрузки ... 250

7.1. Типы датчиков нагрузки и давления ... 250
Тензометры ... *251*
Пьезоэлектрические датчики ... *252*
7.2. Ключевые аспекты измерения нагрузки ... 253
Диапазон измерений ... *253*
Разрешающая способность измерений ... *254*
Точность измерений ... *254*
Специфика ... *254*
Характеристики сигналов ... 255
Термокомпенсация ... 255
Линеаризация ... 256
Калибровка ... 256
Источники электрического шума ... 257
Сбойные ситуации ... 257
7.3. Разработка приложения ... 258
Спецификация системы ... *258*
Формирование сигналов от датчика ... *258*
Анализ цифровой фильтрации ... *259*
Алгоритмы анализа данных ... *259*

7.4. Микропрограммная реализация 260
Дискретизация сигнала *261*
Реализация цифрового фильтра *266*
Реализация анализа данных *266*
Реализация обработки ошибок *268*
7.5. Резюме 268

Глава 8. Реальный пример: датчики потока 269

8.1. Типы датчиков потока 270
Турбинные датчики *270*
Гравиметрические датчики *272*
8.2. Основные аспекты измерений потока 272
Диапазон измерений *272*
Разрешающая способность измерений *273*
Точность измерений *274*
Специфика измерений потока *274*
Характеристики сигналов 275
Компенсация изменений в плотности материала 275
Линеаризация 275
Калибровка 276
Причины погрешностей 276
Сбойные ситуации 276
8.3. Разработка приложения 277
Спецификация системы *277*
Формирование сигналов от датчика *278*
Анализ цифровой фильтрации *278*
Алгоритмы анализа данных *280*
Протокол обмена данными *283*
8.4. Аппаратная реализация 283
Интерфейсная схема турбинного датчика *284*
8.5. Микропрограммная реализация 286
Модуль сбора данных *286*
Отслеживание уровня сигнала датчика 286
Счетчик проходов лопастей через контрольную точку 287
Модуль фильтрации данных *288*
Фильтрация уровня сигнала от датчика 288
Модуль анализа данных *288*
Анализ скорости потока 289
Анализ уровня сигнала 290
Модуль реализации протокола обмена данными *290*

8.6. Резюме ... 291

Глава 9. Взгляд в будущее ... 292

9.1. Технологические тенденции ... 292
Тенденции развития чувствительных элементов ... 293
Тенденции развития вычислительных элементов ... 294
Значительное снижение энергопотребления ... 294
Непрерывное уменьшение размеров ... 296
Тенденции в сфере коммуникаций ... 296
Повсеместное распространение сети Internet ... 297
Беспроводная связь ... 298
Безопасность ... 299
Специализированные сети ... 299
9.2. Экономические тенденции ... 300
Демография стареющего населения ... 300
Повышение степени глобализации производственных процессов ... 301
Новые бизнес-возможности ... 301
9.3. Резюме ... 302

Приложение А. Содержимое прилагаемого к книге компакт-диска. 304

А.1. Web-сайт ресурсов ... 304
А.2. Исходные тексты для трех приложений ... 304

Приложение Б. Инициализация ЦКС dsPIC и код запуска системы 306

Приложение В. Управляемый по прерываниям последовательный ввод-вывод ... 308

В.1. Псевдокод для структуры ... 310
В.2. Инициализация системы ... 310
В.3. Считывание данных из интерфейса ... 311
В.4. Запись данных в интерфейс ... 311

Введение

Как не существует ни одной книги, в которой было бы рассказано обо всем, так невозможно и создать книгу, которая угодила бы каждому читателю. Как бы я ни хотел, чтобы это было по-другому, но "Проектирование интеллектуальных датчиков с помощью Microchip dsPIC"— не исключение. Эта книга рассчитана, в основном, на три группы читателей:

- опытные разработчики встроенных систем, которые хотят относительно быстро освоить цифровые контроллеры сигналов (ЦКС) (DCS — Digital Signal Controller) dsPIC от компании Microchip, но не имеют времени на пересмотр огромного количества детализированной документации;
- студенты инженерных специальностей, которые только начинают изучать встроенные системы, и нуждаются в дополнительном руководстве по согласованию этапов разработки;
- все, кто разбирается в программировании и инженерии и хочет узнать побольше об удивительном мире измерений с помощью интеллектуальных датчиков.

Для того чтобы извлечь из предоставленного здесь материала максимум полезного, читатель должен быть знаком с основами программирования на языке С. Во всех рассмотренных примерах используются простые конструкции этого языка, поскольку основное внимание в книге уделено все-таки разработке измерительных систем, а не программированию как таковому. Если читатель не владеет основами языка С, то рекомендуем предварительно прочитать какую-нибудь хорошую книгу по этой тематике.

Эта книга также не является учебником для начинающих в области цифровой обработки сигналов (ЦОС) (DSP — Digital Signal Processing). Хотя в ней и дан краткий обзор (одна глава) основных принципов ЦОС, строгое научное изложение темы отсутствует. Конечно, данный вопрос очень важен, и автор сам прочитал массу книг, посвященных ЦОС. Однако в нашем случае ЦОС — это всего лишь инструмент, и мы исходим из предположения, что читатель уже имеет определенные познания в данной сфере. Поскольку цель этой книги — развить интуитивное по-

нимание принципов ЦОС, чрезмерная детализация предмета была бы просто напрасной тратой времени. Существуют другие специалисты, гораздо более квалифицированные, чем автор, которые пишут такого рода книги. Соответствующую информацию ищите на прилагаемом к книге компакт-диске в папке Ресурсы.

Наконец, эта книга не является руководством по разработке аппаратного обеспечения, хотя в примерах и обсуждаются некоторые схемы. В действительности, фокус направлен больше на создание базовых платформ формирования сигналов с последующим использованием возможностей программного обеспечения ЦКС dsPIC для извлечения полезной информации из контролируемой системы. Как и в случае с программированием и ЦОС, существует множество превосходных учебников по разработке аналоговых и цифровых схем, написанных мастерами в этой области.

Итак, мы объяснили, чего в этой книге *нет*, и теперь можно взглянуть на то, что в ней *есть*. Цель изложенного здесь материала — помочь читателю быстро освоить программную и аппаратную часть платформы разработки и обрести знания, позволяющие легко экспериментировать с различными подходами к проектированию интеллектуальных датчиков. Этот новый класс датчиков быстро заменяет традиционные, предлагая улучшенные возможности измерений, возможность легко создать интерфейс с самыми разнообразными системами контроля и управления, и другие возможности, отсутствующие в обычных датчиках.

С учетом этого, в книге рассмотрены три отдельные сферы применения интеллектуальных датчиков:

- измерение температуры;
- контроль за изменением нагрузки;
- контроль за потоком жидкости.



На прилагаемом к книге компакт-диске в папке Коды находится хорошо комментированный исходный код программ для всех трех сфер применения.

В книге используются различные компоненты аппаратной и программной разработки от компании Microchip и, на первый взгляд, может показаться, что это — срытая стратегия маркетинга Microchip, однако это совсем не так. Со стороны Microchip не было никакой финансовой поддержки для этого проекта, а компания автора Omnisys Corporation является авторизированным консультантом не только Microchip, но и ряда других производителей полупроводников, включая Cypress, Freescale и Lattice.

Первопричиной для использования продукции Microchip послужило то, что она легкодоступная и недорогая (зачастую даже бесплатная).

Кроме того, существуют форумы пользователей, где можно получить помощь в случае возникновения каких-либо вопросов или трудностей. Например, компилятор языка С (версия для студентов) и интегрированную среду разработки MPLAB (IDE) можно бесплатно загрузить с Web-сайта Microchip, как и программу Filter Lab 2.0 для разработки фильтров защиты от наложения спектров (о них мы поговорим чуть позже). Кроме того, Microchip предлагает бесплатное средство анализа ЦОС dsPICWorks и программное обеспечение dsPIC Filter Design за очень приемлемую цену.

Большинство известных мне разработчиков не хотят тратить много денег только для того, чтобы проверить на практике несколько идей, и компания Microchip делает большую работу по обеспечению разработчиков недорогими и бесплатными инструментами. Автор сам Пользуется этими инструментами в своей повседневной работе, поэтому знаком с ними не понаслышке.

Надеемся, эта книга будет вам интересна и полезна. Автор имеет 20-летний опыт разработки встроенных систем (в частности, систем контроля реального времени и систем связи), и на протяжении этого времени ему посчастливилось работать бок о бок с очень талантливыми и проницательными специалистами в различных прикладных сферах — в том числе и очень сложных. Некоторые из их наработок задействованы и в этой книге.

Благодарности

Этой книге положила начало моя статья "Digital Signal Processing Turns Thermocouples Into Superstars" ("Цифровая обработка сигналов превращает термопары в суперзвезд"), опубликованная в журнале "Sensors Magazine" в январе 2003 года. Эта статья попалась на глаза Кэрол Льюис (Carol Lewis), которая в то время была одним из редакторов Butterworth-Heinemann/Newnes — подразделения Elsevier Press. Именно она предложила мне написать отдельную книгу об интеллектуальных датчиках на базе dsPIC. Хотя Кэрол ушла на пенсию еще до того, как была завершена рукопись, благодаря одному из приятных поворотов судьбы, именно ей представилась возможность отредактировать окончательный черновик. Я чрезвычайно благодарен Кэрол за то, что она с самого начала верила в мои писательские способности и постоянно помогала мне в формировании окончательного варианта книги.

Мне также повезло с моим новым редактором Тиффани Гасбаррини (Tiffany Gasbarrini), которая взяла на себя роль наставника, друга, защитника, а порой — и надзирателя. Если бы не ее мастерство в каждой из четырех ролей, эта книга никогда бы не увидела свет, что могут подтвердить как она, так и моя жена. Тиффани квалифицировано помогала Мишель Кронин (Michele Cronin), которая примкнула к команде за полгода до завершения рукописи. Благодаря внимательности Мишель к мелочам и ее личной обаятельности, книга получилась технически качественна, а мне было приятно над ней работать.

На протяжении многих лет мне посчастливилось работать со многими выдающимися специалистами, которые помогали мне сформировать практическое понимание инженерного проектирования — особенно в сфере систем реального времени. Четверых из них хотелось бы выделить особенно, поскольку в этой книге задействованы некоторые из их наработок. Дейл Дюваль (Dale DuVall), Джон Бейтсон (John Bateson) и Дейл Редфорд (Dale Redford), с которыми я впервые познакомился в конце 1980-х годов в компании Scantek Corporation, научили меня ценности досконального понимания физических процессов, лежащих в основе любой электронной системы. Каждый из этих троих инженеров был превосходным специалистом в конкретной области и обладал собственным взглядом на процесс разработки.

Дейл Дюваль — физик по образованию, и его понимание мельчайших деталей физических процессов — нечто воистину феноменальное. Благодаря его непревзойденному мастерству сопоставлять сходные явления из разных областей знаний, я научился наблюдательности и внимательному отношению к аномалиям.

Джон, на мой взгляд, — самый лучший из известных мне разработчиков аналоговой техники, а также один из самых умных и приятных в общении людей. В моих глазах его внимательность к мелочам и умение находить оптимальный путь решения задачи — настоящий стандарт.

Наконец, Дейл Редфорд — блестящий разработчик в сфере цифровой техники, который в том, что касается проектирования схем и систем, всегда соглашается только на самый лучший результат.

Эти трое — совершенно уникальные ребята, и период работы с ними в Scantek связан для меня только с самыми приятными воспоминаниями.

Четвертый инженер, о котором мне хотелось бы упомянуть, — это мой близкий друг и партнер по бизнесу в Omnisys Corporation на протяжении последних одиннадцать лет (и кроме того, мы болеем за одну и ту же хоккейную команду). Его зовут Фред Франц (Fred Frantz). Нам посчастливилось вместе разрабатывать системы управления и связи, которые используются в изготовлении самых разнообразных изделий: от мониторов сердцебиения новорожденного до амортизаторов автомобилей и электронных систем управления высококлассными яхтами. За эти годы я многому научился от Фреда и всегда получаю удовольствие от общения с ним.

Само собой разумеется, все разработки в Omnisys — заслуга далеко не двух человек, и я хотел бы поблагодарить Пэрри Эдмайр (Parry Admire), Джона Брашиера (John Brashier), Мери Франц (Mary Frantz) (да, вы не ошиблись, — это жена Фреда, и притом — превосходный инженер), и особенно — Стива Гибсона (Steve Gibson) за их поддержку и готовность поделиться своим немалым опытом.

Впрочем, самую большую благодарность в завершение я хотел бы выразить тем, кто значит для меня больше всего: моей семье. Только став отцом, я понял, сколько времени и любви отдали мне в детстве мама и папа, и через что пришлось пройти моим сестрам Сьюзен и Саре, которые помогали меня растить. Когда Сьюзен вышла замуж за Джеймса Белота (James Belote) (и у меня, наконец-то, появился брат), он стал душой всех наших семейных встреч. То же самое могу сказать и о его детях.

Когда я женился на Лизе, моя семья увеличилась за счет прекрасной тещи и чудесного тестя: Джони и Герхарда Шульц, — которые воистину стали для меня вторыми родителями. Решение жениться на Лизе, несомненно, стало самым умным в моей жизни, и я каждый день радуюсь тому, что у меня есть такая спутница жизни. Что же касается наших троих детей: Кэти, Бет и Дэна, — то они — настоящий свет нашей жизни. В общем, думаю, вас не нужно много убеждать в том, что я — счастливый человек.

Крид Хадлстон

Об авторе

Имея за плечами более, чем двадцатилетний опыт в сфере разработки встроенных систем реального времени, Крид Надлстон является вице-президентом корпорации Omnisys — молодой компании из города Роли, штат Северная Каролина, которая специализируется на создании систем измерения, управления и связи, работающих с жесткой привязкой к реальному времени. Будучи одним из основателей компании, Крид отвечает за проектирование новых изделий и развитие взаимодействия Omnisys, как авторизированного консультанта, с такими компаниями, как Microchip Technologies, Freescale Semiconductor, Lattice Semiconductor и TrollTech.

В дополнение к своим обязанностям в Omnisys, Крид также входит в состав консультационного совета техасской компании Quickfilter Technologies Inc., которая занимается выпуском цифро-аналоговых приборов, реализующих в одном корпусе формирование высокочастотного аналогового сигнала и ЦОС.

До того, как Крид переехал на восток в Роли, чтобы основать корпорацию Omnisys, он по окончании Райсовского университета в Хюстоне, штат Техас, со степенью бакалавра в области инженерии занимался обширными научными исследованиями в сфере цифровой обработке сигналов в университете штата Техас в Арлингтоне.

Крид — настоящий семьянин. Он уже 23 года неразлучен со своей женой Лизой, и у них трое прекрасных детей: Кэти, Бет и Дэн.

Технические интересы Крида сосредоточены на разработке и применении интеллектуальных измерительных систем — особенно в сфере точных измерений, сенсорных сетей и высоконадежной связи в маломощных системах с длительным циклом эксплуатации.

Связаться с Кридом Хадлстоном можно по адресу электронной почты `creedh@intelligentsensordessign.com`.

1

Что такое интеллектуальные датчики, и зачем они нужны?

Мы взяли то, что было роскошью, и превратили это в необходимость.

— Генри Форд

В современном мире быстрого доступа к чему угодно люди хотят получать информацию в любой момент, в удобной форме и по приемлемой цене (а лучше всего — бесплатно). Как отмечает Питер Друкер (Peter Drucker) — величайший из умов последнего столетия в сфере управления — в отличие от физических продуктов, информация не поддается законам экономической теории, согласно которым, чем меньше товара, тем он ценнее. Наоборот, информация становится тем более полезной (и ценной), чем ее больше и чем шире она распространена[1].

Отдельные люди и организации, понявшие этот принцип, начали извлекать огромную выгоду из "мертвых" информационных залежей в коммерческих, образовательных и общественных организациях по всему миру, переводя массу "сырых" данных в цифровую форму, анализируя их для преобразования в смысловую информацию, а затем — рассылая ее по стандартным каналам связи в своей организации или за ее пределы для выполнения полезной работы.

Сложно преувеличить то влияние, которое оказала эта новая парадигма на общество на экономическом, образовательном и бытовом жизни. Люди теперь говорят о работе *в режиме Internet* — понятие, в котором смешалось одновременно сокращение пространства и времени. Информация из любой точки земного шара может быть доставлена практически куда-угодно и притом — быстро и достоверно, так что Индия теперь настолько же близка к Нью-Йорку, как и Бостон.

В этом новом мире люди работают не так, как раньше. Отдельные специалисты или их группы могут легко образовывать команды с другими участниками проектов независимо от их географического положе-

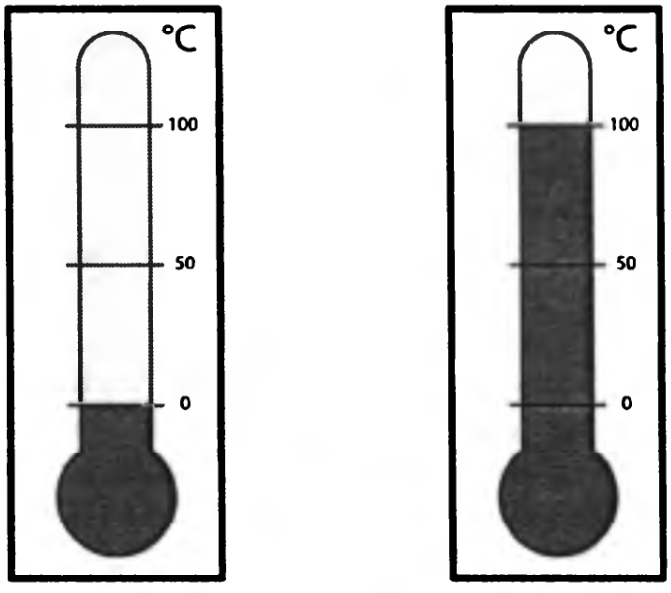
---

[1] Management Challenges for the 21st Century, by Peter F. Drucker. Collins, 2001

ния. Это ускоряет формирование новых идей и производство новых продуктов или услуг, что позволяет сколотить неслыханное состояние и зачастую — за счет крушения других. Воистину, новая парадигма, которую писатель Томас Фридман (Thomas Friedman) называет "сплющиванием земного шара", олицетворяет тектонический сдвиг в том, как люди воспринимают свой мир и живут в нем.

Примечательно, что почти идентичные, хотя и почти никем незамеченные изменения, происходят и в мире датчиков. Для непосвященных можно сказать, что *датчик* (или *измерительный элемент*, как его иногда называют) — это устройство, которое позволяет измерить величину некоторого физического состояния, используя характерные физические свойства датчика. Звучит замысловато, но, на самом деле, все довольно просто: наблюдая за некоторым (относительно) простым для исследования параметром, мы можем судить о значении другого, трудно поддающегося наблюдению, параметра.

В качестве всем понятного примера неэлектронного датчика температуры можно привести обычный ртутный термометр, в котором ртуть в колбе-столбике сокращается или расширяется в зависимости от температуры материала, с которым он соприкасается. В этом случае замеряемым физическим состоянием является температура материала, в который помещен термометр, а характерным физическим свойством датчика — высота ртутного столба (рис. 1.1).



Вода в точке замерзания (0°C=32°F)

Вода в точке кипения (100°C=212°F)

Рис. 1.1. *Два ртутных термометра, показывающих температуру материала (льда и кипящей воды) в двух различных состояниях*

Итак, какие же параметры можно измерить с помощью датчиков? Практически какие угодно — диапазон, по сути, ограничен только нашим воображением. Вероятно, наиболее часто измеряемый параметр — это температура, хотя можно назвать и давление, ускорение, влажность, пространственное положение, уровень pH и многое другое. Впрочем, датчики полезны не только способностью точно измерять разнообразные параметры, но и тем, что могут работать в таких окружающих условиях, в которых присутствие человека просто невозможно. Будь то измерение температуры расплавленной стали в центре домны или наблюдение за океаническим течением на глубине в несколько километров — датчики предоставляют точную информацию, позволяющую нам контролировать самые разнообразные процессы.

На первый взгляд может показаться, что датчики попадают в ту же категорию, что и хлопчатобумажная рубашка: вроде бы и удобно, но не особо впечатляет. Однако такое мнение — крайне ошибочно. В 2005 году население планеты составляло примерно 6,4 млрд. человек[2]. По совпадению, объем рынка промышленных датчиков в одних только Соединенных Штатах на тот момент составил ориентировочно $6,4 млрд.[3], а во всем мире — $40 млрд. Датчиков в мире намного больше чем людей. Они призваны решать задачи от простейших бытовых до передовых научных. И люди готовы платить за ту помощь, которую им оказывают датчики. Это мощное и прибыльное пересечение спроса, технического прогресса и экономических перспектив послужило тому, что в игру вступил новый класс устройств, совершивший настоящую революцию в сфере измерений. Речь идет об *интеллектуальных датчиках*.

Но что же собой представляют эти интеллектуальные датчики? Концептуально, это новый класс электронных измерительных устройств, которые в буквальном смысле слова революционизируют способ сбора данных из окружающего мира, извлечение полезной информации из этих данных и, в конце концов, — использование этой информации для повышения скорости, точности и надежности выполнения операций с меньшими затратами, чем когда-либо ранее. Что еще лучше,

---

[2] Согласно CIA World Factbook, население Земли на июль 2005 года составляло примерно 6 446 131 400 человек (`http://www.cia.gov/cia/publications/factbook/rankorder/2119rank.html`).

[3] По результатам исследовательской программы "Технологии и рынки промышленных датчиков GB-200N", проведенных B.L.Gupta для корпорации Business Communications Company, Inc., в 2004 году объем рынка промышленных датчиков в Соединенных Штатах составил $6,1 млрд. с ожидаемым годовым приростом 4,6% (`http://www.bccresearch.com/instrum/GB200N.html`).

мы можем повысить эффективность отдельных интеллектуальных датчиков, направляя от них информацию другим интеллектуальным датчикам или системам. Это позволяет решать задачи, которые до этого были невыполнимыми, и вносить невероятные улучшения в самые разнообразные сферы применения. Звучит впечатляюще, не правда ли?

1.1. Традиционные датчики несовершенны

Прежде чем перейти непосредственно к обсуждению интеллектуальных датчиков, рассмотрим поближе обычные датчики. Это даст нам прочный фундамент для развития понимания интеллектуальных датчиков.

Невзирая на все свои достоинства, большинство датчиков имеют ряд технических и экономических недостатков. Для эффективной работы датчик необходимо *калибровать*. Это означает, что его выходные данные должны соответствовать некоторому предопределенному стандарту, чтобы полученные значения корректно отображали измеряемый параметр. В примере со ртутным термометром шкала рядом со столбиком ртути должна в точности соответствовать уровню ртути при той или иной температуре. Если датчик не калиброван, то сообщаемая им информация не будет точной, что может стать большой проблемой для использующих ее систем.

Следует также отметить, что не во всех ситуациях требуется одинаковая степень точности. Например, если показание термостата в вашем доме отклоняется от действительного на градус или два, то в действительности это не имеет особого значения — вы просто подстроите температуру, чтобы добиться комфортной среды. Однако та же разница в один или два градуса для химической реакции может привести к трем разным результатам: полезное соединение, бесполезная смесь или взрыв!

Позже мы обсудим вопрос калибровки подробнее. Самое же главное на данный момент — понять, что возможность выполнить точную калибровку датчика — это, зачастую, не просто хорошее, а жизненно необходимое качество. Также, не менее важно помнить, что во многих случаях добраться до датчика для ручной калибровки после того, как он помещен в среду, крайне сложно, если вообще возможно.

Еще одна проблема, характерная для датчиков, заключается в том, что их свойства обычно меняются со временем — явление, известное под названием *дрейф*. Допустим, мы измеряем постоянный ток в некоторой части электрической схемы, наблюдая за напряжением на резисторе в этой схеме (рис. 1.2).

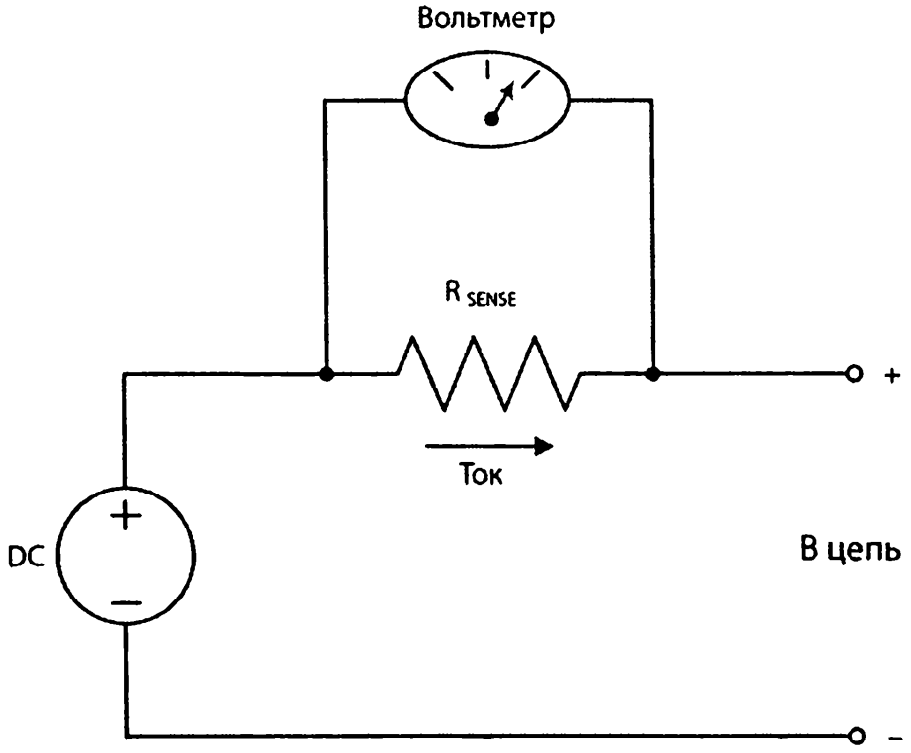


Рис. 1.2. Пример резистивного чувствительного элемента, используемого для измерения тока

В данном случае датчик — это резистор, а измеряемым физическим параметром является напряжение на резисторе, которое, согласно закону Ома[4], прямо пропорционально току, протекающему через этот резистор. Поскольку резистор стареет, его химические свойства меняются, а значит изменяется и его сопротивление. Если, например, в новой системе напряжение на резисторе при силе тока 100 мА составляло 2,7 В, то через пять лет при том же токе это значение может увеличиться до 2,76 В. Хотя разница в 0,06 В может показаться небольшой, в некоторых сферах применения она может играть существенную роль. Возвращаясь к вопросу калибровки, некоторые ситуации требуют более жестких допусков по дрейфу, чем другие. Поскольку свойства датчика меняются со временем, это может привести к нежелательным последствиям, если дрейф каким-то образом не компенсировать.

Кстати, вы заметили, что в предыдущем примере в качестве измерительного элемента определен резистор, а не вольтметр? Как уже было

---

[4] Закон Ома гласит, что V = I * R, где V — напряжение на сопротивлении (в вольтах); I — ток, протекающий через это сопротивление (в амперах); R — величина самого сопротивления (в омах). Закон Ома имеет силу для чисто резистивного элемента — условие, которое в нашем примере соблюдается.

отмечено ранее, в данном случае мы отслеживаем ток в цепи, а измеряем напряжение на резисторе. Таким образом, резистор — первичный датчик, а напряжение на нем — свойство, которое изменяется в соответствии с изменением интересующего нас параметра. Вольтметр — это вторичный измерительный прибор, который мы используем для замера первичного параметра. Не трудно догадаться, что сам вольтметр имеет проблемы с калибровкой и дрейфом. Важность различия между первичным и вторичным датчиками обусловлена тем, что крайне необходимо точно понимать, что именно мы измеряем. Без ясного понимания интересуемого параметра(ов) существует вероятность создания системы, которая в действительности не измеряет то, что требуется, или характеризуется проблемами с точностью. Мы уделим больше внимания этому аспекту чуть позже.

Третья проблема заключается в том, что со временем изменяются не только датчики, но и окружающая их среда. Превосходная иллюстрация этого — электронное зажигание двигателя внутреннего сгорания. Сразу после настройки все ремни натянуты, свечи новые, топливные инжекторы чистые, а воздушный фильтр — "с иголочки". Затем все эти компоненты начинают постепенно деградировать: ремни ослабляются, на свечах зажигания и топливном инжекторе образуется налет, а воздушный фильтр все больше засоряется. Если система электронного зажигания не измеряет степень этих изменений и не выполняет подстройку, то распределение во времени моментов искры постепенно окажется в дисбалансе с параметрами двигателя, что в итоге ухудшит его работу и уменьшит эффективность расхода топлива. Возможно, когда вы вертитесь на машине по городу, где на каждом углу есть заправка, это не так уж и важно, но отношение коренным образом изменяется, если необходимо перелететь на самолете через океан без дозаправки! Способность компенсировать частые изменения в рабочей среде — это очень важное преимущество датчика во многих сферах применения.

Существует еще и четвертая проблема: для того чтобы использовать датчик в той или иной сфере слежения или управления, в большинстве случаев требуется наличие специализированной аппаратной части, называемой *схемой формирования сигнала*. Эта схема отвечает за преобразование физического свойства датчика, за которыми мы наблюдаем (обычно — аналоговое электрическое напряжение, которое систематически изменяется в соответствии с измеряемым параметром) в меру, которая может быть использована остальной частью системы. В зависимости от применения, роль схемы формирования может выполнять обычный усилитель, усиливающий сигнал до приемлемого уровня, или же

сложная схема, фильтрующая сигнал и компенсирующая воздействия окружающей среды. Зачастую сама схема формирования должна быть настроена на конкретное применение датчика и аналоговые сигналы, что обычно предполагает физическую подстройку потенциометра или другого подобного устройства. Кроме того, конфигурация схемы формирования нередко — уникальна по отношению как к конкретному типу датчика, так и к самой области применения. Это означает, что разные типы датчиков или разные области применения обычно нуждаются в настройке схем формирования сигнала.

Наконец, стандартные датчики обычно необходимо устанавливать в физической близости к системам управления и слежения, принимающих данные измерений. В основном, чем дальше находится датчик от системы, тем менее полезны его измерения. Это обусловлено, главным образом, тем, что сигналы от датчика, преодолевая большие расстояния, подвержены электронному шуму, что ухудшает качество считывания на принимающей стороне. Во многих случаях датчики подключаются к системам управления и слежения специализированным (и дорогостоящим) кабелем. Чем длиннее такой кабель, тем дороже обходится установка, что всегда непопулярно среди конечных пользователей. В связи с этим возникает еще одна проблема: становится тяжело распределить выходные сигналы датчика между несколькими системами — особенно если эти системы разделены физически. Эта неспособность к распределению выходных сигналов может показаться неважной, однако в результате мы получаем строгие ограничения по расширению систем. В результате значительно увеличиваются затраты, связанные с установкой и поддержкой избыточных датчиков.

Итак, наша задача — разработать методику для решения (или, по крайней мере, значительного смягчения) перечисленных выше проблем с калибровкой, дрейфом и формированием сигналов. Найдя какой-нибудь простой способ распределения выходных сигналов датчика, мы решим вопрос масштабирования. Давайте посмотрим, как это сделать, и исследуем влияние нового подхода на мир датчиков.

Примечание

Несмотря на огромное разнообразие датчиков (электронные, механические, химические, оптические и др.), в этой книге мы сосредоточимся на электронных датчиках по той простой, но весомой причине, что они позволяют легко реализовывать интерфейс между их выходами и вычислительным элементом (обычно — микропроцессором). Именно вычислительный элемент наделяет датчик характеристиками интеллекта — очень ценное дополнение, как мы увидим позже.

1.2. Начнем сначала — оцифровка выходного сигнала датчика

Когда инженеры проектируют систему, в которой применяются датчики, они строят математическую модель реакции датчика на измеряемый физический параметр, а также математическую модель желаемой реакции схемы формирования сигнала на выходной сигнал датчика. Затем эти модели реализуют в виде электронной схемы. Моделирование — это, конечно, хорошо, но важно помнить, что модели — это только приближения (хотя, обычно, и довольно точные) реальной реакции в физической реализации. Было бы очень хорошо, если бы систему можно было удерживать в рамках математических моделей — числа не имеют дрейфа во времени, и ими можно точно и легко манипулировать.

По сути, дисциплина "Цифровая обработка сигнала" (ЦОС), в которой сигналами манипулируют, скорее, математически, чем с помощью электронных схем, хорошо развита и находит широкое применение. В рамках ЦОС легко реализуются стандартные преобразования, наподобие фильтрации для устранения ненужных шумов или отображения частот для идентификации отдельной составляющей сигнала. Более того, применяя принципы ЦОС, можно выполнять операции, которые были бы невозможны даже в случае применения самой передовой электронной схемы.

По этой причине современные разработчики включают в схему формирования сигнала каскад, в котором аналоговый электрический сигнал преобразуется в цифровой (числовой) вид. Этот этап, называемый *аналогово-цифровым преобразованием*, жизненно важен, поскольку сигналом датчика, преобразованным в цифровые значения, можно манипулировать с помощью программного обеспечения микропроцессора. Аналогово-цифровые преобразователи (АЦП) — это обычно приборы на одном полупроводниковом кристалле, которые характеризуются высокой точностью и стабильностью при изменяющихся внешних условиях.

Схема формирования сигнала зачастую может быть значительно сокращена, поскольку большая часть схемы компенсации воздействия окружающей среды обычно включена в состав АЦП, а фильтрация может быть реализована программно. Как мы скоро увидим, эта комбинация сокращенного электронного оборудования и способности оперировать почти исключительно в мире математики дает потрясающие преимущества как с точки зрения исполнения системы, так и в плане бизнес-перспектив.

1.3. Следующий шаг — добавим немного интеллекта

После того как сигнал датчика оцифрован, существует два основных варианта обработки цифровых значений и алгоритмов работы с ними. Мы можем или реализовать цифровое оборудование самостоятельно, что значительно "зацементирует" наш алгоритм обработки, или же использовать микропроцессор, что обеспечит необходимую мощность вычислений. Вообще, изготовленное самостоятельно оборудование может работать быстрее, чем системы, управляемые микропроцессором, однако обычно — за счет возросшей стоимости изделия и ограниченной гибкости. Микропроцессоры, хотя они и не всегда такие же быстродействующие как индивидуальные аппаратные решения, имеют одно существенное преимущество: они дают гибкость в проектировании и ведут к снижению себестоимости систем, поскольку могут применяться в разных ситуациях, а не только в единственном применении.

Как только у нас появляется встроенный интеллект, мы в состоянии решить сразу несколько из перечисленных выше проблем. Калибровку можно автоматизировать, дрейф составляющих — фактически устранять с помощью одних только математических алгоритмов обработки, а влияние изменений окружающей среды — компенсировать за счет периодического отслеживания условий и автоматической подстройки. Добавление "мозгов" *намного* упрощает жизнь разработчика.

Как будет показано в главе 3, для разработки изделий, где требуется малая стоимость, высокая степень интеграции (т.е. большой объем функциональности на одном полупроводниковом кристалле) и способность эффективно выполнять программы со множеством ветвлений и интенсивными вычислениями[5], стремительно возрастает спрос на сравнительно новый класс микропроцессоров: *цифровые контроллеры сигналов* (ЦКС). Хотя, обычно, ЦКС не столь скоростные, как самостоятельно разработанное оборудование, во многих случаях их быстродействия достаточно для выполнения необходимых алгоритмов (и, по большому счету, что еще нужно?).

---

[5] Под *ветвлением* понимают изменение порядка обработки инструкций программы. *Интенсивные вычисления* требуют значительной части процессорного времени на выполнение математических операций.

1.4. И в завершение — быстрый и надежный обмен данными

Остается только один нерешенный вопрос: распределение выходных сигналов датчика с тем, чтобы можно было легко масштабировать системы. И опять таки, тот факт, что данные датчика — цифровые позволяет достоверно выполнить это требование. Точно так же как обмен информацией повышает ее ценность в мире людей, обмен результатами измерений с другими компонентами внутри системы или с другими системами повышает ценность этих измерений. Для этого необходимо оборудовать интеллектуальный датчик стандартными средствами передачи его информации другим элементам. Используя стандартные методы обмена данными, мы гарантируем, что информация с датчика может распределяться максимально широко, легко и достоверно. Таким образом мы максимизируем эффективность датчика и информации, которую он предоставляет.

1.5. Совместите все это вместе, и получится интеллектуальный датчик

Итак, мы определили три характеристики, которые большинство инженеров считают обязательными для интеллектуального датчика:

- **чувствительный элемент**, измеряющий один или несколько физических параметров (по сути — уже рассмотренный нами традиционный датчик);
- **вычислительный элемент**, анализирующий данные измерений, полученные с помощью чувствительного элемента;
- **интерфейс** с внешним миром, позволяющий устройству обмениваться информацией с другими компонентами в большой системе.

Именно два последних элемента отличают интеллектуальные датчики от их традиционных собратьев (рис. 1.3), поскольку они предоставляют возможность превращать данные непосредственно в информацию для локального применения и передачи другим элементам системы.

По существу, интеллектуальные датчики "сплющивают" мир датчиков, позволяя им соединяться с другими датчиками, расположенными где-нибудь поблизости или на другом конце Земли, и выполнять задачи, которые раньше были просто невыполнимы. И что не менее важно, теперь компании могут модифицировать свои изделия, просто изменяя конфигурацию управляющего программного обеспечения, поскольку именно оно определяет большую часть функциональности.

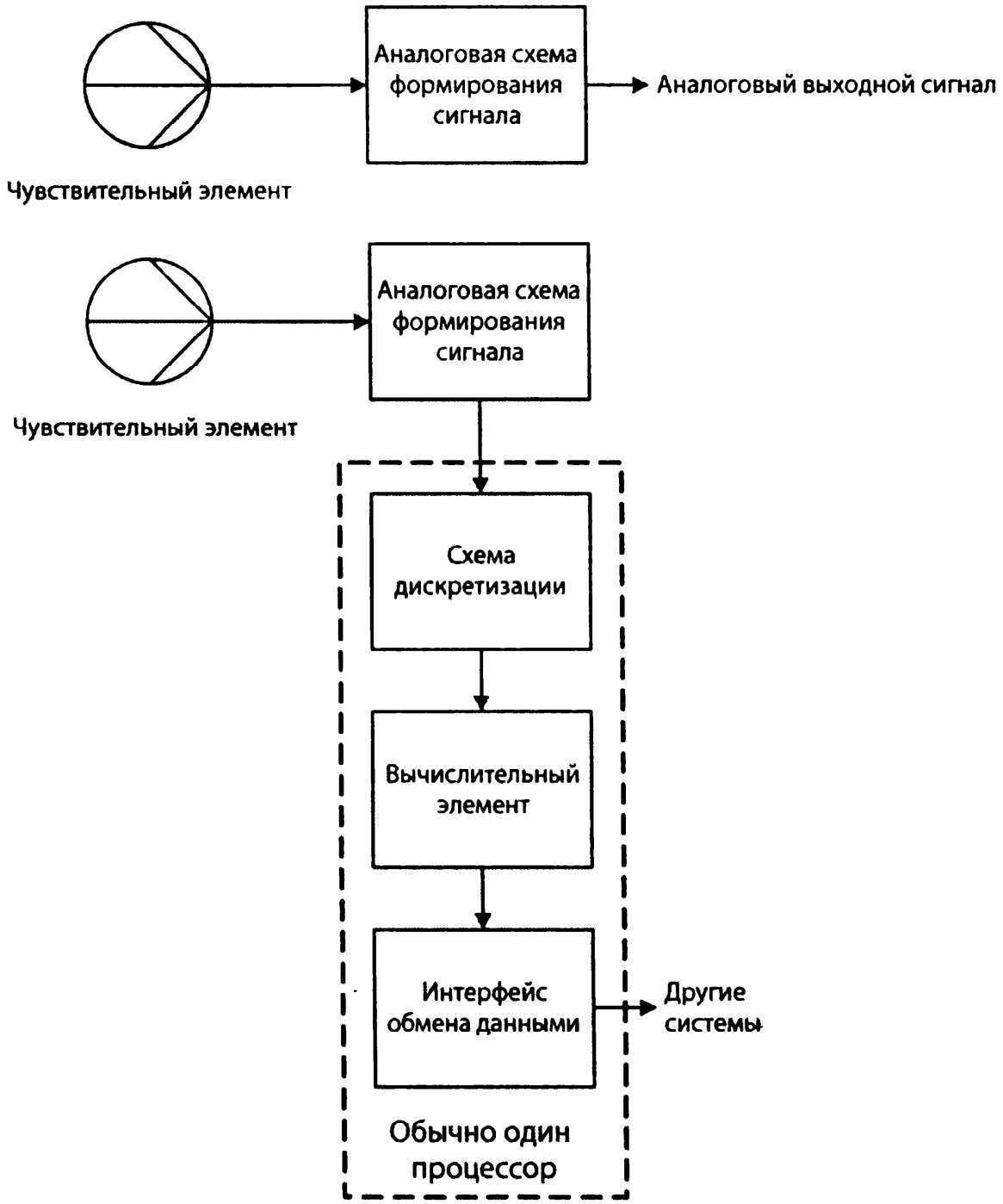


Рис. 1.3. *Блок-схемы стандартного (вверху) и интеллектуального (внизу) датчиков*

Из этого вытекают два очень важных следствия для поставщиков интеллектуальных датчиков. Во-первых, происходит значительное смещение фокуса с продукции, базирующейся на аппаратном обеспечении, на, к продукции, базирующейся на программном обеспечении. Хотя, разумеется, для датчика должна существовать аппаратная платформа (в конце концов, это — физическое устройство), аппаратное обеспече-

ние больше не является основным мерилом ценности им становится программное обеспечение, управляющее интеллектуальным датчиком. Поскольку изготовитель может добавлять или убирать свойства, изменяя программно разряды конфигурации, он получает возможность видоизменять набор своих изделий почти мгновенно, а конкретная конфигурация может не фиксироваться жестко вплоть до завершающего тестирования или поставки. Одна аппаратная платформа может использоваться в различных устройствах, нацеленных на разные сегменты рынка с разной ценовой политикой, а после разработки новых свойств не требуется никаких дополнительных затрат для их внедрения в устройство.

Второе следствие... Поскольку интеллектуальный датчик соединен с внешним миром, у поставщика теперь появляется возможность собирать информацию о работе своих датчиков в той или иной области и обновлять управляющее программное обеспечение уже после того, как датчики покинули завод. Подобная информация с датчиков дает изготовителям не только беспрецедентное понимание потребностей и интересов их клиентов, но и обеспечивает достоверными данными, требуемыми для выявления наиболее важных (а значит — и наиболее ценных для клиентов) аспектов. Вооруженные этой информацией, изготовители датчиков могут быстро добавлять новые возможности, предлагать определенные конфигурации под конкретные условия или организовать техническую поддержку — и все это, без физического контакта с самим датчиком. Теперь услуги могут предоставляться прямо из центральных офисов, что выгодно экономически и позволяет поставщикам получить дополнительную прибыль. Пример этого описан в "Harvard Business Review": "Большинство производителей не могут запрашивать более $90–110 за час своей технической поддержки из-за ценовой политики местных конкурентов. Однако компания GE Energy, благодаря своей эффективной, реализованной на базе компьютерной сети службы удаленного обслуживания, может запрашивать от $500 до $600 за час той же технической поддержки. Что еще важнее, информация, получаемая в результате постоянного мониторинга, позволяет GE браться за дополнительные задачи, наподобие создания каталога комплектующих у клиента или предоставления техническому персоналу клиента и GE полного доступа к унифицированным данным и знаниям о состоянии оборудования[6]".

---

[6] Статья Глен Олмендингер (Glen Allmendinger) и Ральфа Ломбреглиа (Ralph Lombreglia) "Four Strategies for the Age of Smart Devices", опубликованная в выпуске "Harvard Business Review" за октябрь 2005г. Перепечатка R01101

1.6. Почему бы не сделать все датчики интеллектуальными?

Несмотря на все преимущества преобразования традиционных, автономных датчиков в интеллектуальные, соединенные с другими датчиками, существуют ли какие-либо причины, не позволяющие сделать все датчики интеллектуальными? Ответ: "Да", — и важно различать ситуации, в которых добавление рассматриваемой разновидности искусственного интеллекта и способности к объединению неприемлемо. В общем случае добавлять интеллект не имеет смысла в следующих случаях:

- затраты на доработку и производство изделия не могут быть компенсированы клиентами за приемлемый промежуток времени,
- конечный пользователь не в состоянии или не желает создавать инфраструктуру, необходимую для организации питания интеллектуального устройства и/или для обмена данными с ним;
- физические ограничения конкретной области применения препятствуют добавлению дополнительной схемы, требуемой для реализации интеллекта и информационных соединений.

Затраты на доработку и производство превышают выгоду, оцененную потребителем

Для того чтобы любое изделие оставалось дееспособным на протяжении длительного времени, клиенты должны быть готовы платить за него цену, достаточную для покрытия расходов на его разработку и изготовление. Этот принцип справедлив как для передовых технических продуктов, наподобие интеллектуальных датчиков, так и для более прозаических изделий, наподобие бумажных полотенец. Ни одна компания не может позволить себе долго выпускать изделие, от которого она получает меньше денег, чем затрачиваются на его производство. Прежде чем инвестировать время и ресурсы в то, чтобы наделить свои устройства интеллектом, производитель датчиков должен определить, согласятся ли его клиенты платить больше, чтобы покрыть хотя бы затраты на разработку и любые другие возросшие производственные расходы (если конечно у производителя нет специальных резервных фондов на новые разработки). Пока клиенты не оценят по достоинству выгоды, предоставляемые интеллектуальными приборами, производителю лучше заниматься неинтеллектуальными изделиями.

На первый взгляд, кажется, что клиенты должны ясно видеть выгоды от добавления интеллекта, однако некоторые приложения настолько критичны к расходам и дают настолько малые прибыли, что клиенты

напрочь отказываются инвестировать в новую технологию. В качестве примера можно привести одноразовую посуду, на которой производители получают доли цента прибыли от готовой единицы товара. При такой мизерной прибыли изготовители подобной продукции ни за что не будут тратить много денег на оборудование. Их закупочная политика сосредоточена на самых низких ценах, и все, без чего можно обойтись, не представляет для них никакой ценности.

Отсутствие необходимой инфраструктуры

Еще один важный аспект, препятствующий внедрению интеллектуальных датчиков — отсутствие у клиента того минимального уровня инфраструктуры, который требуется для обеспечения питания датчиков и создания информационных каналов. Этот аспект тесно связан с предыдущим, когда производство датчиков экономически невыгодно, поскольку добавление инфраструктуры для функционирования датчиков в глазах клиента не оправдано. Однако и линии питания, и информационные каналы — обязательны, поскольку без питания датчики не смогут даже включиться, а без информационных каналов — неспособны передавать информацию.

Этот аспект реализации систем с интеллектуальными датчиками не следует недооценивать. Хотя сегодня все больше и больше производственных предприятий переходят на цифровые сети, дополнительная инфраструктура все еще связана с существенными расходами. Некоторые из новых сетевых протоколов обеспечивают питание непосредственно по проводам, используемым для обмена данными (например, PoE — Power-over-Ethernet), однако в большинстве случаев (особенно на старых заводах) прокладка проводов обходится слишком дорого.

Для решения двух рассмотренных выше проблем на рынке появились новейшие беспроводные датчики с малым энергопотреблением, однако такие решения обычно более дорогостоящие (хотя их полная стоимость долгосрочной эксплуатации может оказаться ниже).

Реализации дополнительной электронной схемы препятствуют внешние условия

Последней и наименее распространенной преградой для использования интеллектуальных датчиков являются специфические внешние условия, не позволяющие применить какую-либо дополнительную электронную схему. К таким условиям можно отнести габаритные ограничения, рабочую температуру, сильную вибрацию или подверженность действию активных химических веществ. В таких случаях единствен-

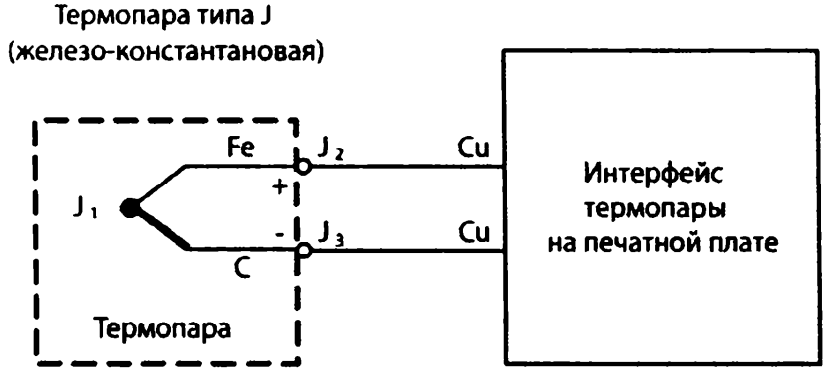
ный вариант — это прочный традиционный датчик, хотя его характеристики обычно можно значительно улучшить, как можно раньше преобразовав измеренный параметр в цифровое значение.

1.7. Реальные примеры интеллектуальных датчиков

В завершение этой главы рассмотрим три реальных примера интеллектуальных датчиков: два — из сферы управления промышленными процессами, и один — из области управления транспортными средствами.

Многоканальный цифровой датчик температуры

Температура — широко используемый параметр при управлении различными промышленными процессами, и один из самых распространенных датчиков температуры — это термопара. В определенном смысле термопара — чрезвычайно простой чувствительный элемент, который состоит из двух разнородных металлов, соединенных вместе в одной точке. В результате проявляется так называемый *эффект Зеебека*, когда в месте соединения двух металлов возникает напряжение, изменяющееся в зависимости от температуры соединения (рис. 1.4).



Обратите внимание на то, что в действительности здесь присутствует три соединения разнородных металлов, и, таким образом, в схеме действуют три термопары:

J_1 – чувствительная термопара (единственная, созданная намеренно)
J_2 – соединение железного проводника с медным печатным проводником платы
J_3 – соединение константанового проводника с медным печатным проводником платы

Рис. 1.4. Схема базовой термопары

Мы рассмотрим термопары подробно в одной из последующих глав, на данный же момент важно понимать тот факт, что термопара дает

очень малое напряжение (порядка милливольт), которое зачастую замеряется в присутствии сильного электронного шума (может достигать сотен вольт). Еще более все усложняет тот факт, что температурная характеристика термопар — нелинейная, и потому обычно перед считыванием данных о температуре требуется операция линеаризации.

Существуют и другие серьезные трудности с использованием термопар, однако на примере этих двух удобно проиллюстрировать применение интеллектуальных датчиков для обеспечения точного считывания в экстремальной среде: литьевой машине.

Для тех читателей, которые незнакомы с литьевым формованием, скажу, что это — производственный процесс, при котором твердые пластмассовые окатыши плавятся в результате нагрева до температуры порядка 150–480°C. Затем расплавленная пластмасса под высоким давлением (порядка 4 500–13 600 кг на квадратный дюйм) нагнетается в литейную форму, где ей дают застыть до твердого состояния. Этот процесс повторяется быстро, чтобы производитель мог получать образцы с максимальной скоростью. Ключевой фактор успешного литьевого формования — обеспечение короткого рабочего цикла (т.е. времени, затрачиваемого на то чтобы открыть и закрыть литьевую форму) при малом проценте брака. До тех пор пока формовочная машина может производить образцы хорошего качества с прибылью от каждого из них, она, по сути, "печатает деньги" со скоростью, пропорциональной периоду рабочего цикла. При этом важным аспектом является надлежащее регулирование температуры пластмассы в различных точках формовочной машины, что требует размещения в ключевых точках процесса датчиков температуры (термопар).

К сожалению, использованию термопар препятствует дороговизна применяемых для их создания проводов. Формовочные машины, в основном, — довольно громоздкие и содержат множество зон мониторинга температуры (от 250 зон в крупных системах). Таким образом, провода термопар проходят немалые дистанции до соответствующих регуляторов температуры. Одна из компаний-пионеров в области регулирования температуры пришла к выводу, что может сэкономить своим клиентам кучу денег, если считанные данные оцифровывать в самой формовочной машине, а затем доставлять их в регулятор по традиционным (и недорогим) медным кабелям. Более того, это можно было сделать для множества каналов передачи данных, и, поскольку считанные данные термопар изменяются относительно медленно, оцифрованные данные при передаче в схему управления могли бы быть мультиплексированы по времени. В конечном итоге для каждого устройства были обеспечены

до 96 каналов передачи данных от термопар, и таким образом дорогостоящая, подверженная шумам система из длинных проводов термопар превратилась в простую в обслуживании пару медных проводов.

Датчики потока

Наряду с тем, что пластмасса должна удерживаться в расплавленном состоянии до тех пор, пока не окажется внутри формы, еще одна задача заключается в получении из расплавленной пластмассы твердых образцов. Для этого в форму встроены каналы охлаждения, по которым для быстрого отвода тепла от пластмассы циркулирует холодная вода или другая жидкость. Если поток охлаждающей жидкости тормозится, то охладитель будет нагреваться, поскольку дольше остается в контакте с горячей формой. Это, в свою очередь, снижает эффективность охлаждения и увеличивает время затвердевания, а значит — удлиняет рабочий цикл, снижая прибыльность. Само собой разумеется, ни одного рационального предпринимателя это не устроит, и в "разумные" формовочные машины для контроля за скоростью движения охладителя в системах охлаждения встраивают датчики потока. Если известны некоторые характеристики самой охлаждающей жидкости, температура жидкости в двух разных точках и скорость потока, то можно вычислить число BTU (британских тепловых единиц), передаваемое в системе охлаждения из одной точки в другую.

Датчики потока выпускают в самых разнообразных конфигурациях, однако одна из самых популярных известна под названием *внутренний датчик потока*. В датчиках этого типа в поток охлаждающей жидкости помещается устройство в виде крыльчатки, скорость вращения которого соответствует скорости потока жидкости. Со временем подшипник, на котором установлена крыльчатка, снашивается, приводя к эксцентрическому движению крыльчатки, что значительно снижает качество считанных данных о потоке. Одна ведущая компания разработала переносное портативное устройство, в котором для компенсации износа крепления крыльчатки использовались специальные алгоритмы фильтрации. Это позволило дольше поддерживать требуемую точность считывания данных о потоке и значительно продлевало срок службы датчика.

Датчик положения в рулевой системе с управлением "по проводу"

Последний пример реального интеллектуального датчика взят из области управления транспортными средствами, в частности — рулевых приводов с управлением "по проводу" для плавучих суден. В обычной

механической системе рулевого управления присутствует физическое соединение между рулевым колесом и исполнительными элементами (например, колесами автомобиля или рулем корабля). Когда водитель поворачивает рулевое колесо, это движение передается через последовательность механических звеньев на исполнительный элемент. В зависимости от конфигурации системы, водитель чувствует отдачу (дороги или воды), что помогает ему соответствующим образом откорректировать свои действия.

Несмотря на их надежность, механические системы рулевого управления характеризует невозможность использовать более одного рулевого колеса без крайне сложных (и дорогостоящих) механических приспособлений. Это не так существенно для автомобиля, где вполне достаточно и одного рулевого колеса, однако для больших кораблей было бы неплохо разместить один штурвал на носу для управления швартовкой, а второй — на корме для изменения курса во время плавания.

В отличие от обычных систем, в системе рулевого управления "по проводу" большая часть механических соединений замещена электронными схемами. Хотя рулевой по-прежнему поворачивает штурвал, рулевое колесо соединено с исполнительными механизмами электронным, а не механическим способом. Это предоставляет несколько явных преимуществ, из которых не самым последним является существенное сокращение габаритов и веса системы рулевого управления. Кроме того, становится проще реализовать два или более рулевых колес, поскольку они могут соединяться электронным кабелем без каких-либо дополнительных механических звеньев.

Однако управление "по проводу" имеет один недостаток: до недавнего времени водитель не чувствовал отклика исполнительных механизмов, что вызывало у него неприятное чувство разъединения с транспортным средством. Но с изобретением специального материала, который меняет свои свойства в зависимости от силы пересекающего его магнитного поля, все изменилось. С помощью датчиков положения, измеряющих позицию рулевого колеса много раз в секунду, уникальная система рулевого управления, разработанная одним из мировых новаторов, создает имитацию отдачи посредством регулирования плотности специального материала. При этом учитывается ряд факторов, включая скорость вращения рулевого колеса и условий несущей поверхности, что создает у водителя не только ощущение комфорта в управлении, но и повышает безопасность вождения.

1.8. Обзор остальной части книги

Хотя большинство глав этой книги можно читать независимо от других, рекомендуем прочитать главу 2, "Интуитивная цифровая обработка сигналов", перед главой 3, "Возможности цифрового контроллера сигналов dsPIC", поскольку некоторые вопросы, обсуждаемые в главе 2, необходимы для полного понимания определенных функциональных возможностей ЦКС dsPIC. В главе 2 мы развиваем интуитивное понимание принципов цифровой обработки сигналов (ЦОС), которые нам понадобятся для реализации трех типов интеллектуальных датчиков, исследуемых далее в книге. При этом мы не ставили перед собой цель превратить читателя в ведущего специалиста по ЦОС — для этого существует достаточно прекрасных книг, в которых данный предмет рассмотрен намного шире, чем здесь. Вместо этого мы хотим развить у читателя понимание способов применения определенных принципов ЦОС, чтобы у него было твердое представление о том, как использовать конкретные подходы и условия успешной реализации поставленных задач.

Базируясь на основах ЦОС, изложенных в главе 2, глава 3, "Возможности цифрового контроллера сигналов dsPIC", подробно описывает семейство ЦКС dsPIC, которое прекрасно подходят для реализации интеллектуальных датчиков. Мы сосредоточим наше внимание на одном конкретном устройстве: dsPIC30F6014A, — которое обладает многими возможностями, характерными для всех микросхем dsPIC. Конечно же, мы ознакомимся и с программным обеспечением для ЦКС dsPIC, а также с некоторыми программными инструментами и библиотеками, предоставляемыми компанией Microchip для ускорения разработки с помощью устройств dsPIC. В заключении главы 3 мы поэкспериментируем с базовым набором программного обеспечения, который мы будем применять для построения специализированных интеллектуальных датчиков в последующих главах.

Глава 4, "Умение быть хорошим коммуникатором", расширяет наше понимание устройств dsPIC через рассмотрение такого важного вопроса, как обмен данными между процессором и другими устройствами. Эти устройства могут быть внешними по отношению к кристаллу, однако входят в состав устройств в качестве периферии, повышающей функциональность системы. Или же они могут представлять собой совершенно независимые системы, физически удаленные от самого датчика. Эти две ситуации совершенно отличаются по своей природе и, разумеется, требуют разных методик обмена данными. К счастью, выход из этой ситуации предоставляют устройства dsPIC, оснащенные различными коммуникационными интерфейсами. В частности, мы рассмотрим

обмен данными по протоколу RS232/485, последовательный периферийный интерфейс SPI (Serial Peripheral Interface) и шину CAN (Controller Area Network). Все это — промышленные стандарты для интерфейсов обмена данными, и их применение позволяет устройствам, основанным на dsPIC, взаимодействовать с самыми разнообразными системами. В частности, шина CAN обеспечивает надежную, среднескоростную (до 1 Мбит/с) связь между множеством узлов в сети и используется в огромном числе промышленных и транспортных изделий. Как и в случае со многими другими коммуникационными протоколами, руководство о том, что является главным, а что — второстепенным, сэкономит драгоценное время читателей и позволяет им обойти ряд "подводных камней".

Глава 5, "Базовые средства разработки для ЦКС dsPIC", продолжает работу, начатую в главах 3 и 4, путем сборки набора модульных программных компонентов, которые будут использованы для реализации конкретных задач на процессоре dsPIC. После формирования средств разработки, глава завершается реализацией многоканального блока фильтров на основе программной оболочки, представленной в главе 3. Он позволит нам надежно и быстро фильтровать (устранять шум) в нескольких каналах информационных сигналов.

Работая с главами 6–8, мы детально исследуем три специфических сферы применения интеллектуальных датчиков: измерение температуры, давления и потока. С целью реализации надежных интеллектуальных датчиков для каждого сферы применения эти главы продолжат построение программной оболочки, разработанной в главах 3 и 4, а также — структурных блоков ЦОС, разработанных в главе 5. Ко времени завершения главы 8 у нас будет прочный фундамент для проектирования интеллектуальных датчиков и путей их реализации с помощью ЦКС dsPIC.

Основной материал книги завершается главой 9, "Взгляд в будущее", которая дает обзор перспектив интеллектуальных датчиков и цифровых контроллеров сигналов. Цель этой главы, включающей обсуждение преимуществ открытых протоколов по сравнению с патентованными и тенденций повсеместного объединения устройств в сети, — проинформировать читателей о ключевых аспектах, которые следует учитывать при разработке изделий, нацеленных в будущее.

В дополнение к основной части, книга содержит три приложения. Приложение А описывает содержимое прилагаемого к книге компакт-диска. Приложение Б посвящено инициализации dsPIC и содержит соответствующий стартовый код. В приложении В затрагиваются важные,

однако зачастую оставляемые без внимания аспекты функционирования системы, а именно: буферизированный ввод-вывод с управлением по прерыванию, который можно использовать для отладки или постоянного обмена данными.

Поскольку ни одна книга не может претендовать на право называться *всеобъемлющей* как по вопросу интеллектуальных датчиков, так и в отношении ЦКС dsPIC, на прилагаемом компакт-диске в папке Ресурсы находится файл `index.html` с перечнем других первоисточников, которые читатель может использовать для углубленного изучения вышеупомянутых вопросов.

Наконец, самую свежую информацию об этой книге, а также дополнительные примеры программ и ресурсы можно найти на Web-сайте `www.intelligentsensordesign.com`.

2

Интуитивная цифровая обработка сигналов

Если у вас есть идея, — это хорошо, но еще лучше, если вы знаете, как ее воплотить в жизнь.

— Генри Форд

Итак, мы уже понимаем ценность интеллектуальных датчиков как для конечных пользователей, так и для тех, кто их производит и продает. Наступил момент заглянуть вглубь процессов, происходящих внутри интеллектуальных датчиков, и первым шагом на нашем пути станет развитие твердого, интуитивного понимания принципов цифровой обработки сигналов (ЦОС). В отличие от многих учебников по ЦОС, здесь основное внимание будет уделено представлению и применению важных принципов, а не их развитию "с нуля". И причина этому очень проста: для углубленного рассмотрения предмета потребовалась бы отдельная книга, а не одна глава. Другие авторы уже проделали превосходную работу по рассмотрению этой темы с более строгим подходом, и наша цель здесь — не пытаться ужать их труд до бессмысленного конспекта, а, скорее, — дать понимание путей использования определенных ключевых понятий для превращения "сырых" данных датчика в значимую информацию. К концу этой главы читатель научится легко определять ключевые требования обработки сигналов для типичных сфер применения, а также идентифицировать соответствующий процесс для получения желаемых результатов измерения.

2.1. Фундаментальные понятия обработки сигналов

Хотя представленное ниже обсуждение ЦОС — не такое строгое, как в большинстве академических работ, для нас важно достичь ясного понимания нескольких ключевых понятий, формирующих базис данного предмета. Начав с точных определений понятий "сигнал" и "шум", мы перейдем к анализу сигналов с точки зрения времени и частоты,

а в завершение затронем некоторые базовые вопросы, связанные с фильтрацией — методикой, которая обычно применяется для извлечения необходимой информации из зашумленных данных. Мы начинаем создавать обобщенную оболочку, которую сможем применять в широком диапазоне сфер применения интеллектуальных измерений.

Конкретизация понятий "сигнал" и "шум"

Словарь определяет понятие "сигнал" как "импульс или колебание электрической величины (например, напряжения или тока), изменение которого представляет собой кодированную информацию"[7], и это определение является хорошей отправной точки. Электронные сигналы обладают одной интересной чертой: им свойствен принцип суперпозиции или наложения. Этот принцип заключается в том, что величина двух или более сигналов, проходящих через одну и туже точку в одной и той же среде в определенный момент времени, представляет собой простую сумму величин отдельных сигналов в данный момент времени. Например, если бы у нас было N разных сигналов, обозначенных как $V_0(t)$, $V_1(t)$,..., $V_{N-1}(t)$, то результирующий сигнал $V_S(t)$, представляющий собой суперпозицию этих N сигналов, математически можно представить как

$$V_S(t) = V_0(t) + V_1(t) + \ldots + V_{N-1}(t) \qquad (2.1)$$

Как оказалось, принцип суперпозиции — очень мощный инструмент. С его помощью зачастую можно разложить сложный сигнал от датчика на отдельные, более простые компоненты, что упрощает анализ проблемы и проектирование конечных систем. Этот принцип активно используется в примере разработки реального датчика, представленном позже в этой книге, для создания соответствующих методик обработки сигналов для каждой сферы применения. Но для начала рассмотрим, каким образом суперпозиция ведет к лучшему пониманию всех сигналов датчика.

Обратимся к схеме (рис. 2.1), которая содержит термопару, соединенную с вольтметром в идеализированной окружающей среде. Как было отмечено в главе 1, термопара производит аналоговое выходное напряжение $V_T(t)$, изменяющееся во времени t в зависимости от температуры спая (рис. 2.2).

К сожалению, такая идеализированная среда существует только в нашем воображении (точно так же, как идеальная тишина в библиотеке существует только в фантазии библиотекаря).

---

[7] Webster's II New Riverside Dictionary, 1984, Houghton Mifflin Company.

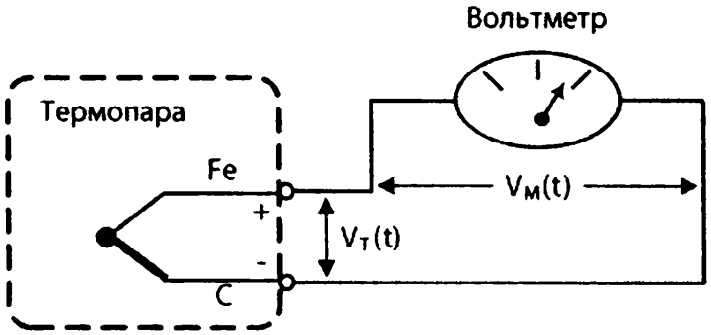


Рис. 2.1. Базовая идеализированная схема включения термопары

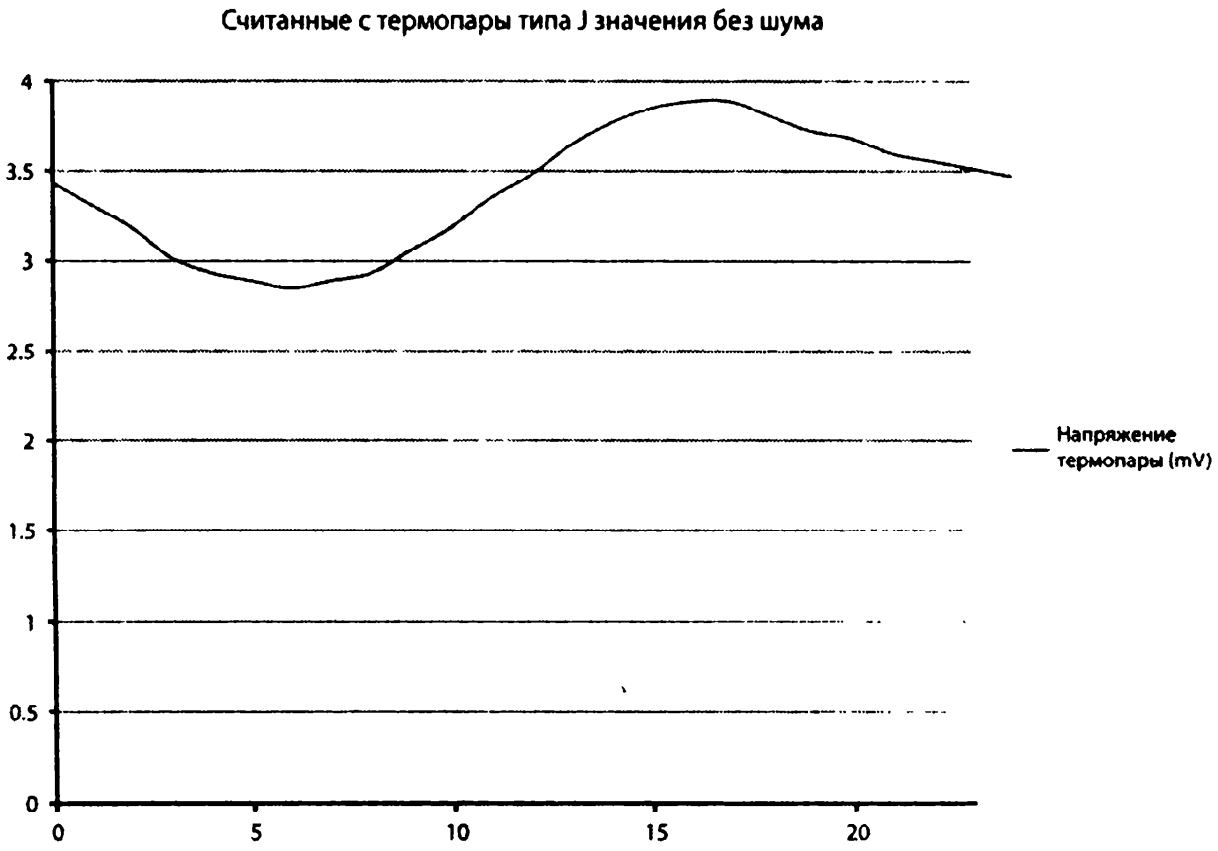


Рис. 2.2. Пример идеализированного сигнала термопары

Как в самой тихой из библиотек присутствует шум, так и реальные электрические схемы (рис. 2.3) содержат электронный шум, источником которого является как окружающая среда, так и компоненты самой схемы. Таким образом, более точное представление базовой схемы включения термопары содержит генератор электрического шума, производящий компоненту напряжения $V_N(t)$ (пример — на рис. 2.4), наложенную на "истинный" сигнал термопары $V_T(t)$ (см. рис. 2.2). Пример результирующего сигнала представлен на рис. 2.5.

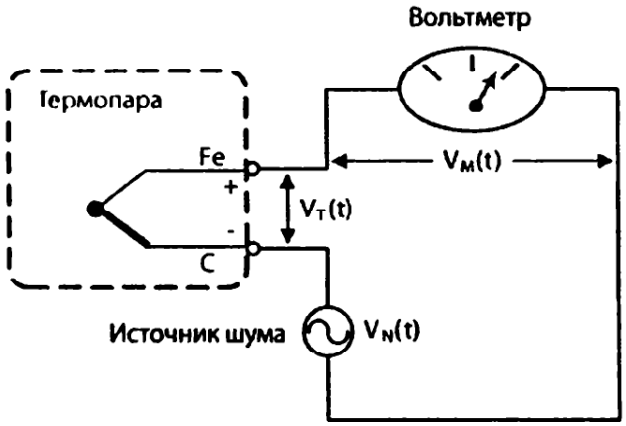


Рис. 2.3. Более реалистичная модель включения термопары при наличии шума

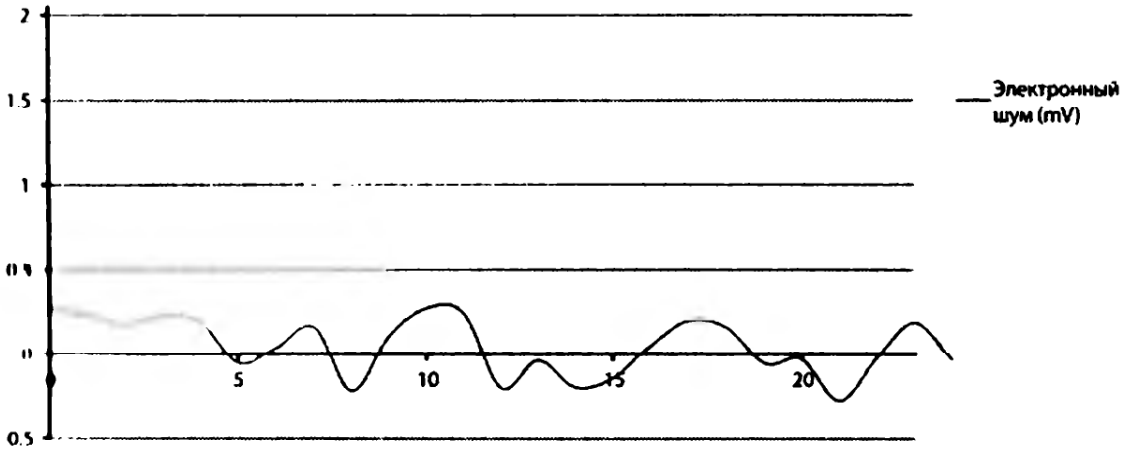


Рис. 2.4. Пример сигнала электронного шума

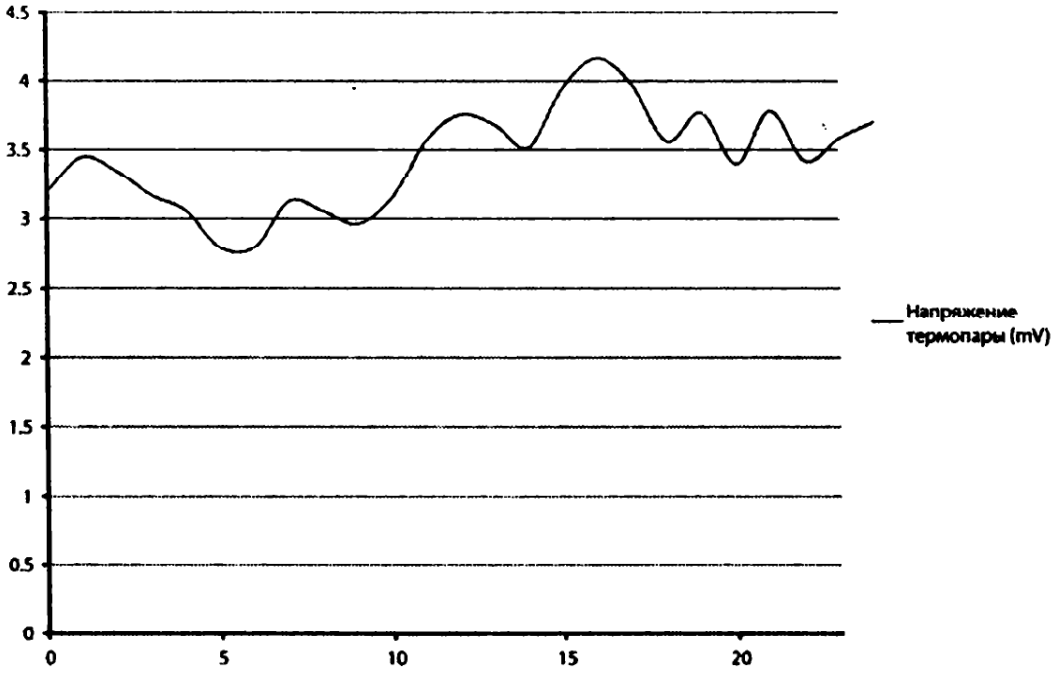


Рис. 2.5. Пример зашумленного сигнала термопары

Для стороннего наблюдателя это различие в действительности не заметно — он просто видит измеренное напряжение $V_M(t)$, которое содержит обе компоненты и равно

$$V_M(t) = V_T(t) + V_N(t) \quad (2.2)$$

Разумеется, конечный пользователь, как правило, не хочет, чтобы компонента шума присутствовала в измеренном сигнале. Его интересует "истинный" сигнал от термопары, содержащий полезную информацию, а не искаженный, маскирующий эту информацию. В зависимости от характеристик сигнала и шума, можно с точностью извлечь интересующий нас сигнал даже при наличии значительных уровней шумов. Для этого применяют методики, о которых сейчас и пойдет речь.

Наблюдение сигналов в частотной области

Любые реальные аналоговые[8] сигналы могут быть представлены в частотной области с помощью математической операции, известной как *преобразование Фурье,* и правильный выбор области (временной, в которой получают результат при измерении вольтметром или осциллографом, или частотной) может существенно упростить анализ в каждой конкретной ситуации обработки сигнала. Основной особенностью преобразования Фурье является то, что непрерывные, линейные[9] сигналы во временной области (как, например, напряжения из рассмотренного выше примера) могут точно представить суперпозицией ортогональных синусоидальных сигналов разных частот. Конечно, здесь мы видим массу пугающего математического жаргона, но что в этой операции самое ценное, — это то, что она позволяет достаточно легко определить частоты, в которых сосредоточена основная энергия сигнала (а это, по сути, определяет наиболее важные части сигнала).

Внести ясность поможет пример. Рассмотрим чистый синусоидальный сигнал (рис. 2.6) и представление в частотной области (рис. 2.7).

---

[8] Для наших целей аналоговым будем называть непрерывный во времени сигнал (т.е. такой сигнал, который не дискретизирован).

[9] В данном случае определение *линейные* относится не к прямолинейной функции, а, скорее, — к функции, которая подчиняется принципу суперпозиции. С математической точки зрения, если функция — это преобразование **H**[], а $y_0(t)$ — зависимость этой функции от входного значения $x_0(t)$, в то время как $y_1(t)$ — зависимость этой функции от входного значения $x_1(t)$, то **H**[] линейна тогда и только тогда, когда

$$\mathbf{H}[ax_0(t) + bx_1(t)] = a\mathbf{H}[x_0(t)] + b\mathbf{H}[x_1(t)] = ay_0(t) + by_1(t).$$

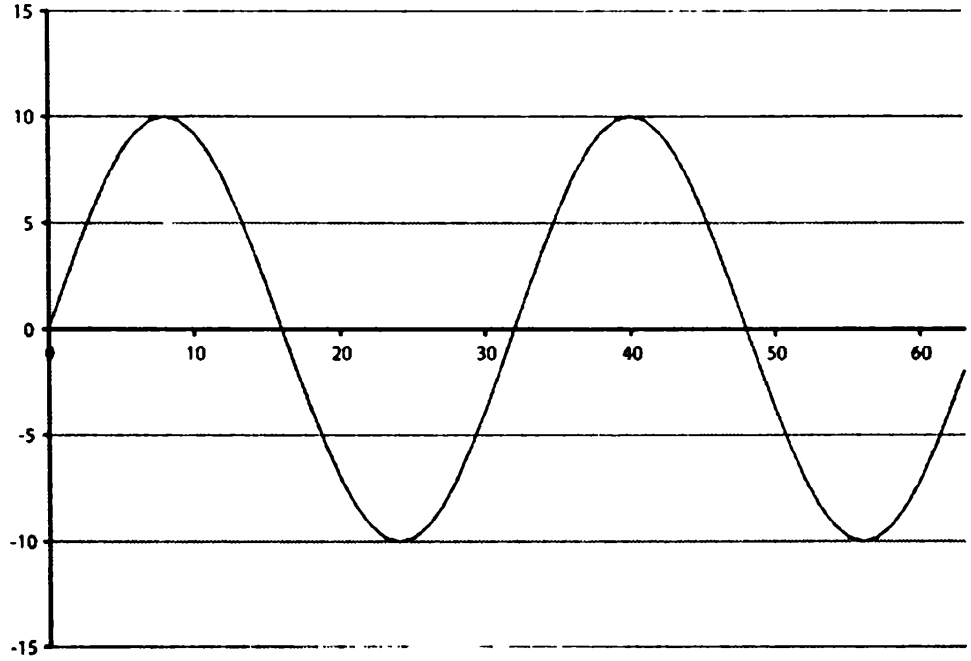


Рис. 2.6. *Синусоидальный сигнал во временной области*

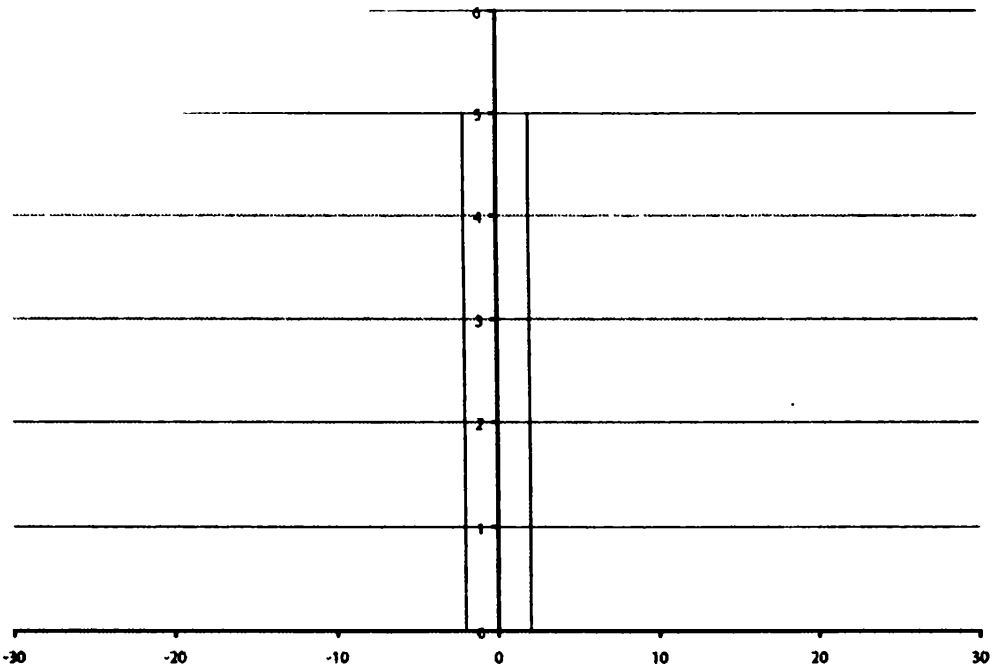


Рис. 2.7. *Представление того же сигнала в частотной области*

Заметьте очень интересное соотношение между этими двумя областями: непрерывному сигналу во временной области фактически соответствуют два пика в частотной области, распределенные вдоль оси частот! Хотя график на рис. 2.7 изображает два пика, распределение частотного спектра обусловлено ограничениями дискретной математики, которая применялась программой, сгенерировавшей данное изображе-

ние. С математической точки зрения, в частотной области присутствуют два всплеска, расположенных точно в точках, соответствующих частоте синусоиды. Если мы формируем более сложный сигнал, добавляя вторую синусоиду к первой, то по принципу суперпозиции мы получим новый сигнал, пример которого показан на рис. 2.8, а его представление в частотной области — на рис. 2.9.

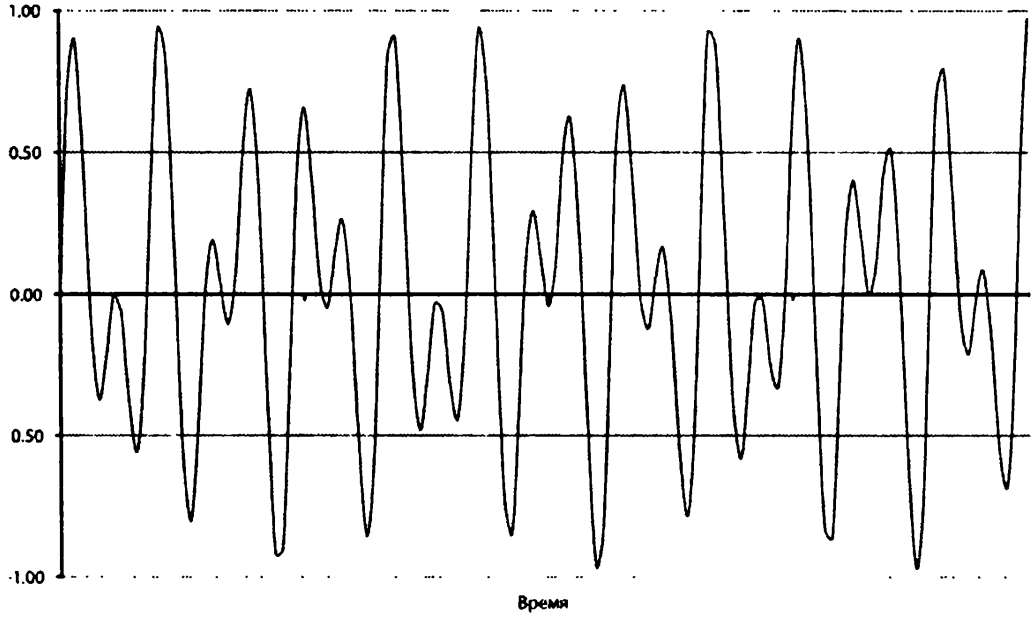


Рис. 2.8. *Сигнал DTMF для кнопки "5" во временной области*

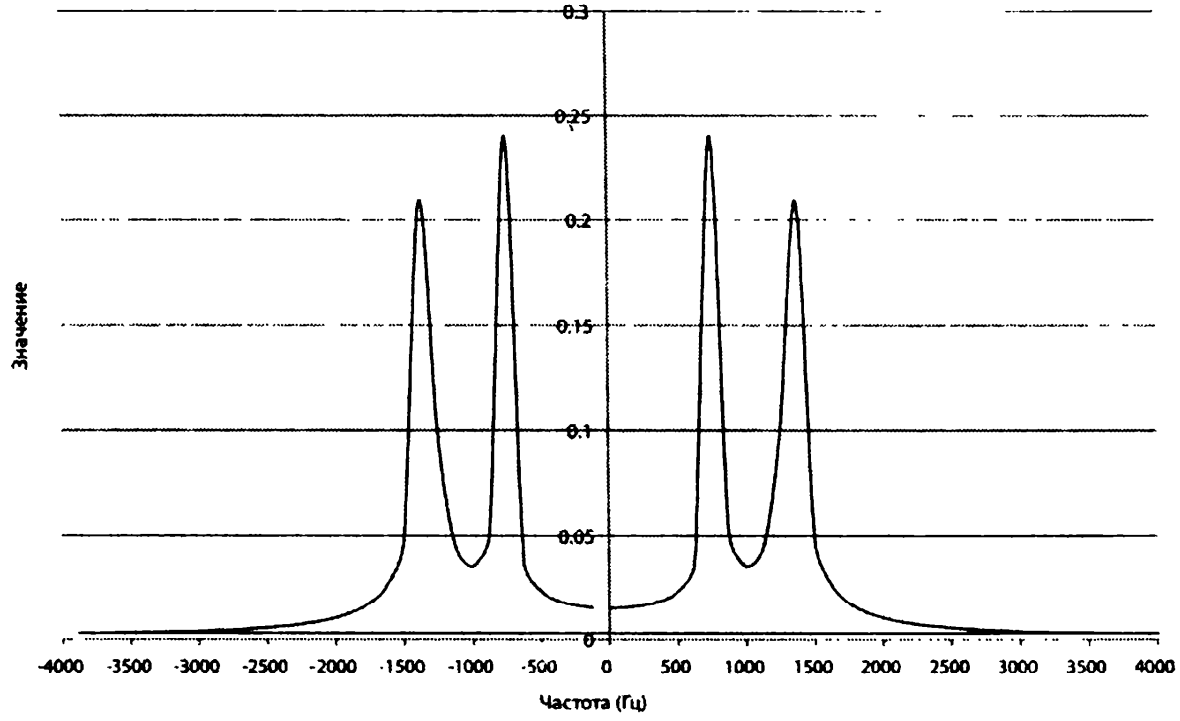


Рис. 2.9. *Представление сигнала DTMF в частотной области*

Здесь мы видим стандартный сигнал DTMF (Dual Tone Multi-Frequency — двухтональный многочастотный набор), который мы получаем при нажатии цифровой кнопки на телефоне с кнопочным набором. Добавление одной дополнительной частоты привело к значительной потере синусоидальности сигнала во временной области, а тот же сигнал в частотной области — это четыре всплеска, где два дополнительных пика соответствуют новой частоте.

Рис. 2.8 и рис. 2.9 демонстрируют один очень мощный аспект преобразования Фурье: принцип суперпозиции справедлив как для временной, так и для частотной области. Сигналы, которые суммируются во временной области, имеют частотный спектр, представляющий собой сумму спектров отдельных компонентов сигнала. Просто наблюдая этот сигнал в частотной области, разработчик может быстро определить его составляющие, что очень полезно при анализе и проектировании системы обработки, требуемой для извлечения интересующей информации. Концепция *спектрального анализа* (анализ представления сигнала в частотной области) — очень эффективная, и мы будем применять ее в реальных примерах в последующих главах.

При выполнении спектрального анализа часто фигурируют два термина:

- *полоса частот* — просто непрерывный диапазон частот;
- *пропускная способность* (*полоса пропускания*) — означает, главным образом, самую высокую частотную составляющую в сигнале.

Например, на рис. 2.9 разработчика могла бы интересовать полоса частот 770–1 477 Гц, которая содержит две частоты, образующие частный случай сигнала DTMF. Поскольку самая высокая частотная компонента сигнала — это 1 477 Гц, то теоретическая пропускная способность данного сигнала DTMF составляет 1 477 Гц.

Существует еще один важный аспект, имеющий отношение к представлению сигнала во временной и частотной области. Быстрое изменение сигналов во временной области формирует более широкий соответствующий спектр в частотной области, в то время как медленное изменение сигналов дает более узкий общий частотный спектр, ограниченный более низкими частотами. Как будет показано в следующем разделе, разработчики датчиков могут использовать этот факт для определения оптимальных подходов к устранению шума из интересующих сигналов. На рис. 2.10–2.12 показаны представления в частотной области низкочастотного (медленно изменяющегося), высокочастотного (быстро изменяющегося) и широкополосного (с частотами от низких до высоких) сигналов.

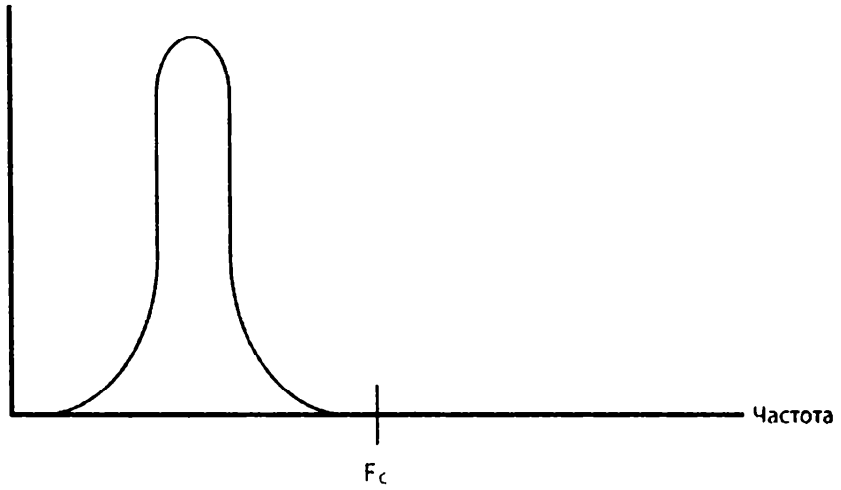


Рис. 2.10. *Низкочастотный сигнал в частотной области*

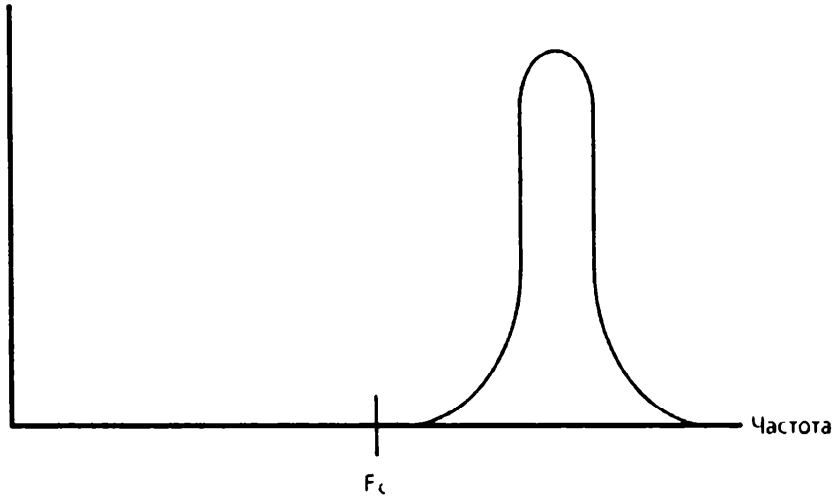


Рис. 2.11. *Высокочастотный сигнал в частотной области*

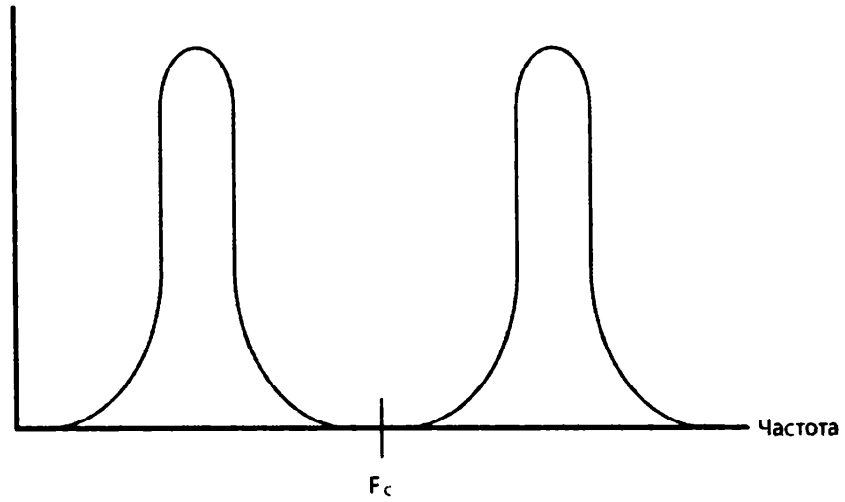


Рис. 2.12. *Комбинация низко- и высокочастотного сигнала в частотной области*

На практике искажение сигнала расширяет реальную полосу пропускания сигнала DTMF немного дальше за теоретическое значение 1477 Гц.

Очистка сигнала — введение в фильтры

Мы все знакомы с общим понятием фильтра: он удаляет то, что нам не нужно, из того, что нам нужно. Фильтры кофе, пропускающие кофейную жидкость, но задерживающие гущу, и воздушные фильтры, пропускающие чистый воздух, но улавливающие пыль и другие загрязнители, — два типичных примера механических фильтров в повседневной жизни. Эта же идея применима и к зашумленным электрическим сигналам: необходимо пропускать "истинный" сигнал, предоставляющий интерес, и блокировать нежелательный сигнал шума.

Глядя на рис. 2.12, представим, что интересующий нас сигнал находится в низкочастотной области, а сигнал шума — в высокочастотной. В идеальном случае нам необходимо избавиться от этого высокочастотного шума, оставив только ту составляющую, которая нас интересует. Для этого представим следующий процесс. К частотной области применяется маска, которая пропускает все составляющие низкочастотного сигнала, никак на них не влияя, но обнуляет все высокочастотные компоненты шума. Графически такая маска могла бы выглядеть, как частотный спектр, представленный на рис. 2.13.

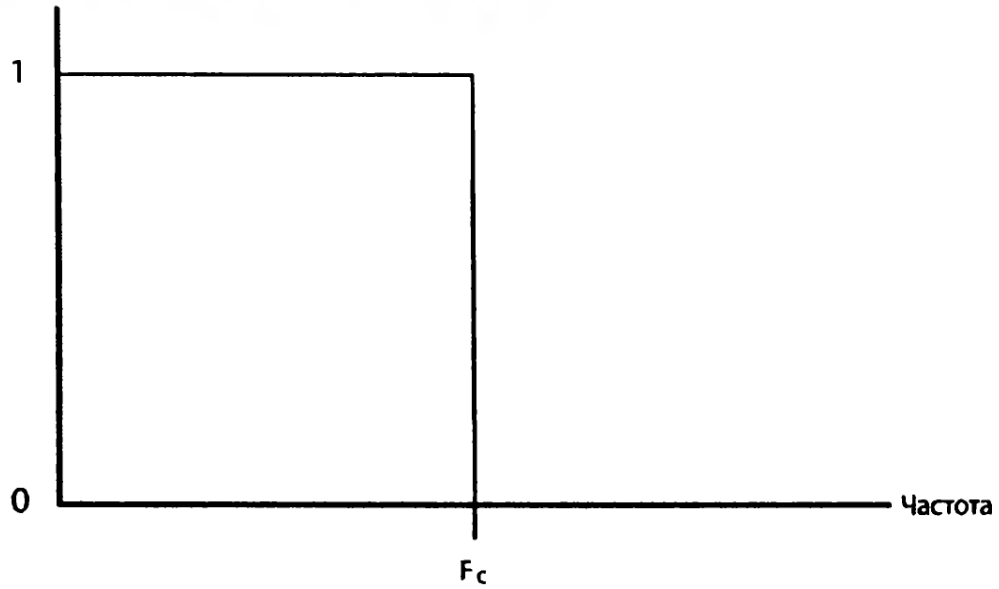


***Рис. 2.13.** Пример маски для фильтра нижних частот*

Если умножить значение в каждой точке графика на рис. 2.12 на значение в соответствующей точке графика маски на рис. 2.13, то будет

получен частотный спектр, показанный на рис. 2.14, — именно то, что нам нужно.

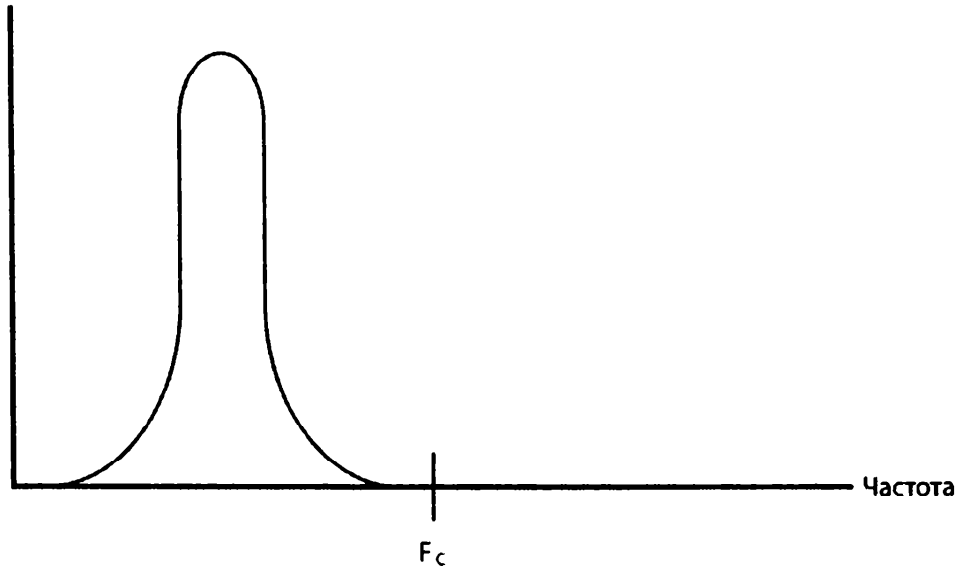


Рис. 2.14. *Результат умножения маски рис.2.13 на спектр на рис. 2.12*

Теоретически все просто, но возможно ли реализовать это на практике? Ответ: "Да", — хотя и с некоторыми важными оговорками, обусловленными различиями в поведении реальной и идеализированной систем. Впрочем, прежде, чем ознакомиться с этими оговорками, рассмотрим важное фундаментальное понятие дискретизации.

Дискретизация аналогового сигнала

Сигналы датчиков по своей сути — аналоговые, т.е. непрерывные во времени и по значению. К сожалению, обработка аналоговых сигналов "в чистом виде" требует специальных электронных схем, которые зачастую сложны в разработке, дорогостоящие и подвержены сдвигу рабочих характеристик со временем, поскольку свойства их компонентов изменяются ввиду старения. Намного лучше преобразовать входной аналоговый сигнал в цифровое значение, которое затем можно обрабатывать с помощью микропроцессора. Эту методику называют *аналого-цифровым преобразованием* или *дискретизацией*.

На рис. 2.15 показан пример непрерывного во времени сигнал напряжения, а на рис. 2.16 — дискретная версия этого сигнала. Одна из ключевых концепций, которая иногда смущает тех, кто впервые сталкивается с дискретными сигналами, заключается в том, что такой сигнал — это просто последовательность числовых значений, каждое из которых соответствует уровню непрерывного сигнала в определенный момент времени.

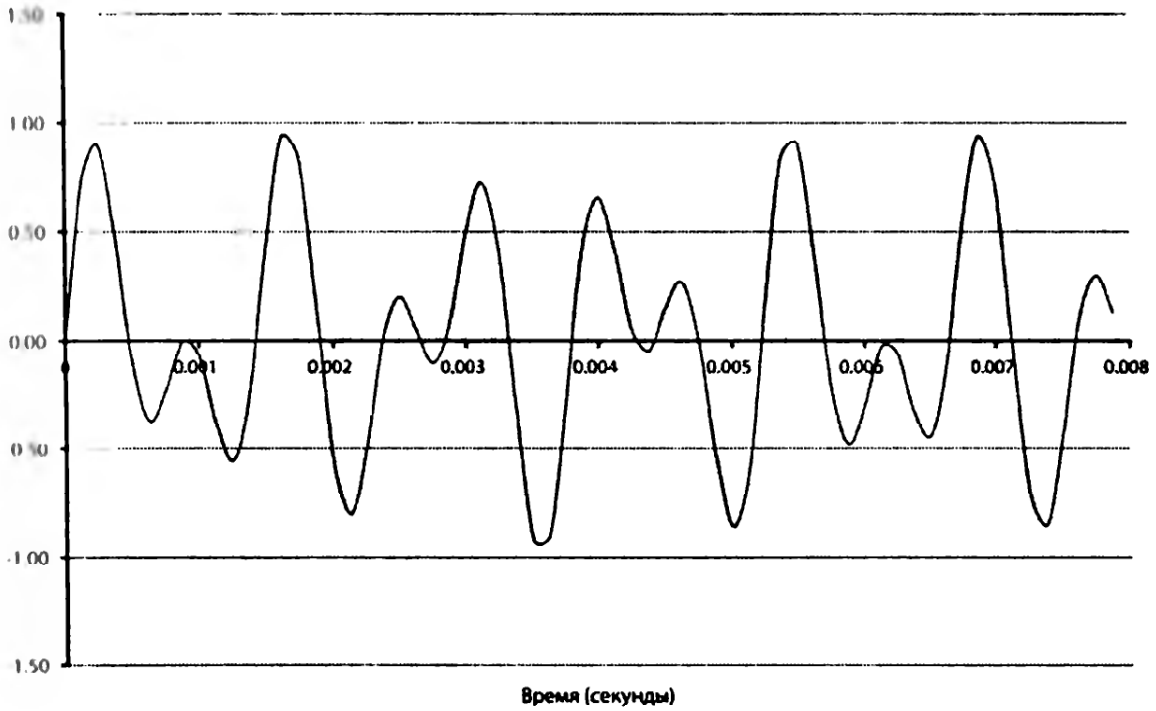


Рис. 2.15. Пример непрерывного во времени сигнала напряжения

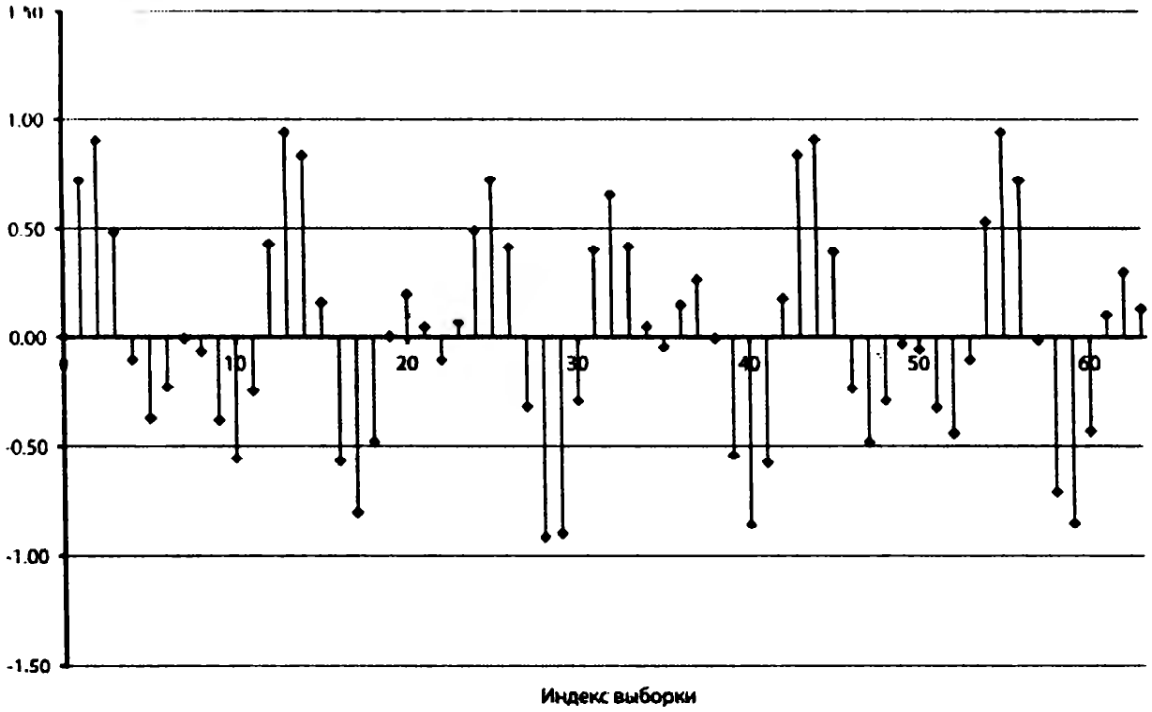


***Рис. 2.16.** Версия сигнала на рис. 2.15 в дискретном представлении*

Для дискретного сигнала (например, показанного на рис. 2.16), сигнал существует только в точке выборки. Он не является нулевым между точками выборки, однако соглашение по графическому представлению

дискретных данных требует отображать выборочные значения на линии (или сетке), где ось X обозначает параметр, используемый для определения момента выборки данных (как правило — время или пространственный промежуток).

Еще одно соглашение предполагает размещение выборочных значений в ряд, используя *индексные номера*. По этой схеме первая выборка сигнала $x(t)$ обозначается как x_0, вторая — как x_1 и т.д. Если мы складываем два сигнала x и y, тогда результирующий сигнал z — это просто сложение выборок:

$$z_0 = x_0 + y_0$$

$$z_1 = x_1 + y_1$$

$$z_N = x_N + y_N$$

Дискретизация оказывает на сигнал два важных воздействия. Первое из них известно под названием *спектральной репликации*. Это означает, что частотный спектр дискретного сигнала повторяется в частотной области с периодом, равным частоте дискретизации.

На рис. 2.17 показан пример частотного спектра аналогового сигнала, а на рис 2.18 — частотный спектр соответствующего дискретного сигнала.

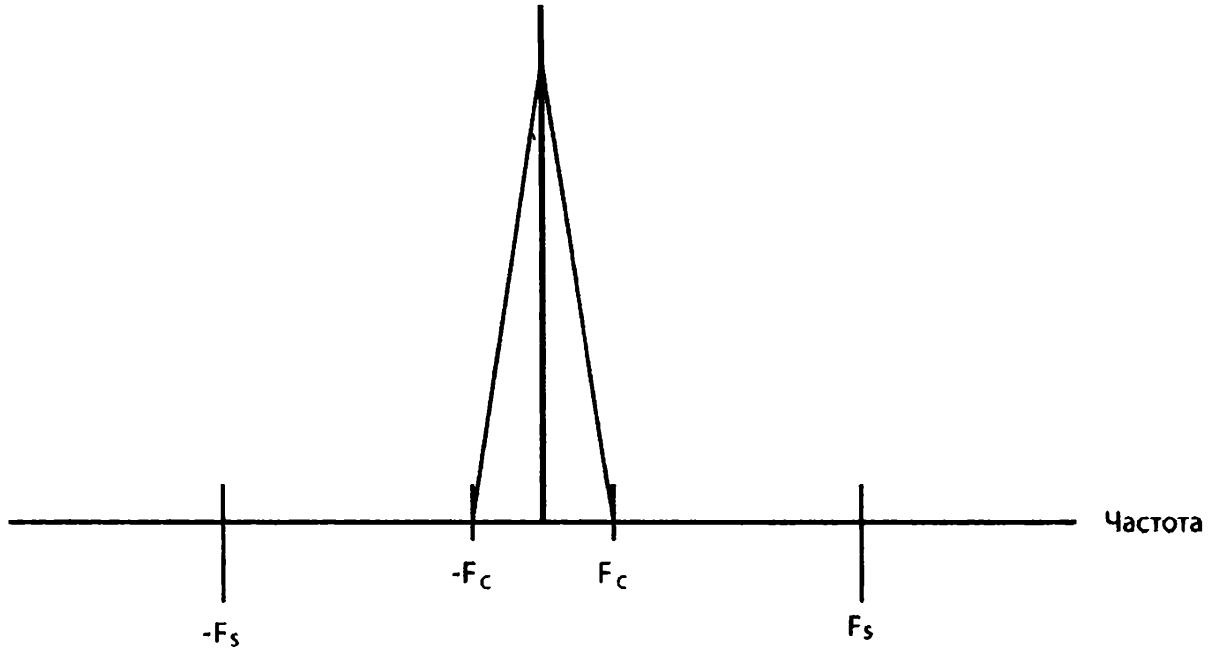


Рис. 2.17. *Пример частотного спектра аналогового сигнала*

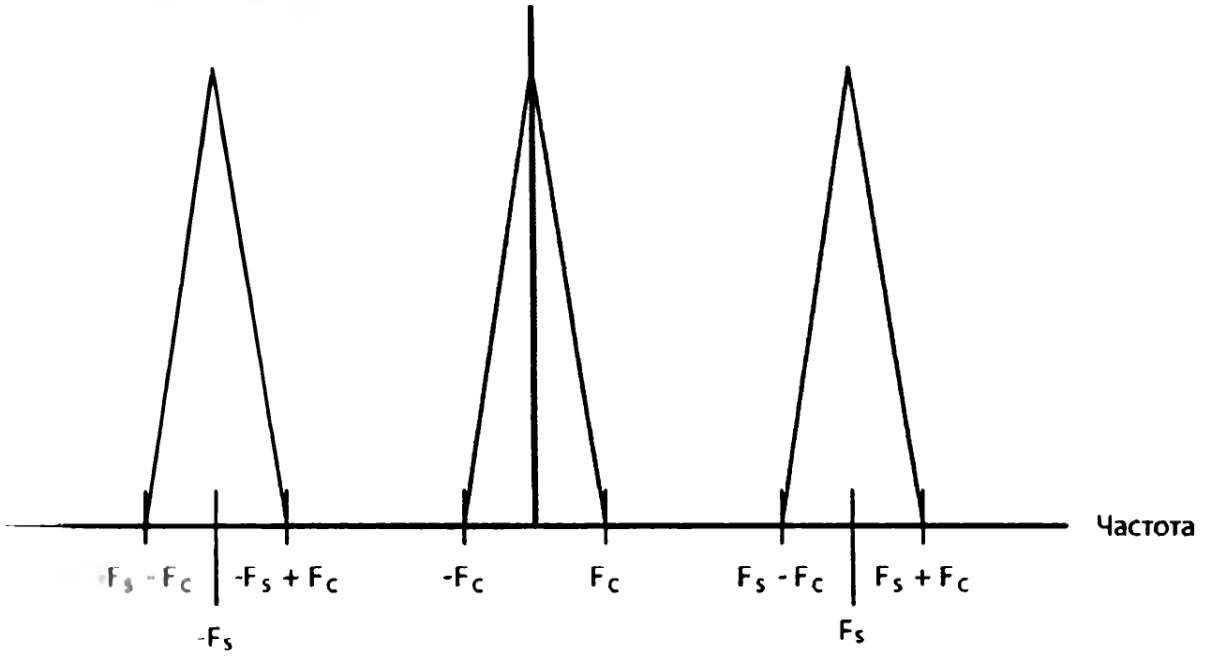


Рис. 2.18. Соответствующий частотный спектр дискретного сигнала

Как можно легко увидеть, проблема возникает, когда наивысшая частота спектра в исходном сигнале больше удвоенной частоты дискретизации — так называемой *частоты Найквиста*. В таком случае частотные составляющие реплицированного спектра частично перекрывают друг друга — состояние, известное как *наложение*, поскольку некоторые высокочастотные компоненты в спектре неотличимы от низкочастотных в ближайшем реплицированном спектре. Обычно присутствие наложения в системе считается нежелательным, и хотя полностью устранить его в реальном мире невозможно, снижение его влияния до незначительного уровня — задача, вполне осуществимая.

Рассмотрим простой пример, иллюстрирующий, каким образом наложение может создать обманчивое впечатление, что сигнал ведет себя определенным образом, хотя в действительности он ведет себя совершенно иначе. Представьте себе, что мы периодически фиксируем положение солнца в течение дня на протяжении длительного времени. Будучи хорошими учеными, мы хотим удостовериться, что частота дискретизации эффективна, и потому организовываем параллельно два измерения с разными частотами дискретизации. Результат первой группы измерений при частоте дискретизации, равной одной выборке за 1,5 часа, показан на рис. 2.19. Как и следовало ожидать, измерения показывают, что солнце в ходе эксперимента продвигается с востока на запад.

Взглянем на результаты второй группы измерений, для которой частота дискретизации составляет одну выборку за 22,5 часа (рис. 2.20).

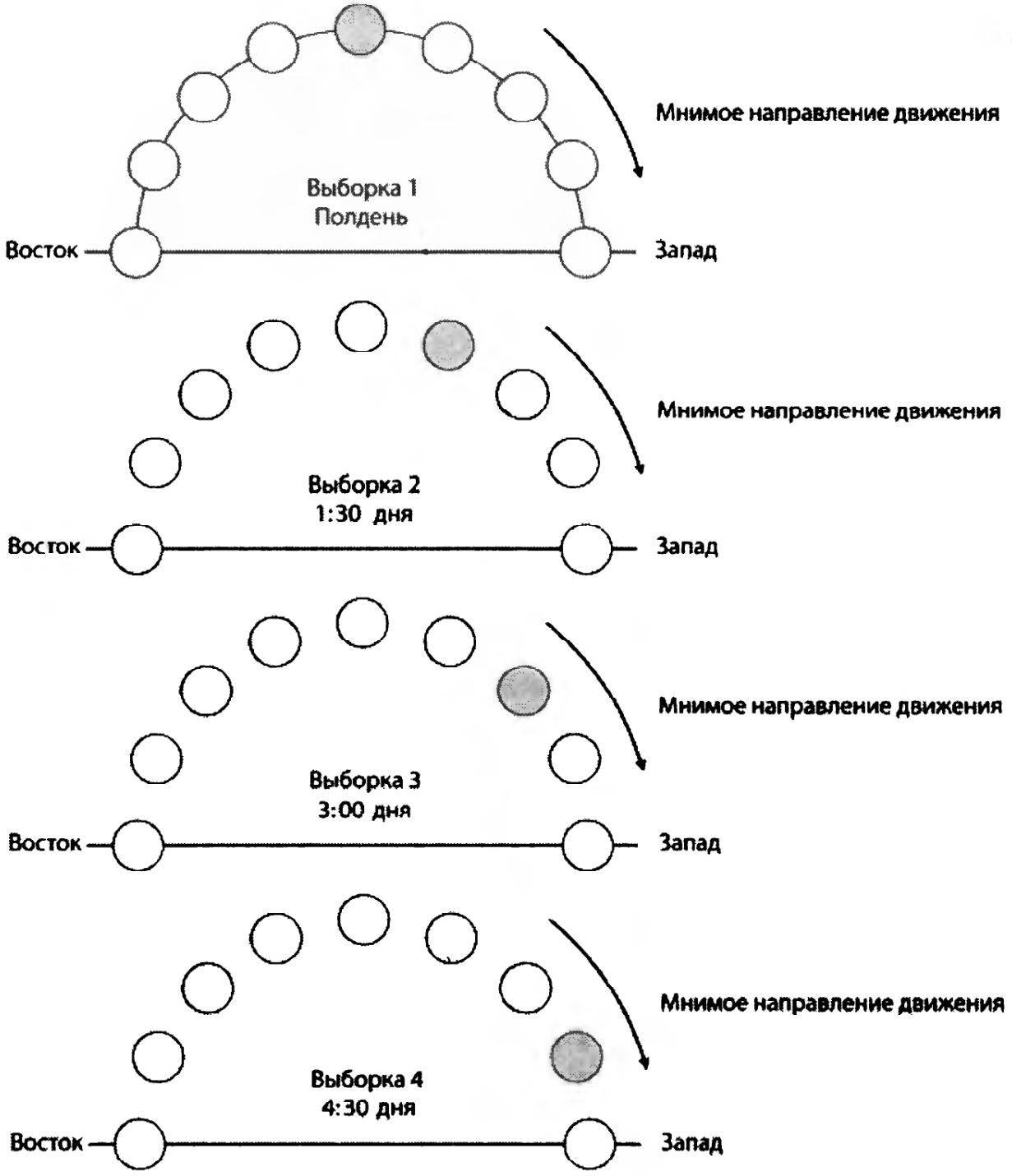


Рис. 2.19. *Положение солнца фиксируется каждые 1,5 часа*

По этим данным кажется, что солнце движется с запада на восток, что никак не соответствует истине! В этом и заключается ошибка, обусловленная наложением: характеристики сигнала выглядят не такими, какие они есть на самом деле.

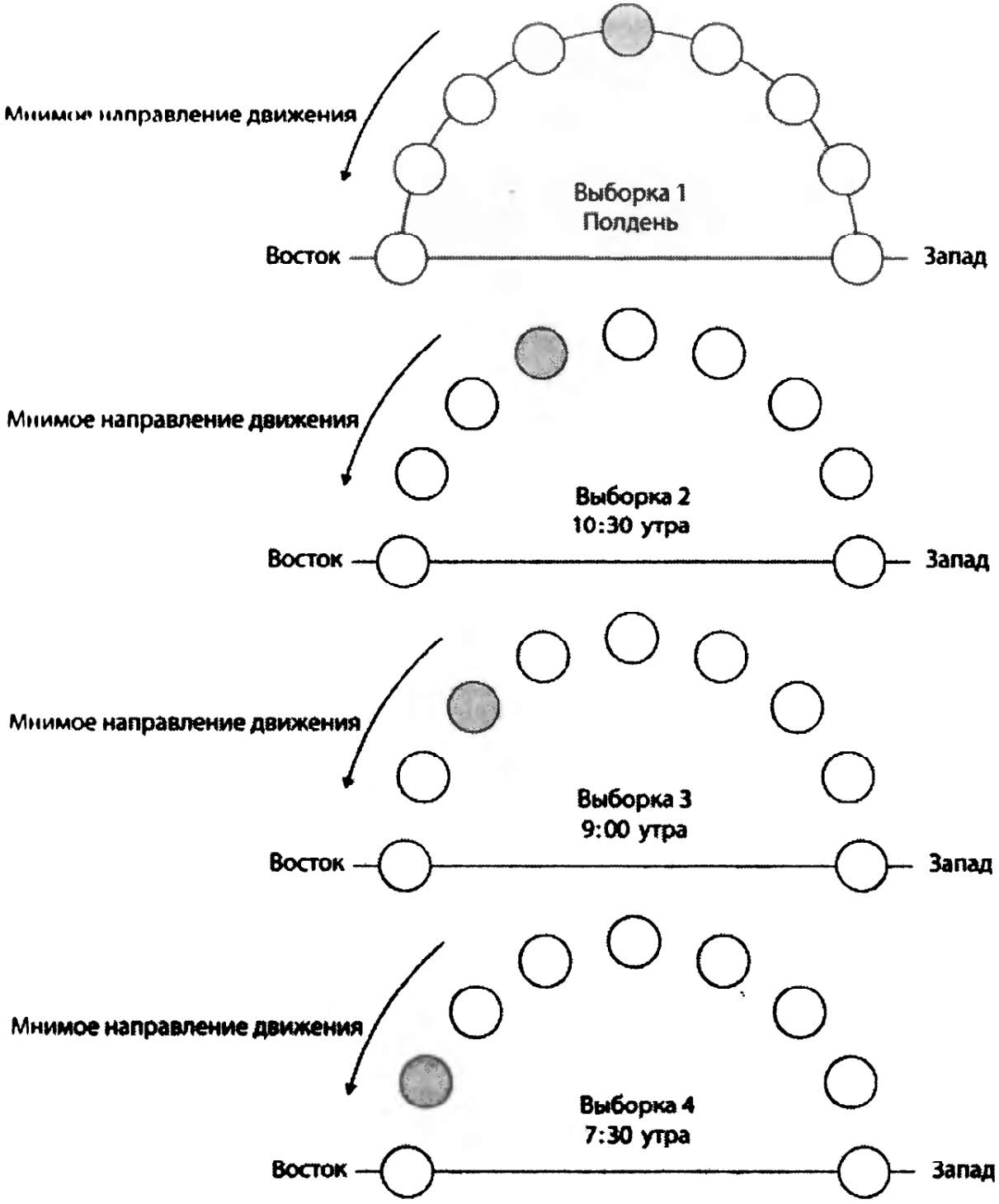


Рис. 2.20. Положение солнца фиксируется каждые 22,5 часа

Фильтры нижних частот

Мы уже рассматривали пример фильтра нижних частот (ФНЧ), который пропускает низкие частоты и блокирует высокие частоты. Идеализированный пример фильтра нижних частот показан на рис. 2.21.

В нем полоса частот (диапазон частот пропускаемых составляющих сигнала) равна 1 500 Гц. Заметьте, что пропускная способность в этом случае также равна 1 500 Гц, поскольку это — наибольшая частотная составляющая фильтра.

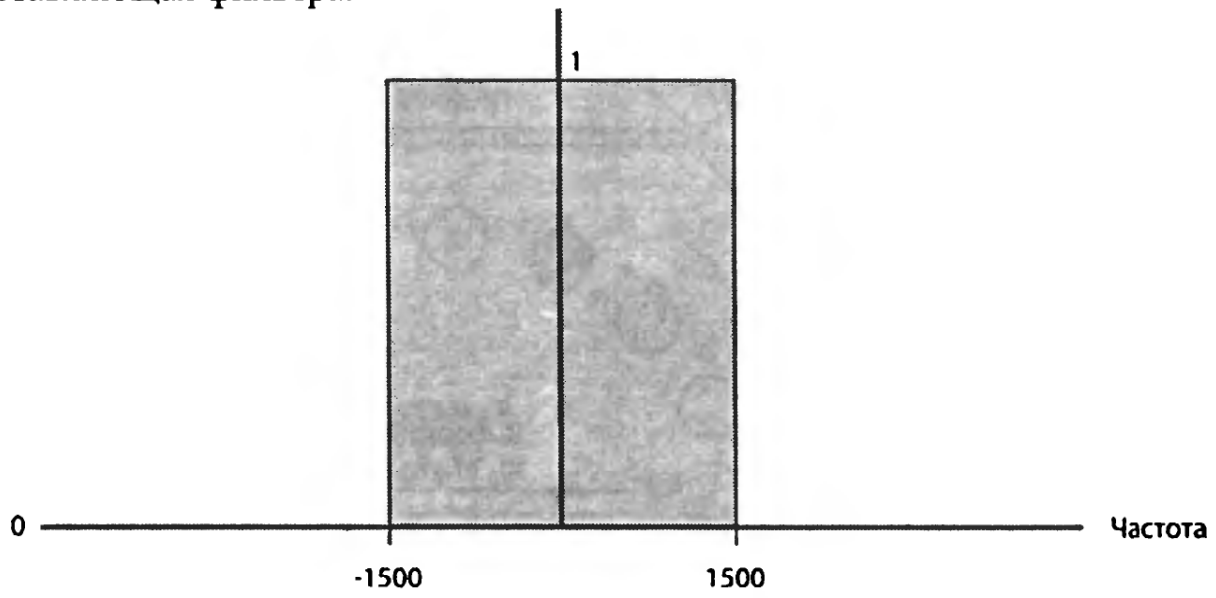


Рис. 2.21. *Идеализированный фильтр нижних частот с полосой пропускания 1 500 Гц*

Фильтр нижних частот — это, наверное, самый распространенный тип фильтров по той простой причине, что в реальном мире мы не сталкиваемся с сигналами, имеющими бесконечную полосу пропускания. В определенный момент времени частотный спектр сигнала понижается до незначительного уровня, поэтому одной из самых распространенных методик шумоподавления является задание некоторой границы для частотных составляющих, которые считаются полезными, и обрезка всех частот выше этой границы. Например, когда мы используем термопары для измерения температуры, напряжение термопары может изменяться только с определенной скоростью и не быстрее, поскольку температура физического тела изменяется с конечной скоростью (т.е. температура не может изменяться скачком). На практике это означает, что частотные составляющие сигнала температуры имеют верхнюю границу, за которой энергия сигнала незначительна. Если мы проектируем фильтр нижних частот, подавляющий все частоты выше верхней границы, то знаем, что он устраняет только шум, поскольку выше этой частоты отсечки составляющая сигнала отсутствует.

Фильтры верхних частот

Дополнением к фильтру нижних частот является *фильтр верхних частот* (ФВЧ), который пропускает только высокочастотные состав-

ающие сигнала, блокируя низкочастотные. В идеализированном фильтре верхних частот (рис. 2.22), полоса частот начинается с 1 500 Гц и продолжается до бесконечности. Обратите внимание на то, что в данном случае полоса пропускания бесконечна, поскольку в фильтр включены все частоты, начиная с нижней границы полосы частот.

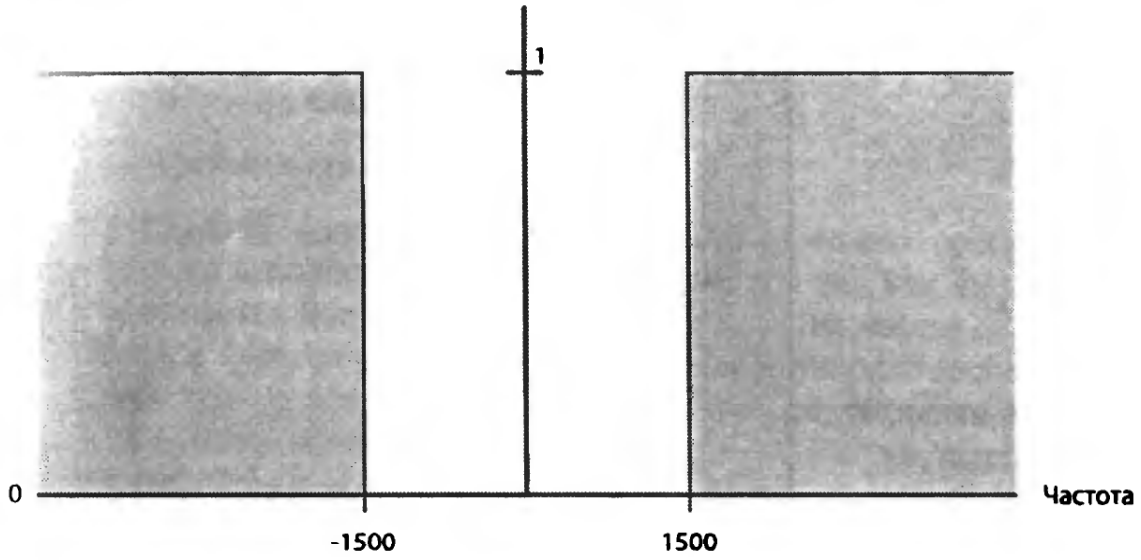


***Рис. 2.22.** Идеализированный фильтр верхних частот с полосой частот, начинающейся с 1 500 Гц*

Мы только что отметили, что никакой сигнал в реальном мире не имеет бесконечную полосу пропускания, зачем же нам фильтр, который противоречит этому утверждению? В некоторых случаях сигнал, который мы измеряем, имеет природу переменного тока. Согласно природе системы, все, что находится ниже определенной частоты, — это, очевидно, шум, поскольку ниже этой частоты отсутствует значимая часть сигнала. В качестве примера можно привести человеческое ухо, которое чувствительно только к частотам из диапазона от 20 Гц до приблизительно 20 кГц. Все, что находится ниже 20 Гц, не представляет практической ценности и может расцениваться как шум.

Полосовые фильтры

Полосовой фильтр — это, по сути, комбинация фильтра верхних частот и фильтра нижних частот, в котором полоса частот ФВЧ начинается с частоты, расположенной ниже, чем полоса пропускания ФНЧ (рис. 2.23). Здесь мы видим, что фильтр будет пропускать частоты в диапазоне 750–1 500 Гц, блокируя все остальные.

Полосовые фильтры применяются в тех случаях, когда разработчику необходим только определенный диапазон частот. Очень распро-

страненный пример этого — узел настройки на частоту в радиоприемнике. В нем применяется полосовой фильтр с очень узкой полосой частот, чтобы выделить сигнал отдельной радиостанции, ослабив сигналы всех остальных станций до такого уровня, котором они не прослушиваются.

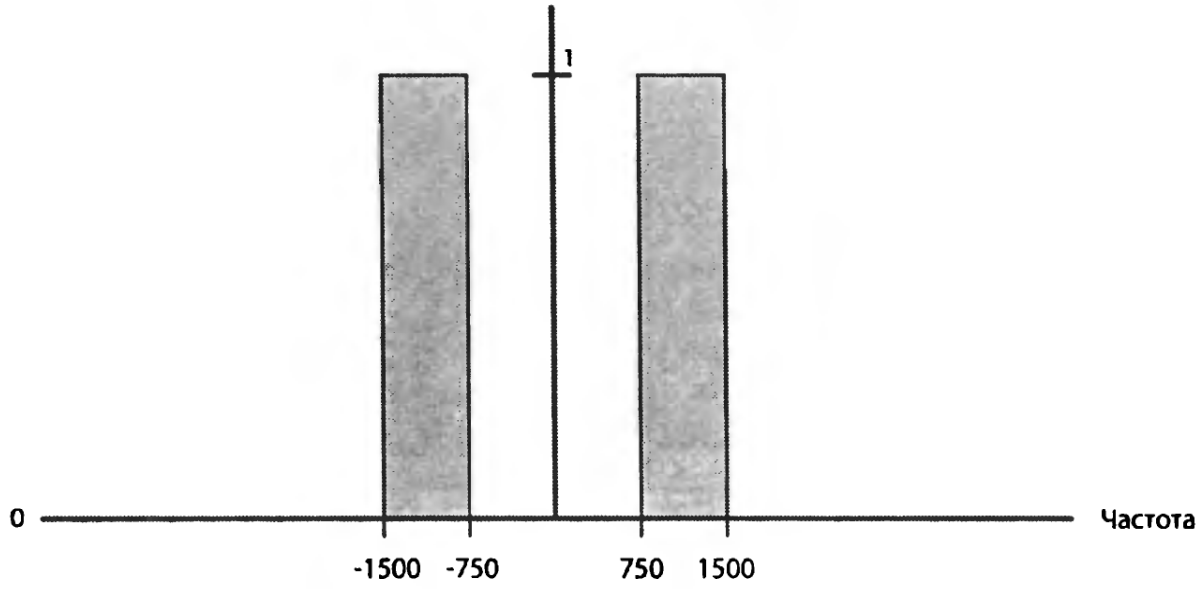


Рис. 2.23. Идеализированный полосовой фильтр

Еще одна распространенная сфера применения полосовых фильтров — наблюдение за уровнем сигнала в определенной полосе частот. Детекторы DTMF-сигналов используют этот принцип для различения нажатия кнопок на телефоне с кнопочным набором. В DTMF-системах каждой кнопке соответствует комбинацией только двух частот, не имеющих общих гармоник. Такая пара частот всегда содержит одну компоненту из четырех значений нижних частот, и другую — из четырех значений верхних частот (табл. 2.1).

Таблица 2.1. Тоновые DTMF-комбинации

	1 209 Гц	1 336 Гц	1 477 Гц	1 633 Гц
697 Гц	1	2	3	A
770 Гц	4	5	6	B
852 Гц	7	8	9	C
941 Гц	*	0	#	D

В основном, для того чтобы быть действительным сигналом DTMF, отклонение каждой из двух частотных составляющих не должна превышать 1,5 % от номинального значения, а разность амплитуд этих двух компонент должна составлять менее 3 дБ. Применяя полосовой фильтр

для каждой из восьми частотных составляющих плюс один для всей полосы пропускания сигнала, детектор может проанализировать выходные данные каждого фильтра, чтобы определить следующее:

- в любой момент времени активны только две частотных составляющих;
- они образуют правильную комбинацию (одна — из группы нижних частот, и одна — из группы верхних частот);
- соотношение их значений допустимо.

Заграждающий фильтр

Заграждающий (или режекторный) фильтр можно рассматривать как дополнение к полосовому фильтру точно так же, как фильтр верхних частот является дополнением к фильтру нижних частот. В то время как полосовые фильтры пропускают частоты в узкой полосе, заграждающие фильтры подавляют частоты в этой полосе, оставляя остальные частоты практически неизмененными. Пример заграждающего фильтра показан на рис. 2.24.

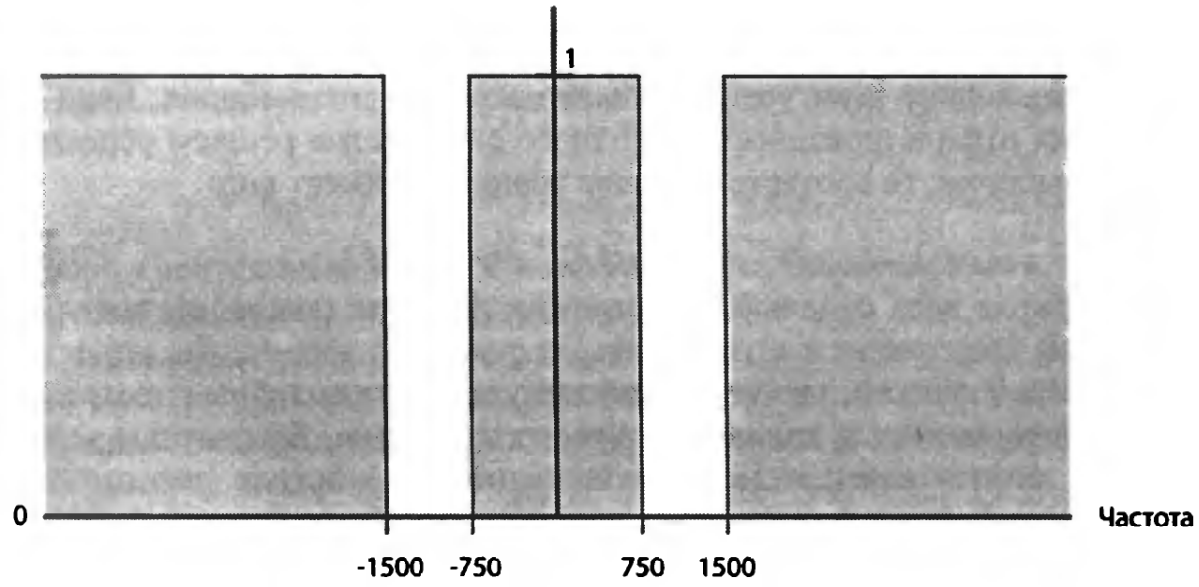


Рис. 2.24. Идеализированный заграждающий фильтр

Наверное, основным способом применения заграждающих фильтров является ослабление шума в линиях электропередачи, сосредоточенного в области 50 Гц или 60 Гц (в зависимости от типа сети). Во многих приложениях сигнал из сети с частотой 50 Гц или 60 Гц проникает в измерительную схему и, к сожалению, его частота зачастую находится среди частотного спектра интересующего сигнала. Простой ФНЧ или ФВЧ, который исключил бы все частоты, соответственно, выше или ниже частоты сигнала питания, слишком сильно ослабил бы по-

лезный сигнал. В результате разработчики пытаются удалять только частотные составляющие в узком диапазоне, включающем частоту электросети.

Реализации цифровых фильтров

До этого момента наше исследование фильтров строилось вокруг концептуальных направлений, теперь же мы переходим к фактической математической реализации этих фильтров. В основном, цифровые фильтры создаются путем применения весовых коэффициентов к одному или нескольким значениям дискретных данных с последующим суммированием этих взвешенных значений. Например, если у нас есть дискретный входной сигнал x_i, где $i = 0, 1, \ldots, N-1$, мы можем сформировать отфильтрованный выходной сигнал y_i:

$$y_i = a_0x_0 + a_1x_1 + \ldots + a_{N-1}x_{N-1} \qquad (2.3)$$

Здесь a_i — постоянные весовые значения, которые применяются к соответствующим дискретным значениям x_i входного сигнала. Пример ФНЧ — усредняющий фильтр, на выходе которого получают простое среднее значение заданного числа выборок. Этот фильтр сглаживает сигнал, поскольку шум усредняется по всей группе выборок. Если для получения отфильтрованного выходного сигнала мы решаем усреднить четыре выборки, то соответствующее уравнение имеет вид:

$$y_i = ¼\, x_0 + ¼\, x_1 + ¼\, x_2 + ¼\, x_3 \qquad (2.4)$$

Подбирая веса отдельных слагаемых фильтра (значений выборок), мы можем подстраивать его характеристику. Для того чтобы упростить жизнь разработчикам, ряд компаний выпускает специальные программы для проектирования и анализа цифровых фильтров, бесплатные версии которых можно найти в Internet. В наших разработках для создания и анализа цифровых фильтров мы воспользуемся программой dsPIC Filter Design™ от компании Microchip.

Предыдущий пример иллюстрирует структуру так называемого *фильтра с конечной импульсной характеристикой* (или *КИХ-фильтра*). Фильтры, сконструированные в соответствии с этим подходом, всегда имеют фиксированное число слагаемых уравнения и, таким образом, их выходная характеристика зависит только от ограниченного числа выборок на входе. Если мы пропускаем импульсный сигнал, показанный на рис. 2.25, через фильтр с числом слагаемых N, то знаем, что выходной сигнал по отношению к входному сигналу затухнет после N выборок, поскольку все последующие входные значения будут нулевыми.

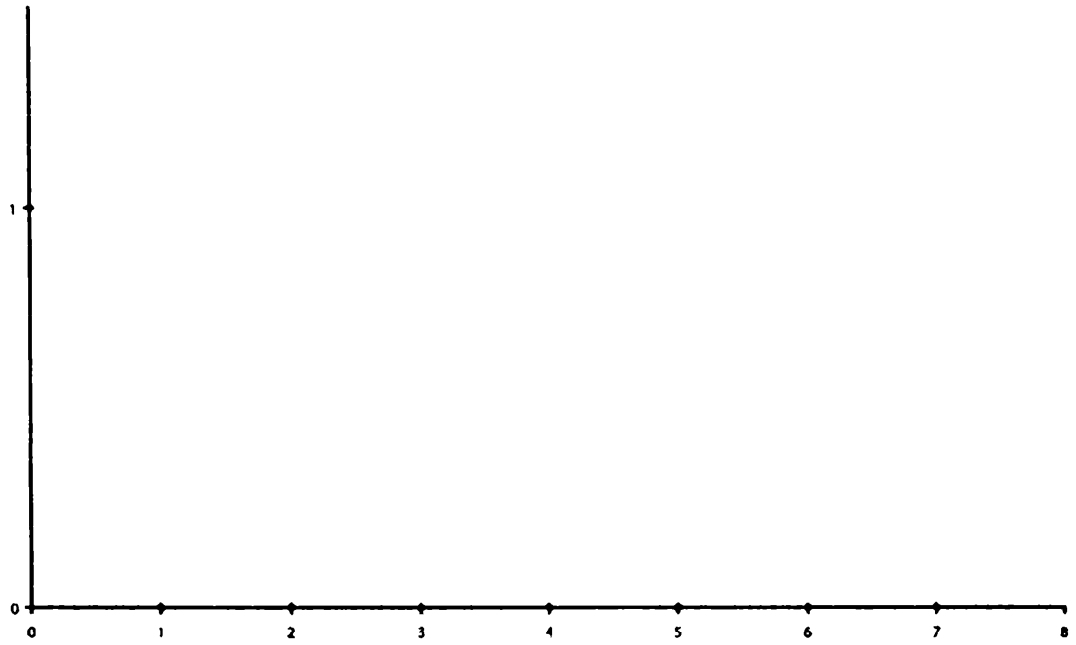


Рис. 2.25. *Единичный импульсный сигнал*

Еще одна структура фильтра — с *бесконечной импульсной характеристикой* (*БИХ-фильтр*). Такие фильтры для формирования выходного сигнала используют как взвешенные входные, так и взвешенные выходные выборки:

$$y_i = b_0 y_{i-1} + b_1 y_{i-2} + \ldots + b_{i-K} y_{i-K-1} + a_0 x_0 + \ldots + a_{N-1} x_{N-1} \qquad (2.5)$$

где b_i — постоянные весовые коэффициенты, применяемые к соответствующим значениям составляющим y_{i-1}.

На первый взгляд может показаться, что мы усложнили фильтр, однако это не обязательно так. Например, функцию для усредняющего фильтр из четырех слагаемых, который мы исследовали, рассматривая КИХ-фильтр, можно было бы реализовать в следующем виде:

$$y_i = y_{i-1} + ¼\, x_0 - ¼\, x_4 \qquad (2.6)$$

Хотя сокращение вычислительных потребностей для реализации уравнения (2.6) по сравнению с уравнением (2.4) невелико, в более сложных фильтрах применение БИХ-фильтров может дать значительную экономию ресурсов процессора и памяти. Однако такое сокращение имеет свою цену. В отличие от КИХ-фильтра, БИХ-фильтр теоретически может реагировать на входной сигнал всегда (отсюда и название структуры), что может быть крайне нежелательно. Разработчики также должны следить за тем, чтобы не накапливались ошибки, иначе характеристики схемы могут деградировать до такой степени, что фильтр уже не принесет пользы.

Медианные фильтры

Все фильтры, рассматриваемые до сих пор, базируются на простых математических уравнениях, так что их поведение легко проанализировать с помощью общеизвестных, понятных методик. Эти фильтры оптимально подходят для обработки шума, содержащегося в специфичном спектре, что обычно является вполне подходящей моделью. Однако иногда системы восприимчивы к так называемому *дробовому* или *импульсному* шуму, при котором измеренный сигнал содержит шум в виде пиков, а не непрерывного сигнала. Для противодействия такому шуму в системах можно реализовать еще один вид фильтрации, известный как *медианная фильтрация*. Она несколько более эвристическая, однако превосходно подходит для ослабления дробового шума.

В медианном фильтре сигнал дискретизируется, как и в других типах фильтров, однако выборки не просто подвергаются математическим операциям, а упорядочиваются в порядке убывания (или наоборот, что в действительности не имеет значения), после чего определяется средняя или *медианная* выборка. Если продолжительность медианного фильтра превышает продолжительность импульсного шума, то шумовые сигналы будут полностью устранены.

Пример семикомпонентного медианного фильтра и его воздействие на сигнал, искаженный дробовым шумом продолжительностью максимум в три выборки, показан на рис. 2.26–2.29.

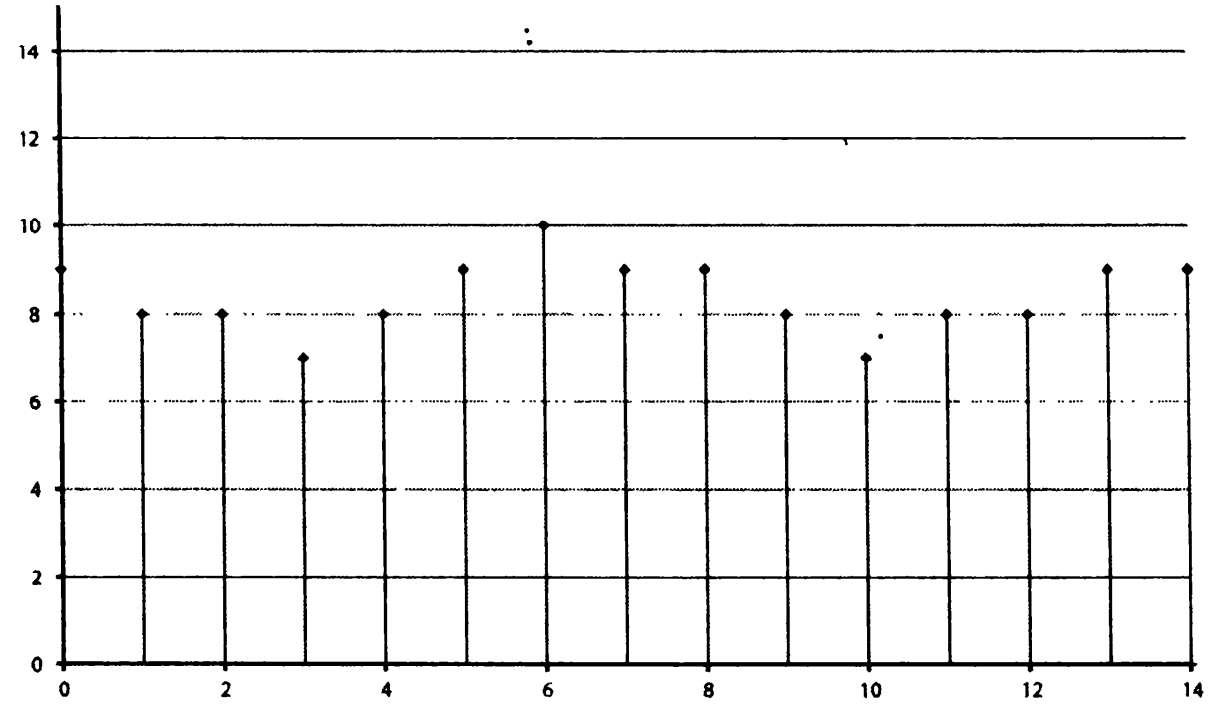


Рис. 2.26. *Образец "истинного" сигнала без шума*

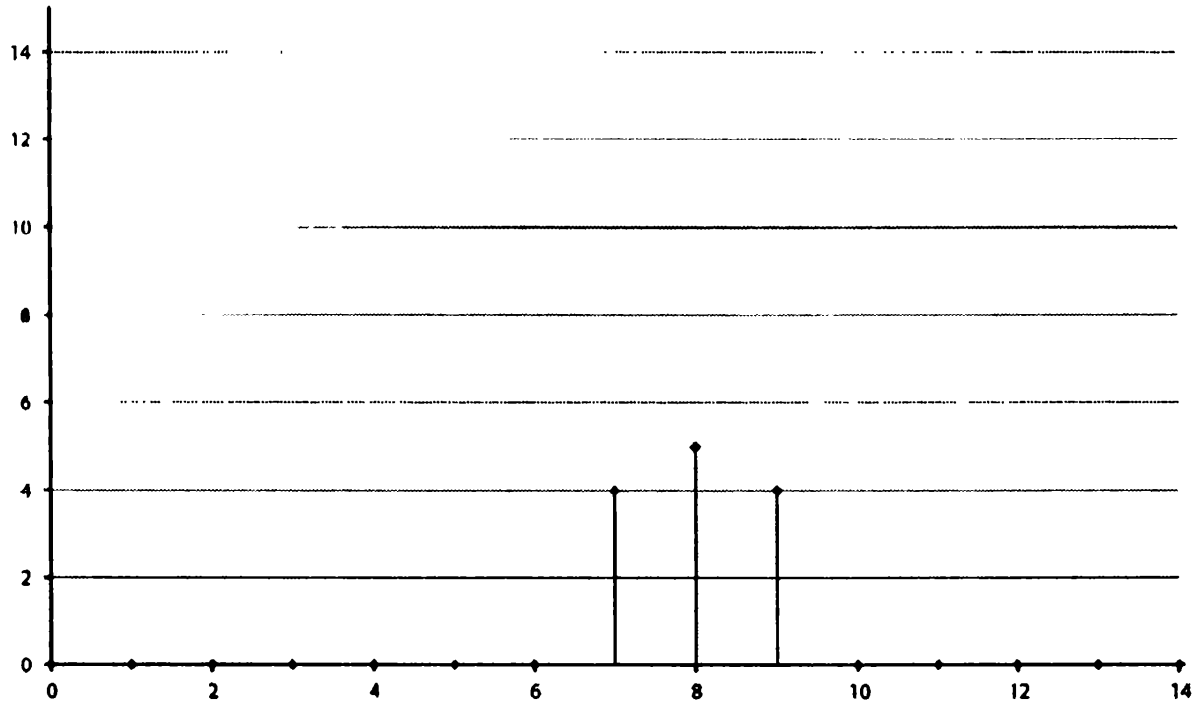


Рис. 2.27. *Образец дробового шума продолжительностью в три выборки*

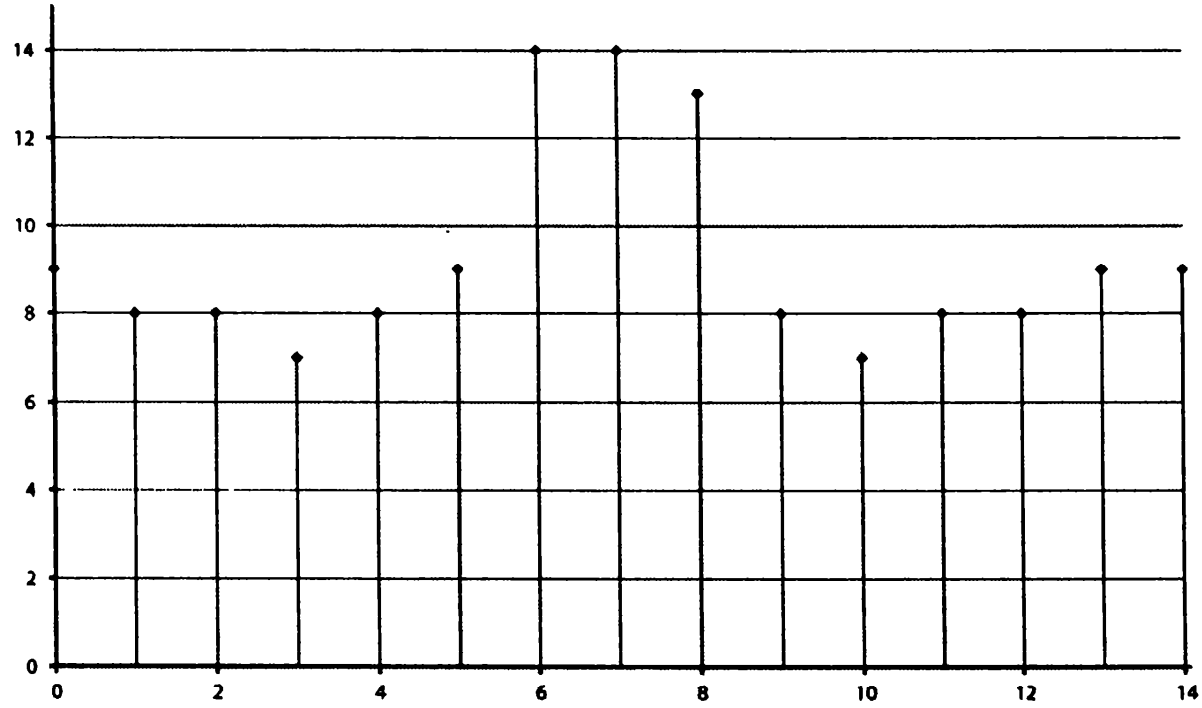


Рис. 2.28. *Измеренный "истинный" сигнал с дробовым шумом*

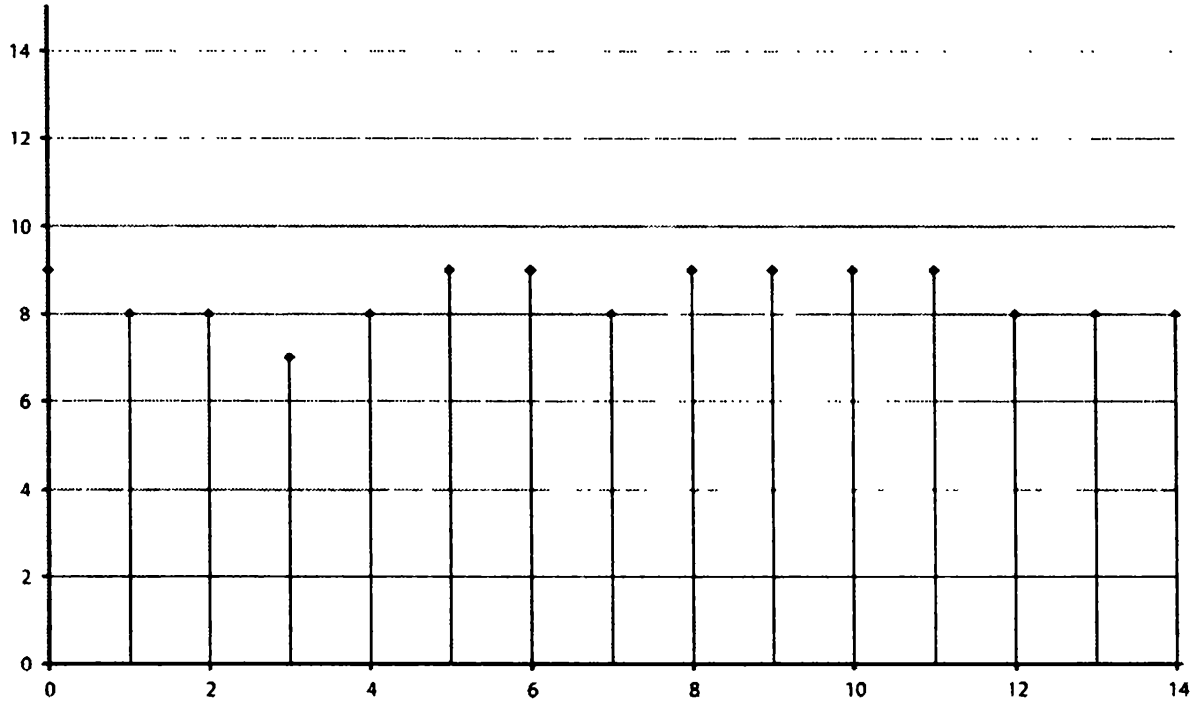


Рис. 2.29. *Результирующий сигнал после обработки с помощью семикомпонентного медианного фильтра*

2.2. Дискретизация сигналов

Мы уже коснулись одной проблемы, которая может возникать при дискретизации аналогового сигнала, а именно — наложения. Среди других проблемных вопросов, связанных с дискретизации сигналов, рассмотрим еще три: влияние оцифровки, влияние ограниченного размера регистра и дискретизацию с запасом по частоте.

Влияние оцифровки на дискретный сигнал

До сих пор мы предполагали, что все измеряемые нами сигналы — это непрерывные аналоговые значения, т.е. что наши измерения совершенно точны. Даже в случае присутствия шума мы исходили из предпосылки, что само по себе измерение (например, зашумленное выходное напряжение датчика) известно в точности. В действительности же (по крайней мере для систем с цифровой обработкой сигналов), это не так, поскольку измеренные аналоговые сигналы проходят через так называемую *оцифровку* — преобразование аналогового сигнала в соответствующие цифровые значения, над которыми можно выполнять математические операции в процессоре. Это иллюстрирует рис. 2.30 (выборка значений сигнала была выполнена в указанных точках).

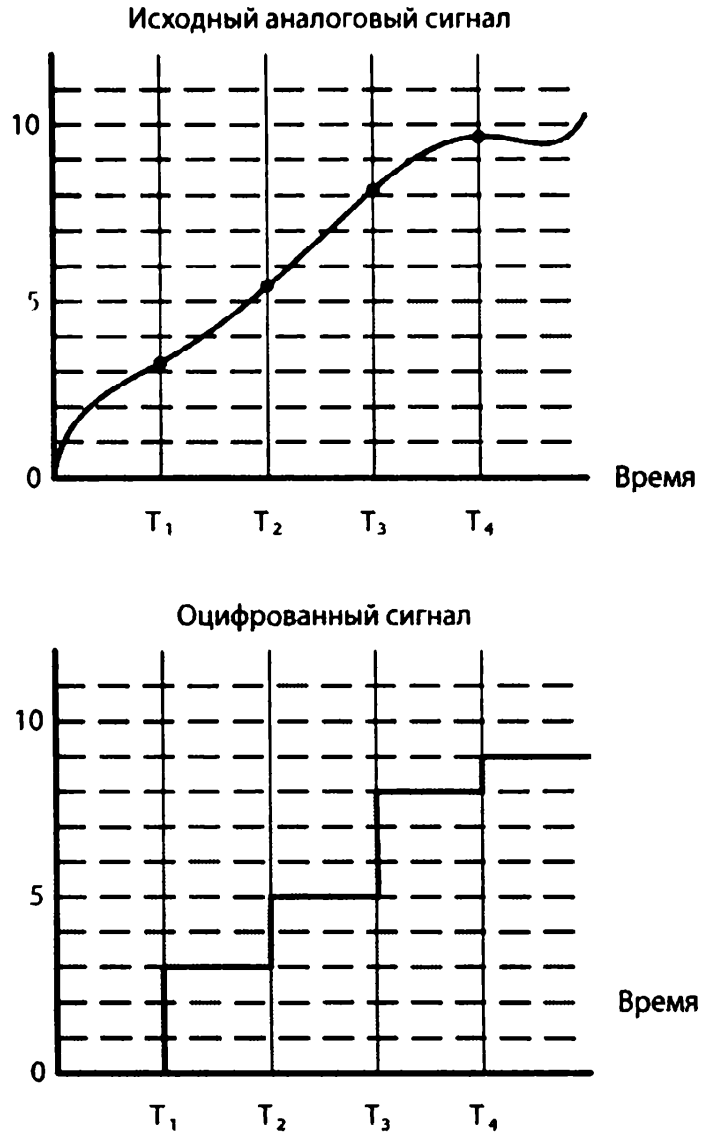
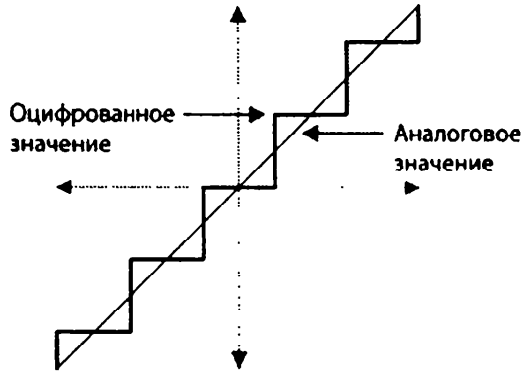


Рис. 2.30. *Процесс оцифровки сигнала, состоящий из четырех последовательных выборок*

Проблема, с которой мы сталкиваемся при оцифровке, заключается в том, что любой процессор имеет в распоряжении ограниченное число разрядов для представления измеренного сигнала. Предположим, необходимо дискретизировать сигнал, изменяющийся в диапазоне 0–5 В. Если попытаться представить измерение одним разрядом, то мы получим в распоряжение только два возможных значения: 0 и 1. Обозначив измеренный сигнал напряжения как V_S, мы могли бы сопоставить нижней половине диапазона сигнала ($0 \leq V_S < 2{,}5$ В) значение 0, а верхней ($2{,}5 \leq V_S < 5$ В) — значение 1. К сожалению, такая разрешающая способность слишком мала! Хотя мы и можем существенно повысить разрешающую способность, отведя больше разрядов для представления числовых зна-

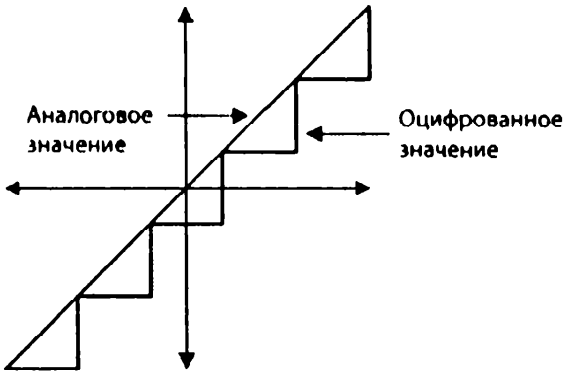
чений, мы всегда будем сопоставлять определенный диапазон входных значений с единственным выходным значением. Это означает, что почти все измеренные значения сигнала, лежащие в этом диапазоне, будут с ошибкой (единственное исключение из этого — случай, когда значение сигнала в точности соответствует цифровому значению).

Ошибка оцифровки может рассматриваться как сигнал шума, который накладывается на истинное значение измеренного сигнала (рис. 2.31 и рис. 2.32).



Когда оцифрованный сигнал округляется, ошибка равномерно распределяется вокруг "истинного" аналогового значения с максимальной абсолютной ошибкой ½ от одного интервала дискретизации

Рис. 2.31. *Погрешность оцифровки, вносимая округлением*



Когда оцифрованный сигнал усекается, ошибка имеет такой же ступенчатый эффект, однако ее абсолютное значение теперь лежит в диапазоне от 0 до одного уровня дискретизации

Рис. 2.32. *Погрешность оцифровки, вносимая усечением*

Заметьте, что в зависимости от того, выполняем ли мы оцифровку округлением (см. рис. 2.31) или усечением (см. рис. 2.32) измеренного значения, мы, по сути, получаем треугольный (при округлении) или пилообразный (при усечении) сигнал шума. Хотя полностью устранить этот недостаток невозможно, можно уменьшить его влияние, задействовав для представления числовых значений в алгоритмах относительно большое число разрядов (скажем, от 16 до 32 — в зависимости от приложения). Например, в случае использования 16-разрядных значений, сигналы можно представлять с точностью 0,0015% (если предположить, что отсутствуют любые другие источники шума оцифровки). В случае с 32-разрядными значениями эта разрешающая способность улучшается до $2{,}3 \times 10^{-8}$ % (поскольку присутствует 2^{32} дискретных уровня).

Влияние ограничений в размере регистра

Ограничения в размере регистра относятся к проблемным вопросам, которые связаны с выполнением повторяющихся математических опе-

раций над значениями, представленными конечным числом разрядов. Все это имеет непосредственное отношение к ошибке оцифровки, вносимой по той причине, что для представления непрерывного сигнала существует ограниченный набор числовых значений. Дело в том, что повторяющиеся математические операции могут вызывать переполнение в накапливающем сумматоре процессора. Проиллюстрируем это на простом примере.

Предположим, приложение оцифровывает входной сигнал в 16-разрядные значения (0–FFFFh), а мы используем процессор с 16-разрядным накапливающим сумматором. Если мы пытаемся усреднить две выборки, лежащие на ¾ диапазона дискретизации (BFFFh), то в случае, если бы у нашего регистра была бесконечная точность (1/2 (BFFFh + BFFFh) = BFFFh), мы получили бы значение BFFFh. Однако при сложении двух выборок в 16-разрядном накапливающем сумматоре сумма будет равна не 17FFEh, а 7FFEh, поскольку старший разряд будет отброшен (для него в регистре нет места). Если затем разделить эту сумму пополам, то получится среднее значение 3FFFh вместо корректного BFFFh.

Хотя мы можем использовать регистры бо́льшей разрядности, эту проблему устранить до конца невозможно — особенно, если используются операции умножения. Разработчики могут ослаблять влияние ограничения в разрядности регистров, выбирая подходящие способы для внутреннего представления числовых значений и внимательно отслеживая случаи переполнения и потери значащих разрядов, однако этот вопрос всегда должен быть под контролем во избежание катастрофических системных сбоев. В следующей главе будет показано, каким образом с подобными проблемами справляется процессор dsPIC.

Дискретизация с запасом по частоте

Как было упомянуто ранее, разработчики во избежание наложения должны убедиться в том, что системы, применяющие ЦОС, осуществляют выборку входных сигналов с частотой, превышающей частоту Найквиста (частота, которая в два раза больше наибольшей частотной составляющей входного сигнала). Тем не менее, на практике частота дискретизации входных сигналов должна превышать наибольшую частотную составляющую входного сигнала по крайней мере в пять раз, чтобы учесть различия между реальными и идеальными характеристиками аналого-цифрового преобразования. Этого отодвигает дискретные спектры друг от друга, сводя их наложение к минимуму.

Еще одна проблема, связанная с любым фильтром, — это задержка между поступлением сигнала в фильтр, и моментом его выхода из

фильтра (и это справедливо как для цифровых, так и для аналоговых фильтров). В основном, чем сильнее фильтруется входной сигнал, тем больше задержка его прохождения через систему. Если задержка становится чрезмерной (что зависит от приложения), отфильтрованный сигнал на выходе может утратить свою ценность, поскольку появляется слишком поздно и становится бесполезным для остальной части системы.

Дискретизация с запасом по частоте заключается в дискретизации сигнала намного быстрее, чем это требуется. Это позволяет выполнять сильную фильтрацию без внесения чрезмерной задержки прохождения через систему. Поскольку в таком случае задержка умеренна, а частота дискретизации намного превышает необходимую, отфильтрованный сигнал своевременно поступает в другие компоненты системы. Недостаток этого подхода заключается в том, что для обработки более интенсивного потока данных требуется больше вычислительной мощи, что обычно повышает стоимость системы и потребляемую ею мощность.

2.3. Как анализировать прикладную задачу, в которой применен датчик

Анализируя отдельную прикладную задачу, связанную с обработкой сигнала датчика, разработчикам необходимо знать следующие аспекты системы:

- физическое свойство, подлежащее измерению;
- взаимосвязь между измеряем физическим свойством и фиксируемым значением параметра;
- ожидаемый частотный спектр интересующего сигнала и возможных вероятных источников шума окружающей среды;
- физические характеристики рабочей среды;
- условия возникновения ошибок и соответствующие методики их предотвращения;
- требования по калибровке,
- требования к пользовательскому и/или системному интерфейсу;
- требования по обслуживанию.

Зачастую природа измеряемого физического параметра и рабочей среды помогает разработчику разобраться в выборе подходящих средств обработки сигнала, которые необходимо включить в систему. Например, в случае измерения температуры большой массы металла, нагреваемой относительно небольшим нагревательным элементом, можно уверенно предположить, что частотный спектр интересующего сигна-

ла — минимален, поскольку температура может меняться только постепенно. Это означает, что для уменьшения шума к сигналу датчика может применяться сильная фильтрация. В противоположность этому, датчик, следящий за температурой маленького устройства, нагреваемого лазером, должен быть в состоянии реагировать на интенсивные изменения температуры, которые могут происходить очень быстро. В такой ситуации степень фильтрации шума должна быть меньше, и может потребоваться дополнительная обработка, нацеленная на подавление шума, проникающего сквозь начальные фильтры.

Также крайне важно понимать взаимосвязь между измеряемым физическим свойством и соответствующим параметром, предоставляемым пользователю или остальной части системы. Изменяется ли такой параметр линейно в зависимости от физического свойства (как в случае RTD-датчиков температуры), или же имеет нелинейную зависимость (как во многих термопарах)? Если зависимость нелинейная, то возможно ли разбить ее на кусочно-линейные области, чтобы упростить вычисление? Недостаточное или неверное понимание зависимости между физическим свойством и сообщаемым параметром может сделать сенсорную систему бесполезной.

Наконец, разработчики должны всегда учитывать, что сенсорные системы будут работать в реальном мире, где проблемы возникают в самые неподходящие моменты. Система должна обнаруживать типичные ошибки, и чем надежней обнаружение ошибок и схема их обработки, тем лучше. Выход из строя датчика на производственной площадке может остановить всю линию, так что конечные пользователи высоко ценят любые свойства, позволяющие быстро выявлять и легко устранять неполадки. Еще более ценной является способность системы датчиков обнаруживать опасные состояния, которые могут привести к некорректной работе. При проектировании датчиков, работающих в помехоустойчивой среде, следует уделять пристальное внимание обнаружению неисправностей, а также их протоколированию и устранению.

2.4. Обобщенная структура процесса обработки сигнала от датчика

Теперь мы готовы составить обобщенную схему процесса обработки сигнала от датчика, который будем пользоваться в каждом из примеров, рассмотренных в главах 4–7. Как и все хорошие разработки, эта схема (рис. 2.33) обманчиво проста. Самое главное — ее надежная реализация, чтобы она выполняла все поставленные перед ней задачи точно, вовремя и в любой момент.

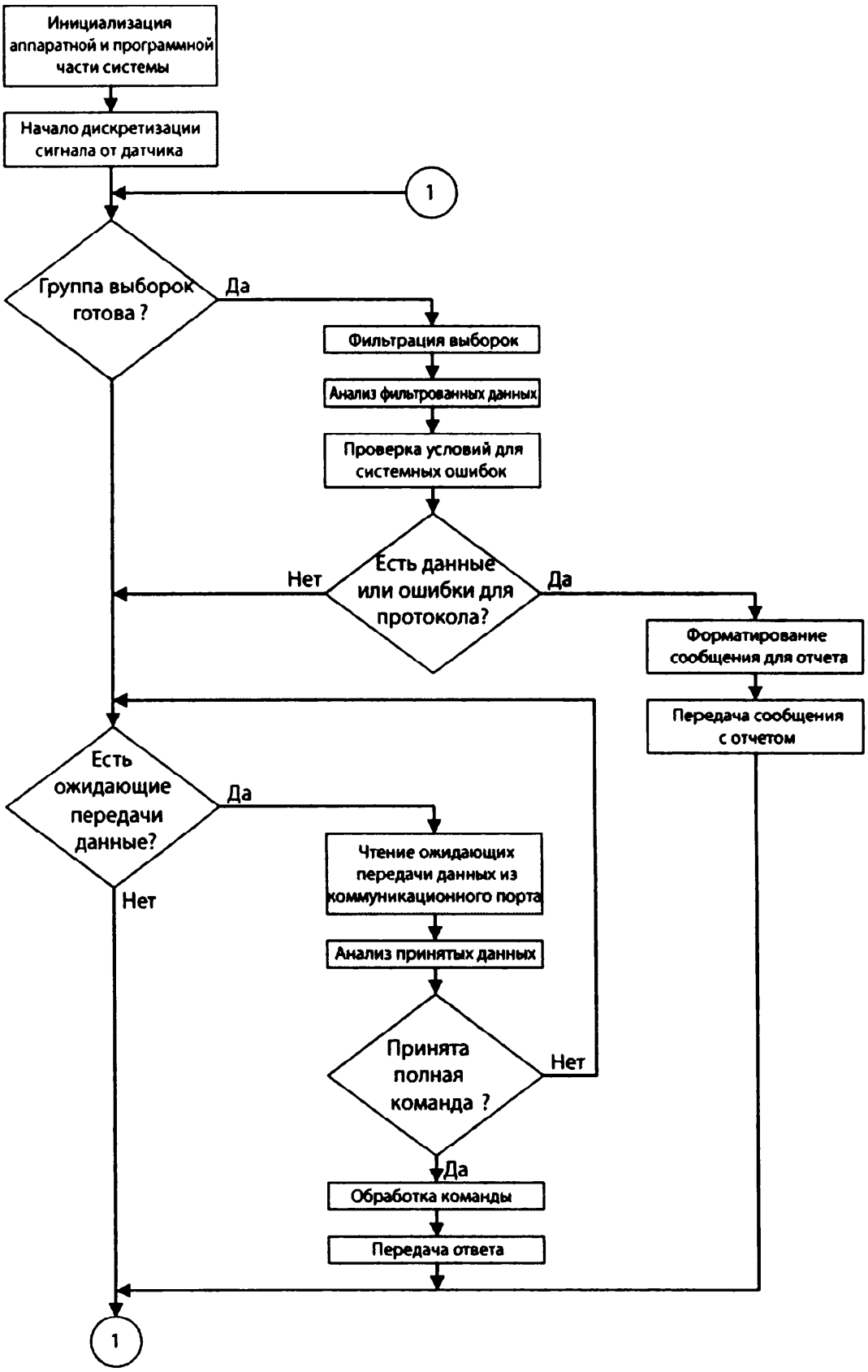


Рис. 2.33. *Обобщенная схема обработки сигнала от датчика*

Структура должна представлять собой систему *жесткой привязки к реальному времени*, т.е. ее отклик на входные данные и события должен быть детерминированным (наступать в фиксированные моменты времени), а вся обработка входных данных или событий должна заканчиваться до поступления очередных входных данных или событий (по крайней мере, в критических по времени секциях обработки). Менее критичные секции (например, протокол обмена данными) также важны, однако не обязаны быть жестко привязаны к реальному времени. Они должны быть в состоянии обрабатывать все входные данные или события со временем, однако могут помещать эти входные данные или события в очередь для обработки в удобное для приложения время.

Формирование и обнаружение сигнала

Секция формирования и обнаружения сигнала отвечает за выполнение любого требуемого формирования аналогового входного сигнала для того, чтобы:

- ограничить его частотный спектр такой полосой, которая может успешно обрабатываться;
- усилить уровень сигнала до степени, пригодной для оцифровки;
- оцифровать результирующий входной аналоговый сигнал.

На выходе этой секции получают поток дискретных данных, которые могут быть обработаны в виде чисел остальной частью системы.

Фильтрация перед анализом

После дискретизации "сырого" сигнала физического свойства зачастую требуется применить к нему специфичную для сферы применения фильтрацию, чтобы устранить ненужный шум или придать сигналу более приемлемую форму. Эта фильтрация обычно выполняется сразу же после обнаружения сигнала, чтобы алгоритмы обработки, реализованные далее в цепи прохождения сигнала, имели возможность использовать относительно "чистые" данные для получения лучших результатов.

Линеаризация сигнала

Иногда интересующий параметр не изменяется линейно по отношению к измеряемому физическому свойству. Простой пример этого — сигнал термопары, который имеет сложную полиномиальную зависимость напряжения от температуры. В таких случаях сигнал зачастую необходимо линеаризовать, чтобы с ним было проще работать в секции

параметрического анализа. Специфика применяемой методики линеаризации варьируется в зависимости от измеряемого свойства.

Анализ параметра

Анализ параметра также очень специфичен для каждой сферы применения. Хотя анализ из ограничивается только воображением разработчика, среди типичных операций можно назвать преобразование параметра (при котором измеренный сигнал математически преобразуется в желаемое соответствующее значение параметра), частотный анализ и проверка на сравнение с пределами. Зачастую, это — наиболее сложный аспект сенсорной системы, и та сфера, которая может придать изделию наибольшую ценность.

Фильтрация после анализа

После вычисления значения параметра оно нередко фильтруется с целью сглаживания для использования другими составными частями системы. Как и в случае фильтрации перед анализом, выбор конкретного типа фильтра зависит от сферы применения.

Обнаружение ошибок и их обработка

В то время как секция анализа параметра, в основном, определяет ценность системы, секция обнаружения и обработки ошибок может или повысить жизнеспособность системы или загубить ее. Умение обнаруживать ошибки и устранять их может дать изделию конкурентное преимущество — особенно в ситуациях, когда последствия отказа могут быть катастрофичными. Простое обнаружение ошибки может включать в себя проверку на наличие чувствительного элемента и проверку извлекаемых значений параметра на принадлежность допустимому диапазону. Более совершенное обнаружение ошибок может реализовать диагностику с целью оповещения пользователя до фактического возникновения отказа.

Обмен данными

Последний элемент структуры — секция обмена данными, которая протоколирует всю информацию, собранную интеллектуальным датчиком, и позволяет пользователю конфигурировать его работу. Таким образом, крайне важно, чтобы этот интерфейс был надежным и устойчивым в работе. Существует множество различных интерфейсов обмена данными: от RS-232 и Control Area Network (CAN) до Ethernet и беспроводных технологий. Впрочем не все системы поддерживают все типы

интерфейсов. Разработчик должен выбрать такой интерфейс, который обеспечивает самую простую интеграцию продукта с другими элементами системы, оставаясь при этом в рамках бюджета и требований к надежности, определенных для конкретной сферы применения.

2.5. Резюме

В этой главе мы рассмотрели основные понятия цифровой обработки сигналов на концептуальном уровне. Теперь читатель должен осознавать, что доскональное знание ЦОС — бесценно для разработки надежных сенсорных систем. Впрочем, целью этой главы было привить интуитивное (а не исчерпывающее) понимание ЦОС. Тем не менее, это понимание дало нам возможность разработать обобщенную структуру цифрового анализа и протоколирования информации от датчиков, которую мы будем использовать в последующих главах для разработки сенсорных систем в специфических прикладных областях.

3

Возможности цифрового контроллера сигналов dsPIC

Успех складывается из мелочей. Было бы, наверное, неплохо оградить себя от деталей и заниматься лишь глобальными вопросами, однако... если уделять внимание только общему, не обращая внимания на мелочи, то большое становится маленьким, то есть бизнес усыхает.

— Генри Фаерстоун

Отправляясь в путешествие, транспортное средство выбирают, в основном, в зависимости от цели поездки и от того, что требуется для достижения этой цели. Хотя для доставки тысячи компьютеров в дистрибьюторский центр на другом конце страны теоретически можно было бы использовать спортивный автомобиль, все же для решения этой задачи эффективнее задействовать трейлер. Это же относится и к реализации интеллектуальных датчиков: вероятность достижения требуемых характеристик намного выше в случае использования аппаратной платформы, которая специально разработана для поддержки необходимых нам задач. Такая платформа должна предоставлять детерминированное[10] обнаружение, фильтрацию и анализ наблюдаемого сигнала, а также обеспечивать надежную связь со внешним миром. Во многих прикладных задачах эти требования предъявляются к системам, которые обрабатывают множество сигналов и, возможно, содержат несколько каналов связи, что еще больше повышает требуемый уровень производительности платформы.

---

[10] Требование для поведения *детерминированной* системы просто подразумевает, что структура гарантирует отклик системы на событие в течение определенного периода времени, которое должно быть достаточным для решения поставленных задач. Нет ничего хорошего, если система откликается на событие в течение трех секунд, а новые события поступают каждые три миллисекунды.

К счастью, недавно был разработан новый класс процессоров, известных как *цифровые контроллеры сигналов* (или ЦКС). Эти контроллеры объединяют в себе мощные математические возможности, присущие "чистым" цифровым процессорам сигналов, и в высшей мере детерминированное поведение стандартных микроконтроллеров. Один из таких ЦКС — это dsPIC® от компании Microchip. Он объединяет в себе множество функциональных возможностей на одном кристалле, позволяя разработчикам создавать надежные изделия в сфере измерений и управления в очень маленьком корпусе. Мы будем применять ЦКС dsPIC в последующих главах для решения типичных прикладных задач из сферы интеллектуальных измерений, но прежде исследуем ресурсы этой микросхемы. Изучение возможностей dsPIC преследует сразу две цели: узнать состав микросхемы и понять, почему разработчики ЦКС dsPIC реализовали именно такой набор средств (это поможет нам оптимизировать системы для наиболее эффективного использования ресурсов dsPIC).

Цифровой контроллера сигналов dsPIC можно рассматривать с разных точек зрения, однако тремя наиболее важными аспектами являются:

- архитектура обработки данных;
- математические представления и операции, поддерживаемые микросхемой;
- периферийные компоненты, встроенные на кристалл.

Четкое понимание этих трех вопросов послужит прочным фундаментом для создания эффективных систем с помощью dsPIC.

По большому счету, называть dsPIC "микросхемой" (в единственном числе) — некорректно, поскольку этот ЦКС, в действительности, представляет собой два родственных семейства микросхем со множеством конфигураций. Это позволяет разработчику подобрать такую комбинацию периферийных устройств, которая наилучшим образом подходит для конкретной задачи. По этой причине мы выберем для исследования конкретную микросхему dsPIC30F6014A, которая содержит, по сути, все периферийные устройства, встречающиеся у семейств dsPIC.

Хотя мы подробно рассмотрим все важные аспекты этой микросхемы, по ней существуют буквально тысячи страниц документации (одно только справочное руководство по программированию, предоставляемое компанией Microchip, содержит более 350 страниц). Таким образом, можно сказать, что мы сделаем всего лишь беглый обзор. При необходимости весь рассмотренный здесь материал можно найти в стандартной документации Microchip, однако в ней не всегда удается быстро найти то, что нужно.

Ценность этой главы не в том, что она открывает что-то новое, а, скорее, в том, что информация в ней преподнесена в удобной для работы форме. Изложенного здесь материала достаточно для того, чтобы использовать dsPIC со знанием дела. Таким образом, читатель получит прочный фундамент для более углубленного изучения свойств микросхемы, когда это потребуется.

В дальнейшем мы будем время от времени сталкиваться с печальными реалиями отличий в предполагаемом и реальном функционировании ЦКС dsPIC. Эти отклонения, публикуемые в списках под названием "Errata", могут вызывать у разработчика огромное чувство разочарования, поскольку изменяют поведение микросхемы по сравнению с документацией. С точки зрения разработчика, существует два типа отклонения:

- для которых существуют обходные пути (программные или аппаратные методики, устраняющие проблему);
- для которых обходных путей не существует.

Отклонения первого типа причиняют неудобства, однако не фатальны, а вот второй тип отклонений (особенно в тех сферах, которые необходимо для реализации задачи) может привести к существенной задержке процесса разработки, к повышению стоимости проекта или, в худшем случае, — к полному его срыву.

Учитывая такую возможность, проницательный разработчик всегда проверяет списки "Errata" для конкретной микросхемы и для всего семейства устройств в целом. В разных компаниях-производителях полупроводниковых приборов публикация информации об обнаруженных отклонениях организована по-разному, однако в компании Microchip сбор подобной информации налажен очень хорошо. Перечень проблем обычно всегда сопровождает техническую спецификацию изделия или справочное пособие по программированию, а также размещается на Web-сайте `www.microchip.com` вместе с технической информацией.

Сверившись со списком обнаруженных отклонений перед проектированием и в процессе отладки, разработчик может уберечь себя, свою компанию и клиентов от срывов планов, простоев и лишних затрат. Хотя, конечно, никогда не стоит сбрасывать со счетов (по крайней мере, на начальной стадии) возможные ошибки в коде программы или в аппаратной части проекта. Если же в этом смысле все гарантировано корректно, а микросхема не работает, как ожидается, то лучше всего исследовать список обнаруженных отклонений или обратиться за помощью к заводу-изготовителю.

3.1. Архитектура обработки данных в ЦКС dsPIC

Семейство устройств dsPIC разработано специально для высокоскоростного выполнения операций с интенсивными математическими вычислениями и оснащены специальными аппаратными элементами, управляющими интенсивными по времени процессами с минимальной нагрузкой на ядро процессора. Однако, как компоненты механического двигателя должны работать сообща по предписанным правилам для выполнения полезной работы, так и функционирование каждого отдельного элемента ЦКС dsPIC должно координироваться, иначе производительность устройства будет сильно отличаться от заложенного в нее потенциала. Тем не менее, для создания необходимой гармонии разработчик должен досконально понимать архитектуру микросхемы и входящих в ее состав элементов.

Память ЦКС dsPIC

Изучение ЦКС dsPIC начинается с архитектуры памяти, поскольку это — один из элементов, специально разработанных для выполнения требований по пропускной способности в системах цифровой обработки сигналов. Контроллер dsPIC построен по модифицированной гарвардской архитектуре, которая имеет раздельные шины памяти программ и памяти данных, позволяя процессору одновременно производить выборку как команды, так и данных, которыми эта команда будет оперировать.

В "чистой" гарвардской архитектуре эти две шины полностью разделены, и обмен данными между ними невозможен. Хотя такая архитектура вполне логична, она не подходит для потребностей многих задач ЦОС, поскольку подразумевает извлечение за один цикл единственного значения данных. Но в действительности зачастую требуется извлекать за один цикл два значения данных: значение выборки и коэффициент, на который оно умножается, — а в "чистой" гарвардской архитектуре на это потребуется два машинных цикла. Для того чтобы преодолеть это ограничение, модифицированная гарвардская архитектура поддерживает три шины: одну — для выборки команды (шина памяти программ) и две — для выборки связанных с командой значений данных (в литературе их часто называют шинами X и Y). Это позволяет реализовать настоящую одноцикловую операцию, тем самым удваивая пропускную способность системы по сравнению с "чистой" гарвардской архитектурой при той же тактовой частоте. Различие между "чистой" и модифицированной гарвардскими архитектурами показана на рис. 3.1.



Рис. 3.1. *Две разновидности гарвардской архитектуры*

Как мы упомянули в начале главы, ЦКС объединяют в себе вычислительную мощь цифровых процессоров сигналов (ЦПС) с детерминированным поведением микроконтроллера. Это сочетание отражается в небольших отличиях модели памяти ЦКС dsPIC при выполнении команд с интенсивными математическими вычислениями, известных как команды *умножения с накоплением* (*MAC* — Multiply And Accumulate), и команд других типов (иногда их называют командами микроконтроллеров). Таким образом ЦКС dsPIC может задействовать структуру памяти, наиболее подходящую для определенного типа команд. Команды, требующие нескольких источников данных для математического ядра, получают преимущество от использования двух шин данных, тогда как команды, которые не нуждаются в этом, могут рассматривать процессор как имеющий единое пространство данных. Аппаратные компоненты, используемые для доступа к памяти по этим двум структурам, рассматриваются в конце раздела 3.1 в подразделе "Режимы адресации и модули формирования адреса".

Следует отметить, что в ЦКС dsPIC не реализован один аспект, распространенный в "чистых" ЦПС и других высокопроизводительных микропроцессорах: многоступенчатая конвейерная обработка команд. Назначение конвейера — обеспечить для процессора внутреннюю оче-

редь команд, которые могли бы предварительно декодироваться для ускорения выполнения. Хотя этот подход успешно применяется в различных прикладных областях, где данные обрабатываются непрерывным потоком без прерываний, конвейеризация, в действительности, может уменьшить пропускную способность процессора, если системе приходится часто изменять ход обработки команд (это происходит в системах, работа которых основана на прерываниях). Когда изменения в последовательности обработки значительны (например, если процессор должен выполнять переходы к ячейкам памяти вне конвейера), конвейеру перед тем, как продолжить работу, приходится освобождаться от своего содержимого и загружаться новыми, корректными командами. Если опять провести сравнение с транспортными средствами, то это подобно управлению автомобилем на шоссе. Пока автомобиль двигается с предельной скоростью, он может быстро преодолеть большое расстояние, но если водителю приходится постоянно съезжать с шоссе и опять возвращаться на него, то для преодоления той же дистанции потребуется намного больше времени.

Хотя контроллеру dsPIC не достает многоступенчатого конвейера, в нем реализован механизм одноступенчатой предварительной выборки команд, который считывает и частично декодирует команды за один цикл до момента их выполнения. Это позволяет выполнять большинство команд за один цикл, существенно улучшая детерминированные временные характеристики системы.

Распределения памяти в пространстве данных

Память данных в dsPIC делится на три области: регистры специального назначения, статическое ОЗУ (SRAM) и память программ, которая отображается в пространстве памяти данных.

Регистры специального назначения

Контроллер dsPIC использует отображенные в памяти регистры для конфигурирования, управления и отслеживания различных аспектов своей работы. Эти регистры специального назначения располагаются в диапазоне 2 Кбайт младших адресов пространства данных. В отличие от стандартных ячеек памяти данных, которые используются в качестве хранилищ общего назначения, содержимое регистров специального назначения анализирует поразрядно. Таким образом, запись в определенные регистры специального назначения (или даже чтение из некоторых таких регистров) вызывает определенные действия ЦКС. Например, как мы увидим позже, передача байта данных через последовательный порт

микросхемы требует, чтобы приложение сконфигурировало рабочие параметры последовательного порта путем записи соответствующих данных в регистры специального назначения последовательного порта в определенной последовательности. Подобным же образом приложение может проверять, приняты ли новые данные, путем опроса регистра состояния последовательного порта — одного из регистров специального назначения периферийных устройств.

При работе с регистрами специального назначения следует проявлять крайнюю осторожность, поскольку чтение любого такого регистра или запись в него влияет на работу аппаратной части dsPIC и может привести к непредсказуемым последствиям.

Статическое ОЗУ

В контроллере dsPIC30F6014A статическое ОЗУ (SRAM — Static Random Access Memory) занимает диапазон адресов от 0x0800 до 0x27FF, что составляет 8 Кбайт памяти с произвольной выборкой. В отличие от регистров специального назначения, обмен данными с этой областью памяти никак не влияет на функционирование dsPIC, а только — на информацию, хранимую в соответствующих ячейках. Приложения используют сегмент SRAM для хранения данных, изменяющихся в ходе выполнения программы (например, переменных, используемых для фильтрации, или буферов, которые хранят принятые по коммуникационному интерфейсу данные). При определенных условиях, которые мы обсудим позже, память SRAM с целью повышения пропускной способности может быть разделена на две независимые области: X и Y. Начальный и конечный адрес этих двух областей аппаратно фиксирован для каждого устройства dsPIC. Так, в микросхеме dsPIC30F6014A область X-данных занимает в памяти диапазоне адресов от 0x0800 до 0x17FF, а область Y-данных — диапазон от 0x1800 до 0x27FF.

Память пространства программ, распределенная под память пространства данных

В некоторых алгоритмах (особенно в тех, которые используются для цифровой обработки сигналов) зачастую требуется перемножать фиксированные коэффициенты с данными переменных. Поскольку коэффициенты не изменяются, можно было бы сэкономить драгоценное пространство памяти SRAM, сохранив фиксированные значения в памяти программ, однако оставив их доступными для аппаратных средств обработки данных. К счастью, архитектура dsPIC поддерживает такую возможность, которая будет рассмотрено подробнее чуть позже.

Распределение памяти пространства программ

Поскольку конфигурации памяти в устройствах семейства dsPIC варьируется в соответствии со спецификой доступных ресурсов, за информацией о точном распределении памяти разработчик должен обращаться к соответствующим техническим описаниям. Представленное ниже распределение памяти относится к микросхеме dsPIC30F6014A, хотя во многом оно применимо и к другим представителям семейства dsPIC с незначительными отличиями в отдельных адресах.

Как было отмечено ранее, ЦКС dsPIC поддерживает адресное пространство как программ, так и данных, причем пространство данных разбито на пространства X и Y для определенных команд. Объем памяти пространства программ может составлять до 4 Мслов 24-разрядных команд, хотя не все это адресное пространство доступно пользователю. Сама память программ разбита на пространства пользовательской памяти (000000H–7FFFFEH) и конфигурационной памяти (800000H–FFFFFEH) (рис. 3.2).

Приложение в ходе выполнения может получить доступ исключительно к пространству пользовательской памяти, если только не используются команды `TBLRD` (табличное чтение) и `TBLWT` (табличная запись) для чтения/записи идентификатора устройства, идентификатора модуля и ряда разрядов конфигурации устройства, расположенных в конфигурационной памяти. Пространство конфигурационной памяти, главным образом, зарезервировано для использования компанией Microchip в целях тестирования и хранения некоторой полезной идентификационной информации об устройстве. Для сохранения совместимости с адресацией пространства данных (об этом — чуть позже), адрес каждого следующего слова в памяти программ вычисляется путем приращения текущего адреса на два.

В основании памяти (адрес 0) содержится команда `GOTO`, за которой (по адресу 2) следует адрес точки входа в стартовый код приложения. Эта последовательность из двух слов выполняется всякий раз при сбросе процессора. Далее размещена таблица векторов прерываний, которая содержит адреса подпрограмм, обслуживающих состояния прерывания. Она занимает область адресов от 4 до 7EH. Затем, после двух зарезервированных командных слов по адресам 80H и 82H в памяти программ размещается дополнительная таблица векторов (адресам с 84H по FEH). Таблица векторов прерываний рассматривается в разделе 3.2, “Структура прерываний”.

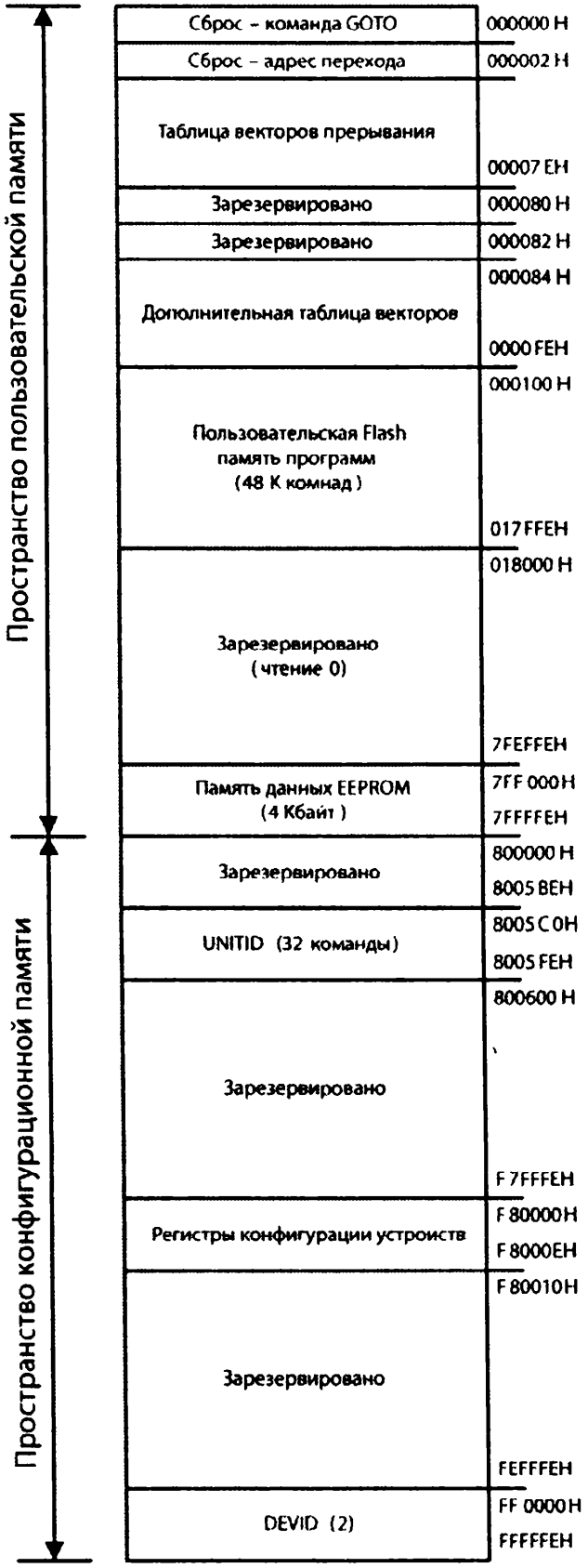


Рис. 3.2. *Распределение памяти программ в dsPIC30F6012A/6014A*

Область памяти программ 100H-17FFEH содержит флэш-память пользовательских программ объемом до 48 К команд (144 Кбайт). За ней следует большая область зарезервированной памяти (18000H–7FEFFEH), заполненная нулями, а последние 4 Кслов пространства пользовательской памяти занимает область данных EEPROM.

Доступ к конфигурационной памяти из пространства пользовательской памяти

Обычно приложение работает с пространством пользовательской памяти, однако, как было упомянуто выше, оно может получать доступ и к определенным областям пространства конфигурационной памяти для чтения идентификатора модуля (32 слова), идентификатора устройства (2 слова) или чтения/записи регистров конфигурации устройств (16 слов). Для этого приложение должно установить в 1 разряд 7 регистра специального назначения TBLPAG (Table Page — страница таблицы), а затем считать или записать данные по требуемому адресу с помощью команды `TBLRD` (чтение данных из конфигурационной памяти) или `TBLWT` (запись данных в конфигурационную память).

Отображение памяти программ в пространстве данных

Зачастую приложения ЦОС хранят константы в энергонезависимой памяти пространства программ, чтобы освободить драгоценные ячейки ОЗУ в пространстве обычных данных. Модифицированная гарвардская архитектура ЦКС dsPIC поддерживает возможность подобного использования памяти программ с помощью вышеупомянутых команд `TBLRD` и `TBLWT` или отображения страницы пространства программ размером 16 Кслов (32 Кбайт) в верхней половине пространства данных с помощью регистра PSVPAG (Program Space Visibility Page — страница видимости пространства программ). Поскольку в архитектуре ЦКС dsPIC задействованы 24-разрядные (трехбайтные) командные слова при 16-разрядных (двухбайтных) словах данных, приложение должно учитывать это различие в выравнивании размещаемых байтов. Применяемые методы перераспределения памяти в случае использования команды `TBLRD` /`TBLWT` и регистра PSVPAG отличаются. Хотя описание механизма применения этих двух подходов представлено ниже в подразделе "Режимы адресации и модули формирования адреса", важно понимать условия, при которых один из них более предпочтителен, чем другой.

Если данные, к которым производится доступ, должны как считываться, так и записываться, то приложение должно применять метод табличного чтения/записи, поскольку применение регистра PSVPAG

допускает только чтение данных. Тем не менее, доступ к данным с помощью регистра PSVPAG быстрее, чем в случае табличного чтения/записи, и он идеально подходит для ситуаций, когда для масштабирования динамических значений используются константы (например, при реализации цифрового фильтра или выполнения быстрого преобразования Фурье). Еще одним ограничением при доступе с помощью регистра PSVPAG является возможность чтения только из пространства X-данных, однако на практике это не столь важно, поскольку динамические данные можно размещать в пространстве Y с помощью MAC-команд, описанных в следующем подразделе. Метод табличного чтения/записи особенно полезен (и даже обязателен) при доступе к ячейкам в пространстве конфигурационной памяти.

Цифровой процессор сигналов

Контроллер dsPIC содержит в себе мощный цифровой процессор сигналов (ЦПС) для выполнения команд умножения с накоплением (MAC), лежащих в основе многих алгоритмов обработки сигналов. Извлечь пользу из применения ЦПС могут только некоторые команды, выполняемые с ограничениями в отношении источников обрабатываемых ими данных, однако эти ограничения минимальны и позволяют модулю считать два операнда, выполнить над ними MAC-команду и сохранить их обратно в памяти — и все это, обычно, — за один цикл.

Для достижения такой пропускной способности разработчики ввели в состав dsPIC специальные аппаратные средства:

- блок умножения 17-разрядных дробных/целых чисел;
- два 40-разрядных накапливающих сумматора;
- 40-ступенчатая схема циклического сдвига, которая за одну команду может выполнять сдвиг на 16 разрядов влево или вправо;
- два модуля формирования адреса.

При надлежащем использовании, эти подсистемы значительно сокращают время обработки сложных алгоритмов. Структурная схема блока ЦПС показана на рис. 3.3. Однако, прежде, чем перейти к обсуждению аппаратных компонентов, необходимо рассмотреть способ представления числовых данных в ЦКС dsPIC.

Представление числовых данных

Контроллер dsPIC может оперировать знаковыми данными в вещественном или целочисленном представлении. Для представления знаковых целых чисел применяется стандартный дополнительный код, в ко-

тором самый старший разряд отводится под знак, а все остальные разряды хранят степень двойки (рис. 3.4).

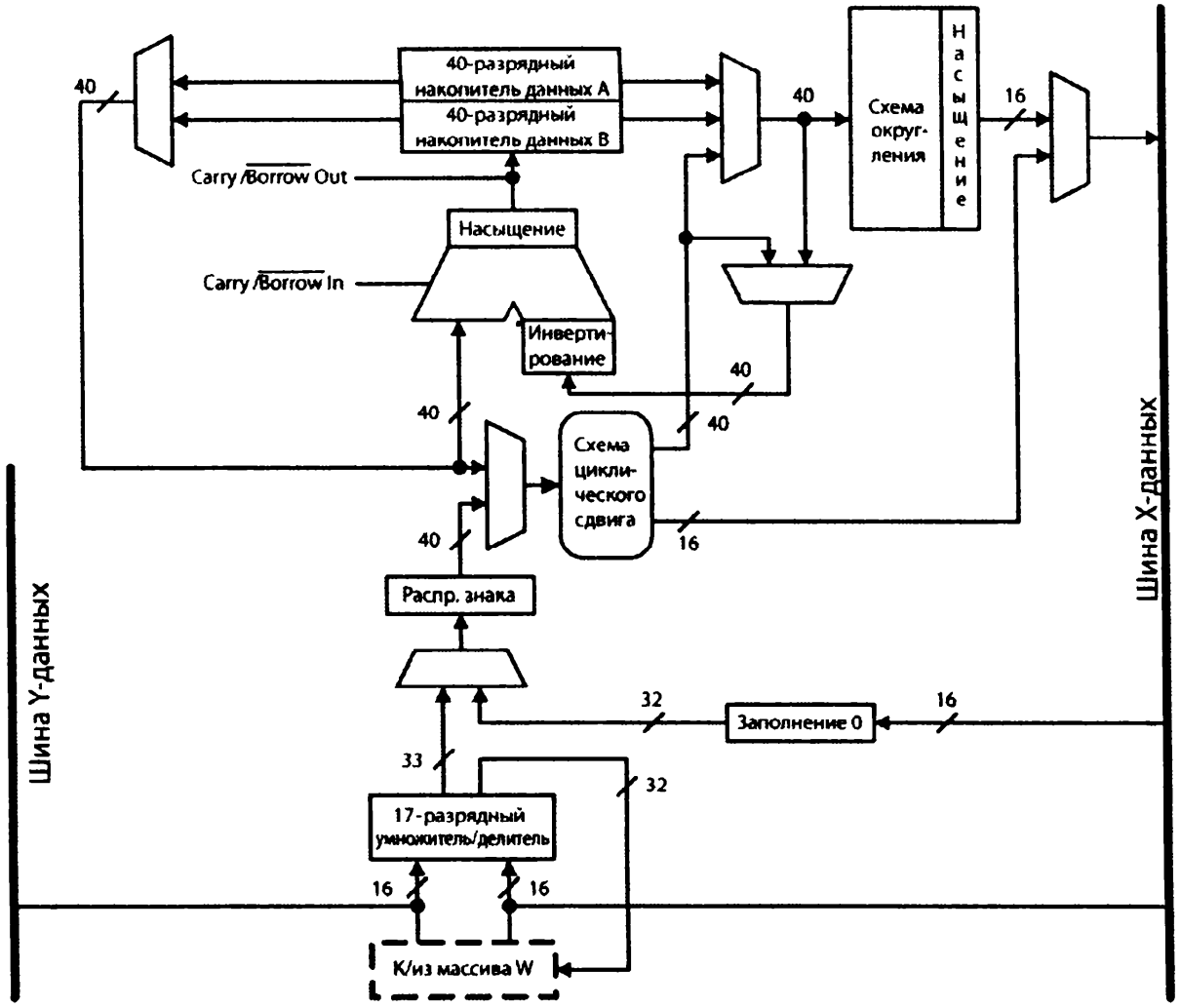


Рис. 3.3. Структурная схема блока ЦПС контроллера dsPIC

-2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}

Рис. 3.4. Представление 16-разрядного целого числа в дополнительном коде

N-разрядное целое число в дополнительном коде может представлять значение в диапазоне от -2^{N-1} до $2^{N-1} - 1$. Таким образом, 16-разрядное целое число в дополнительном коде может принимать любое значение от –32 768 (0x8000) включительно до 32 767 (0x7FFF) включительно, а 32-разрядное число охватывает диапазон от –2 147 483 648 (0x80000000) до 2 147 483 647 (0x7FFFFFFF).

Работая с целочисленными данными в дополнительном коде важно убедиться, что знаковый разряд обрабатывается корректно при выпол-

нении операций сложения или вычитания над числами разной разрядности, иначе могут возникать ошибки. Например, если необходимо сложить восьмиразрядное значение –4 в дополнительном коде (0xFC) с 10-разрядным значением 20 (0x0014) без распространения знакового разряда в восьмиразрядном значении, то в результате будет получено 0x00FC + 0x0014 = 0x0110 или десятичное 272, что далеко от ожидаемого 16 (0x0010). Однако, если до сложения распространить знак[11] в восьмиразрядном значении, то будет получен результат 0xFFFC + 0x0014 = 0x10010 (или 0x0010, если сохранить 16-разрядое представление с потерей старшего разряда результата). Итак, для сложения в дополнительном коде двух значений разной разрядности необходимо прежде распространить знак числа с более коротким представлением до достижения им длины числа с более длинным представлением. Заметьте, что нас интересует длина представления, а не конкретное значение. Например, если мы складываем восьмиразрядное значение с 16-разрядным, то необходимо распространить знак восьмиразрядного значения на восемь разрядов, независимо от фактических значений, содержащихся в восьми- и 16-разрядном представлениях.

Хотя целочисленные данные быстры в обработке, они зачастую ведут к переполнению при умножении. Как следствие, в приложение приходится включать дополнительный программный код для проверки на переполнение и учета его в вычислениях во избежание появления ошибок в результатах. Умножение M-разрядного целого числа на N-разрядное целое число дает в результате (M+N)-разрядное значение. Понятно, что для переполнения блока умножения даже с большой разрядностью, много умножений не потребуется.

Один из способов избежать переполнения — применить арифметику вещественных чисел, в которой все значения ограничиваются диапазоном от –1 до +1. В этом случае результаты умножения гарантировано остаются в постоянном диапазоне. Поскольку этот подход довольно популярен, ЦКС dsPIC можно сконфигурировать для выполнения арифметических операций со знаковыми вещественными числами, используя формат QN. В этом формате данные представляются дробным числом

---

[11] При распространении знака значения в дополнительном коде, знаковый разряд повторяется до тех пор, пока не будет достигнута желаемая длина представления. Например, если необходимо распространить знак восьмиразрядного значения для получения 16 разрядов, то копия знакового разряда (старший разряд) восемь раз повторяется слева этого числа. Распространение знака восьмиразрядного значения 0xFC дает значение 0xFFFC, а распространение знака восьмиразрядного значения 0x7F — значение 0x007F.

в дополнительном коде, где самый старший разряд отведен под знак, а точка (десятичная, если используется десятичная система исчисления) находится, как предполагается, сразу же после знакового разряда. Символ "N" в обозначении "QN" обозначает полное число разрядов в числе. Так запись Q16 соответствует 16-разрядным значениям, а Q32 — 32-разрядным. В литературе этот формат также обозначают как "1.(N–1)", где "(N–1)" соответствует числу разрядов, используемых для представления мантиссы (дробной части числа). Применяя такую форму записи, число Q16 было бы представлено как 1.15, а число Q32 — как 1.31. Пример представления дробного числа Q16, или 1.15 показан на рис. 3.5.

-2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	2^{-9}	2^{-10}	2^{-11}	2^{-12}	2^{-13}	2^{-14}	2^{-15}

Рис. 3.5. *Представление вещественного знакового числа Q16, или 1.15*

Знаковое значение QN может представлять числа в диапазоне от –1 до $1 - 2^{1-N}$. Таким образом, числа Q16 могут представлять значения от –1 до 0,999969482421875, в то время как числа Q32 — значения от –1 до $(1 - 4{,}66 \times 10^{-10})$, т.е. по сути, — от –1 до 1 для всех практических целей.

Как и в случае с целыми числами в дополнительном коде, сложение или вычитание при использовании вещественных чисел QN требует, чтобы перед выполнением операции данные были надлежащим образом расширены. Однако в этом случае расширение представляет собой заполнение нулями вправо от младшего разряда мантиссы числа с более коротким представлением, а не распространение знака, как в случае целочисленной арифметики. Например, чтобы сложить число Q8 с числом Q16, вначале добавляют восемь нулей справа от значения Q8, чтобы сформировать соответствующее значение Q16.

Умножение вещественного числа M.N на вещественное число R.S дает в результате (M+R).(N+S). Когда перемножаются два значения формата 1.15, результатом будет число формата 2.30. Это аналогично ситуации с умножением десятичного вещественного числа с *N* знаками после десятичной точки на второе десятичное дробное число с *S* знаками после десятичной точки, когда в результате будет получено *N*+*S* знаков справа от десятичной точки. Однако при перемножении вещественных чисел в множимом и множителе предварительно распространяют знак (т.е. перед старшим разрядом отрицательных значений добавляется "1", а положительных значений — "0"), а результат смещается влево на один разряд для корректного учета знака и выравнивания разрядов дробного числа. Проиллюстрируем это на примере.

Пример 3.1

Последовательность корректного умножения -0,25 на 0,25, используя представление Q8 (1.7).

1. Распространить знак двух перемножаемых чисел:

-0.25: 1.110 0000b - 1 1.110 0000b

0.25: 0.010 0000b - 0 0.010 0000b

2. Перемножить эти два значения:

```
                1 1.110 0000b
              x 0 0.010 0000b
           11 1100 0000 0000b
          000 0000 0000 0000b
         0000 0000 0000 0000b
       0 0000 0000 0000 0000b
       0 0001 1110 0000 0000b
```

3. Сдвинуть результат на одну позицию влево:

0011 1100 0000 0000b

4. Поместить подразумеваемую десятичную точку перед 13-м разрядом:

001.1 1100 0000 0000b = -0,0625 (десятичное)

Аппаратный умножитель

Аппаратный 17-разрядный умножитель позволяет ЦКС dsPIC перемножать два знаковых 16-разрядных значения, применяя арифметику вещественных или целых чисел. Следует отметить, что, за исключением умножения, операции над целыми и вещественными числами идентичны (если не учитывать способ их интерпретации). Например, если сложить два 16-разрядных целых значения в дополнительном коде (скажем, 0x2001 и 0x17F0), то в результате получится 0x37F1. Если сложить те же знаковые значения (0x2001 и 0x17F0) в формате Q16, то результат будет тот же: 0x37F1. Разница в том, что 0x37F1 в виде 16-разрядного целого числа в дополнительном коде представляет десятичное число 14,321, в то время как то же значение в виде знакового числа Q16 представляет число примерно 0,437042236.

Как было отмечено выше, перемножение знаковых вещественных чисел в дополнительном коде требует распространения знака на один разряд, следовательно для перемножения двух значений Q16 необходим аппаратный 17-разрядный умножитель. Для того чтобы учесть особый

случай, когда перемножаются два числа Q16 со значениями 0x8000 (–1), аппаратный умножитель автоматически корректирует конечный результат к 0x7FFFFFFF (ближайшее значение к +1). Если бы не было этой аппаратной коррекции, то результирующее переполнение произвело бы некорректное значение –1 × –1 = 0. Хотя коррекция вносит незначительную погрешность, она гораздо менее важна, чем переполнение при умножении.

Из-за того, что ЦПС выполняет умножение целых и вещественных чисел по-разному, в приложении должно быть указано, какое представление чисел применяется. Для этого предназначен конфигурационный разряд IF в регистре специального назначения CORCON (Core Configuration). Когда этот флаг сброшен (содержит "0"), умножение производится в вещественном представлении, а когда установлен (содержит "1"), задействовано стандартное целочисленное умножение. В режиме вещественного умножения, автоматически выполняется распространение знака и сдвиг влево для обеспечения корректного размещения точки. Если приложение выполняет команду MAC или MPY (две команды умножения в ЦПС) над данными в виде вещественных чисел, не сбросив предварительно флаг IF, то должен быть программно реализован явный сдвиг влево, иначе результирующее значение окажется неверным.

Тот же самый умножитель используется и для стандартных команд умножения микроконтроллера (варианты команды MUL), которые могут быть восьми- или 16-разрядными (знаковыми или беззнаковыми). Когда оба операнда — восьмиразрядные, то результат будет 16-разрядным, в то время как 16-разрядные операнды дают 32-разрядный результат.

Сдвоенный 40-разрядный накапливающий сумматор

Результат с выхода умножителя подается в накопитель данных — подсистему, состоящую из 40-разрядного сумматора/вычитателя со схемой распространения знака и двух 40-разрядных накопителя (обозначены как "A" и "B"), которые могут служить в качестве как источника, так и получателя для операций накопления. Для определенных команд, таких как ADD (сложение содержимого накопителей) и LAC (загрузка накопителей), данные перед выполнением операции накопления могут масштабироваться с помощью схемы циклического сдвига, что может быть удобным способом нормализации этих данных. Наличие двух накопителей особенно полезно при выполнении арифметических вычислений с комплексными числами — типичных для ЦОС.

Возможно у кого-то возник вопрос: почему накопитель данных имеет разрядность 40, в то время как на выходе умножителя получают 32-

разрядное значение? Дело в том, что накопитель с большей разрядностью обеспечивает дополнительное "пространство" для вычислений во избежание проблем с переполнением при выполнении многократных сложений (как в случае с фильтром, представленного многочленом). Вспомните, что причина, по которой зачастую применяют арифметику вещественных чисел, — это желание избежать проблем с переполнением при умножении путем ограничения значений входных операндов диапазоном от –1 до +1. Так можно быть уверенным, что любой результат на выходе будет находиться в том же диапазоне от –1 до +1.

К сожалению, не существует соответствующего способа ограничить выходное значение, полученное после выполнения сложения или вычитания. Всегда есть потенциальная опасность получить результат, выходящий за пределы диапазона от –1 до +1. Единственный способ предотвратить переполнение — обеспечить представление числа "контрольными" разрядами слева от точки — и чем их больше, тем лучше. В случае с ЦКС dsPIC разработчики предоставили восемь контрольных разрядов (разница между разрядностью сумматора 40 и разрядностью входных значений 32), что позволяет выполнять до 256 последовательных сложений граничных значений диапазона (т.е. двух значений –1 или двух значений +1) прежде, чем накопитель данных переполнится. На практике, в зависимости от значений, уже хранящихся в накопителе, эти восемь контрольных разрядов позволяют выполнить гораздо больше операций до наступления переполнения, или же накопитель вообще никогда не переполняется.

Из-за того что арифметическое переполнение может стать огромной проблемой, для его проверки приложением в ЦКС dsPIC реализованы два флага на каждый накопитель. Эти флаги, расположенные в регистре CORCON, указывают наличие любого из двух состояний:

- переполнение за 39-й разряд (флаг SA или SB);
- переполнение, когда старший разряд результата оказывается в области контрольных разрядов (флаг OA или OB).

Руководство по эксплуатации семейства dsPIC30F обоснованно называет первое из этих состояний "катастрофическим", поскольку теряется знаковый разряд в накопителе. Второе состояние не настолько ужасно, однако оно указывает на потенциальную опасность возникновения катастрофического переполнения при последующих сложениях.

Катастрофическое переполнение может привести к очень серьезным проблемам, поскольку выходные данные после выполнения операции, вызвавшей такое переполнение, перейдут "по кругу" от положительного значения к отрицательному или наоборот. Это соответствует математи-

ческой неоднородности, что нарушает первый принцип ЦОС: "системы могут быть смоделированы как линейные системы". В большинстве микропроцессоров создание программного кода для проверки на наличие катастрофического переполнения и коррекция результата — очень трудоемкий процесс. К счастью, разработчики dsPIC реализовали специальную конфигурируемую схему, которая не только обнаруживает катастрофические переполнения, но и позволяет накапливающему сумматору справляться с ними без любого дополнительного программного кода. Сконфигурированный подобным образом ЦКС dsPIC преобразовывает значение, соответствующее катастрофическому переполнению, в наибольшее отрицательное или наибольшее положительное число (в зависимости от направления переполнения). Такую ситуацию называют *насыщением аккумулятора*. В этом отношении ЦКС dsPIC имитирует обработку переполнения аналоговым умножителем, насыщая результат. В зависимости от приложения, насыщение в ЦКС dsPIC можно ограничить его естественным 32-разрядным или расширенным 40-разрядным диапазоном.

Контроллер dsPIC дает значительную гибкость в обработке флагов обнаружения переполнения. Оба типа флагов (OA/OB и SA/SB) модифицируются при каждом прохождении данных через сумматор/вычитатель, однако только флаги SA/SB могут сбрасываться программно (флаги OA/OB сбрасываются автоматически аппаратным обеспечением при исчезновении состояния переполнения). Кроме того, установка флага любого из этих типов может вызвать системное прерывание "Arithmetic Warning" ("Арифметическое предупреждение"), если в регистре INTCON1 (Interrupt Control 1 — контроль прерываний) установлен соответствующий разряд разрешения внутреннего прерывания по переполнению (OVATEN/OVBTEN — в случае переполнения в области контрольных разрядов, или COVTE — в случае катастрофического переполнения). Это позволяет пользователю сразу же обрабатывать подобные ошибочные состояния (например, уменьшив системный коэффициент).

Для дальнейшего сокращения программных ресурсов, необходимых для обнаружения и обработки состояний переполнения, в ЦКС dsPIC присутствуют также флаги OAB (логическое "ИЛИ" разрядов OA и OB) и SAB (логическое "ИЛИ" разрядов SA и SB). Эти два флага находятся в регистре состояния SR (Status Register) и позволяют приложению быстро обнаруживать переполнение в любом из двух накопителей, что полезно при выполнении сложных арифметических вычислений.

40-разрядная схема циклического сдвига

40-разрядная схема циклического сдвига обеспечивает быстрый (за один цикл) сдвиг данных до 15 разрядов вправо или до 16 разрядов влево. Эта возможность особенно полезна при нормализации данных к определенному диапазону или выравнивании последовательного потока битов, поступающих из порта. Данные помещаются из шины X в схему сдвига в разряды 16–31 (для сдвига вправо) или 0–15 (для сдвига влево).

Режимы адресации и модуль формирования адреса

Одна из основных причин высокой пропускной способности ЦКС dsPIC заключается в применении для эффективного доступа к данным и их обработки нескольких гибких схем адресации. И эта микросхема не только предоставляет многочисленные режимы адресации, но и содержит аппаратные модули формирования адреса, которые позволяют задействовать режимы адресации по модулю и с инвертированием разрядов адреса. Это значительно сокращает программные ресурсы, требуемые для реализации сложных алгоритмов обработки сигналов. Разумное использование возможностей модулей формирования адреса и различных режимов адресации может существенно сократить объем кода и время исполнения, требуемое для реализации таких алгоритмов.

Контроллер dsPIC поддерживает четыре базовых режима адресации (некоторые из них могут быть расширены специальными командами):

- прямая к регистрам специального назначения или к памяти;
- прямая к рабочим регистрам;
- косвенная;
- с непосредственным операндом.

Прямая адресация к регистрам специального назначения или к памяти предполагает наличие в составе команды 13-разрядного абсолютного адреса. Поскольку этот режим позволяет использовать только 13 разрядов адреса, то с его помощью можно получить доступ только к области в 8 Кбайт пространства данных, которая известно как "близкое" ("near") пространство данных. Наверное, наиболее распространенный способ применения этого режима адресации — доступ к регистрам специального назначения, расположенным в нижних 2 Кбайт пространства данных. Учтите, что "ближняя" адресация не может быть использована для доступа к верхним 2 Кбайт области SRAM.

Прямая адресация к регистрам позволяет получить доступ ко всему массиву W-регистров, и команды, поддерживающие этот режим, в качестве источника операнда или пункта назначения результата задают оп-

ределенный W-регистр (например, W1, W4, и т. д.). Этот режим особенно полезен в случае команд с тремя операндами вида Z = X + Y, где X, Y и Z — W-регистры, а в качествен операции может выступать сложение, вычитание или какая-либо поразрядная операция. Имейте в виду, что для команды с тремя операндами оба источника операндов должны быть разными регистрами, хотя пунктом назначения результата может быть любой из этих регистров.

Реализовав режим косвенной адресации, разработчики dsPIC значительно упростили поддержку языков высокого уровня (особенно C), в которых применяются указатели на данные. Отличительной чертой косвенной адресации в dsPIC является то, что адрес хранится в одном из 16-разрядных W-регистров. Этот режим имеет два существенных преимущества над другими режимами адресации:

- он позволяет приложению получить доступ к большему пространству данных (64 Кбайт вместо 8 Кбайт или 32 байт при прямом доступе);
- он позволяет модифицировать адрес данных непосредственно во время выполнения программы.

Среди основных режимов адресации косвенная адресация — наиболее гибкая, поскольку позволяет приложению одновременно с выполнением одноцикловой инструкции инкрементировать или декрементировать содержимое регистра источника операнда и/или регистра назначения результата. Операция инкремента/декремента может выполняться как до извлечения адресуемых данных, так и после него, что позволяет использовать разнообразные циклические конструкции. Кроме того, некоторые команды позволяют включить в регистр адреса регистровое смещение (т.е. объединить для получения требуемого адреса содержимое регистра адреса с содержимым другого регистра) или смещение в виде непосредственного значения (т.е. объединить для получения требуемого адреса содержимое регистра адреса с фиксированным значением). Эти две разновидности адресации позволяют приложению эффективно получать доступ к данным из фиксированных таблиц и структур.

Наконец, в режиме адресации с непосредственным операндом значение данных содержится в самой команде, а их эффективный размер зависит от конкретной команды.

3.2. Структура прерываний

Прерывания — это асинхронные изменения хода выполнения команд, обусловленные каким-нибудь внешним событием (например, изменением состояния сигнала на выходе одного из аппаратных модулей)

или внутренним состоянием (например, завершением счета внутренним таймером). С технической точки зрения, в ЦКС dsPIC различают два типа прерываний:

- "чистые" прерывания, возникающие по наступлению предполагаемого события, обработка которого возлагается на отдельную часть приложения;
- системные прерывания (или процессорные исключения), обрабатывающие ошибочные состояния, которые не должны возникать при нормальном ходе выполнения приложения.

На практике, как "чистые", так и системные прерывания обрабатываются одинаковым образом, поэтому в дальнейшем мы будем, в основном, называть обе эти разновидности просто "прерываниями".

Контроллер dsPIC поддерживает 45 источников прерываний, четыре типа системных прерываний и шесть условий, которые могут привести к сбросу процессора. Поскольку большинство приложений требуют использование лишь нескольких из этих обработчиков исключений, разрешение прерываний в ЦКС dsPIC необходимо устанавливать вручную, используя флаг "Global Interrupt Enable", который позволяет программе быстро, за одну операцию отключить все прерывания. Когда возникает событие прерывания, для которого установлен флаг разрешения, и при этом установлен также и флаг "Global Interrupt Enable", процессор сохраняет адрес следующей команды программы в стеке прерываний и извлекает адрес соответствующей функции обработки прерывания (его еще называют *вектором прерывания*) из таблицы векторов прерываний, расположенной в пространстве программ в области адресов 000004H–00007FH. После извлечения адреса обработчика прерывания, ЦКС dsPIC переходит по этому адресу и начинает выполнять код до тех пор, пока не встретится команда возврата из прерывания (`RTFIE`). С этого момента контроллер выбирает адрес, сохраненный в стеке прерываний последним, и возобновляет выполнение программы, начиная с этого адреса.

Важный аспект обработки прерываний в ЦКС dsPIC — наличие *уровней приоритетности*, которых всего семь, причем уровень 0 имеет самый высокий, а уровень 6 — самый низкий приоритет. Таким образом, прерывания подчиняются встроенной иерархии, когда предполагается, что некоторые источники прерываний важнее других. Принято, что источники прерываний, векторы которых расположены в таблице векторов прерываний по меньшим адресам, имеют более высокий приоритет, а порядок их следования в этой таблице определяет "естественную" приоритетность. Поскольку для разных приложений требуется

различная приоритетность одних и тех же прерываний, разработчики dsPIC предусмотрели возможность изменить приоритет любого прерывания непосредственно во время выполнения программы.

Одним из ключевых аспектов обработки прерываний является время, проходящее с момента возникновения условия прерывания до момента, когда процессор фактически начинает выполнять требуемые действия (его еще называют *задержкой обработки прерывания*). Когда дело касается задержки обработки прерывания, перед нами стоит две задачи:

- минимизировать время, требуемое для того, чтобы начать обработку прерывания;
- удерживать период задержки как можно более стабильным, чтобы обеспечить надежность обработки отдельных частей приложения.

В ЦКС dsPIC эти две задачи решены блестяще. Задержка обработки прерывания фиксирована и занимает пять командных циклов от момента поступления запроса на прерывание (т.е. времени, когда источник прерывания становится активным) до момента, когда начинает выполняться подпрограмма обслуживания прерывания. Однако (что важно помнить) при этом предполагается, что не обрабатываются никакие другие прерывания с приоритетом текущего или выше. Если же такие прерывания существуют, то они будут обработаны до конца, и только потом начнется обслуживание нового прерывания. В таком случае задержка обработки прерывания может значительно превысить базовые пять циклов.

Теневые регистры

До сих пор мы еще не упоминали о таком важном свойстве ЦКС dsPIC, как использование теневых регистров для обеспечения быстрого контекстного переключения при возникновении прерываний или каких-либо других форм исключений. Когда возникает прерывание, приложение, выполнив единственную команду `PUSH.S`, может сохранить в теневых регистрах содержимое определенных флагов из регистра состояния, а также содержимое регистров TBLPAG, PSVPAG и W0–W14. По завершении обработки прерывания подпрограмма его обслуживания должна выполнить команду `POP.S`, чтобы восстановить содержимое стандартных регистров, сохраненное в теневых регистрах.

Поскольку теневые регистры могут хранить только один набор данных, крайне важно следить за тем, чтобы в приложении никогда не использовалось два последовательных вызова команд `PUSH.S` или `POP.S` без промежуточного вызова соответственно `POP.S` или `PUSH.S`, иначе

первоначальное содержимое сохраняемых регистров будет утеряно. Это означает, что подпрограммы обслуживания прерываний с более низким приоритетом не могут использовать теневые регистры, если они уже используются подпрограммами обслуживания прерываний с высшим приоритетом.

3.3. Встроенные периферийные устройства

Одной из самых сильных сторон dsPIC является тот уровень системной интеграции, который этот контроллер приносит в сферу измерений. В зависимости от конкретной версии, один ЦКС dsPIC может содержать встроенные периферийные модули, которые напрямую оцифровывают входные аналоговые сигналы или считывают оцифрованные данные с внешних АЦП; несинхронизированные и/или основанные на событиях таймеры; интерфейсы, поддерживающие несколько промышленных стандартных протоколов обмена данными. Поскольку архитектура dsPIC позволяет разработчику определять приоритет обработки для отдельных периферийных модулей непосредственно в ходе выполнения программы, существует возможность оптимизировать работу микросхемы и пропускную способность в каждом конкретном случае применения.

Цель представленного ниже описания периферийных модулей — разъяснить информацию из технических описаний контроллеров dsPIC и определить контрольный перечень, к которому может обращаться разработчик, чтобы убедиться в надлежащем использовании каждого модуля. Этот материал также содержит выдержки из других ресурсов компании Microchip, наподобие руководств по эксплуатации и рекомендаций по применению.

Периферийные модули сбора данных

В любой системе с интеллектуальным датчиком "цифровая" часть в цепи следования сигнала[12] при ЦОС начинается с того момента, когда входной аналоговый сигнал оцифровывается так называемым аналого-цифровым преобразователем (АЦП). Контроллер dsPIC поддерживает несколько различных способов оцифровки сигнала:

- с помощью встроенного многоканального АЦП;

---

[12] Под *цепью следования сигнала* подразумеваются стадии обработки, через которые проходит сигнал от входа до выхода. Эта цепь обычно состоит как из аналоговой, так и из цифровой части (по крайней мере, в системах, в которых применяются методики ЦОС).

- с помощью внешнего аудиокодека, выходные данные которого считываются через интерфейс преобразователя данных DCI (Data Converter Interface) ЦКС dsPIC;
- через внешний АЦП, управляемый по порту интерфейса SPI (Serial Peripheral Interface) или порты интерфейса I^2C (Inter-Integrated Circuit).

В этой книге рассматривается первый метод: использование встроенного АЦП.

Аналого-цифровые преобразователи

Наверное, самый популярный и, разумеется, — наиболее эффективный метод оцифровки входного сигнала — это использование АЦП, встроенного в ЦКС dsPIC. Все модели серии dsPIC30F содержат от шести до 16 входных каналов АЦП с разрешением 10 или 12 разрядов (в зависимости конкретной микросхемы). Устройства с 10-разрядными АЦП, в основном, предназначены для управления моторами или для видеообработки, для чего достаточно 1 024 уникальных уровней (по сравнению с 4 096 уровнями в случае 12-разрядных АЦП), в то время как 12-разрядное разрешение предназначено для обслуживания датчиков общего назначения. Контроллер dsPIC30F6014A, который мы задействуем в рассматриваемых в книге примерах применения, поддерживает 16 каналов 12-разрядного преобразования.

Для того чтобы представить себе влияние разницы в разрешении, рассмотрим два примера из реального мира:

- слежение за положением рулевого колеса в системе с электронным управления;
- измерение давления внутри формовочной машины для литья.

В первом случае допустим, что установка всех разрядов АЦП соответствует повороту рулевого колеса на 360°. При таких условиях 10-разрядный АЦП позволил бы нам измерять положение рулевого колеса с точностью примерно в одну треть градуса (360° / 2^{10} уровней = 360° / 1 024 = 0,35° / уровень), тогда как 12-разрядный АЦП был бы способен распознавать вращение с шагом менее десятой доли градуса. Оба эти варианта вполне приемлемы для большинства систем электронного рулевого управления, учитывая ограниченность чувствительности в восприятии человека (если, конечно, речь идет об управлении человеком).

Однако все далеко не так же ясно в ситуации, когда осуществляется контроль сигнала с большим диапазоном значений (например, давление внутри установки для литья, которое обычно изменяется в диапазоне от

0 до 13 600 кг/дюйм^2. В этом случае одному уровню 10-разрядного АЦП соответствует около 13,5 кг/дюйм^2, в то время как одному уровню 12-разрядного АЦП — около 3,2 кг/дюйм^2. Это может показаться не особенно существенным, однако в том случае, когда формовочная машина использует определенное давление внутри формы для открытия клапана и впрыска дополнительного количества пластмассы, то разница в разрешении АЦП может означать разницу между стабильным, рентабельным процессом с низким объемом отходов и нестабильным, нерентабельным процессом с недопустимо большим объемом отходов.

Как отмечает Ричард Фишер (Richard Fischer) из компании Microchip в своей превосходной Internet-статье о модуле АЦП контроллера dsPIC[13], АЦП каждый раз, преобразовывая входной аналоговый сигнал в соответствующее цифровое значение, выполняет две основные операции. Вначале он дискретизирует аналоговый входной сигнал и сохраняет эту *аналоговую* выборку в усилителе типа "выборка/хранение". Этот процесс называют *сбором*. Ричард отмечает, что фиксирование аналоговой выборки аналогично получению фотоснимка аналоговой кривой, где время сбора подобно времени экспозиции. Приложение определяет, сколько времени может отводиться для каждой операции сбора, однако этого времени должно быть заведомо достаточно для точной фиксации сигнала, точно так же, как фотограф должен убедиться, что времени открытия затвора объектива будет достаточно для попадания на фотопленку (или в нашу эру цифровых фотоаппаратов — на матрицу ПЗС) достаточного количества света для получения хорошего снимка.

После того как сигнал зафиксирован, АЦП выполняет вторую операцию: преобразует квантованный аналоговый сигнал в соответствующее цифровое значение. Этот процесс обычно называют *преобразованием*. На рис. 3.6 показано соотношение между временем дискретизации (временем сбора), временем аналого-цифрового преобразования и общим временем цикла преобразования. В зависимости от того, как сконфигурирован АЦП, цифровые значения, полученные в результате преобразования, могут быть представлены в одном из четырех форматов целых или вещественных чисел. Это позволяет приложению использовать такое представление, которое наилучшим образом подходит для конкретных алгоритмов.

---

[13] Статью "dsPIC30F 12-bit ADC Module, Part1 of 2, page 6" можно найти на Web-сайте компании Microchip.

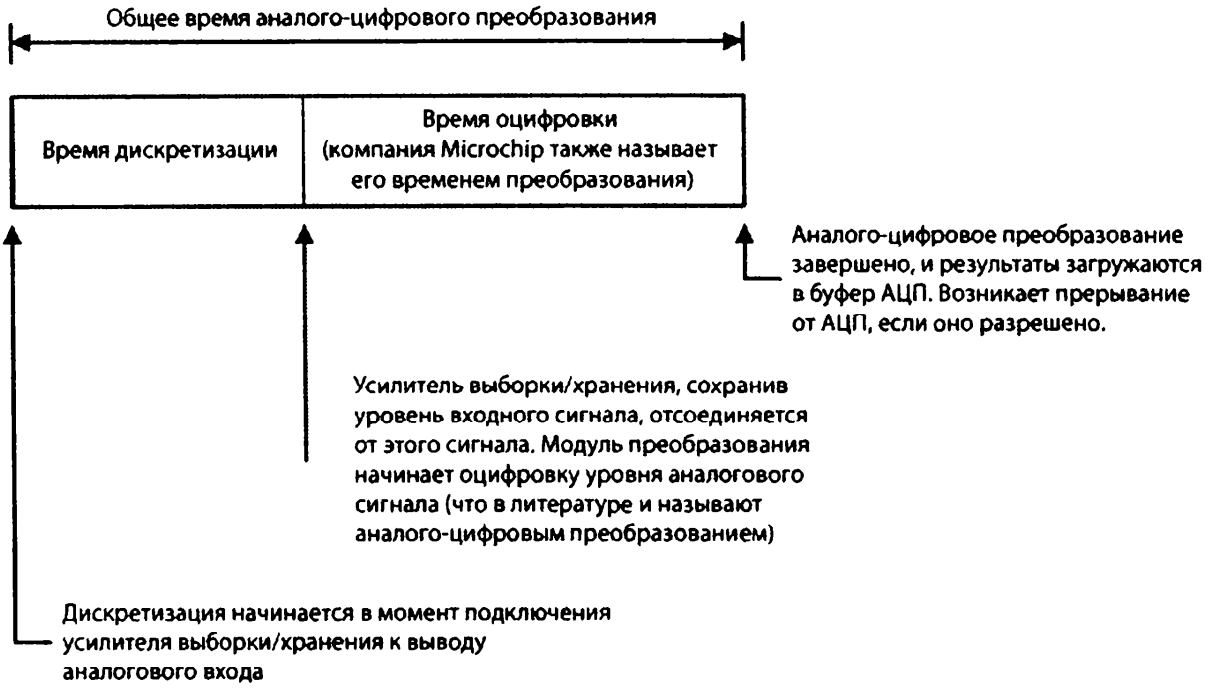


Рис. 3.6. *Временная диаграмма аналого-цифрового преобразования*

В случае применения dsPIC, дискретизация данных может выполняться по методу *опроса*, когда начало каждой операции дискретизации задается в программном коде явным образом, или же автоматически. В последнем случае АЦП предварительно конфигурируется, после чего происходит периодический аппаратный сбор и преобразование, прерывая работу процессора только после преобразования заданного числа каналов. В большинстве случаев предпочтение отдается управлению по прерыванию, поскольку оно гарантирует временную стабильность дискретизации, т.е. время между смежными выборками очень стабильно. Такая временная стабильность очень важна, поскольку для большинства алгоритмов ЦОС предполагается, что частота дискретизации остается постоянной. Избыточный уровень дрожания[14] может приводить к ошибочным результатам, избежать которых позволяет более строгий контроль за частотой дискретизации. Метод опроса обычно используется при отладке аппаратного интерфейса, когда в асинхронном режиме может проверяться значение, измеряемое в выбранный момент времени, или же в том случае, когда желаемая частота дискретизации слишком низкая для модуля АЦП.

---

[14] Под *дрожанием* понимают отклонение периода дискретизации от его номинального значения. Например, если номинальный период дискретизации для системы составляет 2 мс, а реальный изменяется в пределах от 1,8 до 2,1 мс, то дрожание составляет 0,3 мс или 15%, что не особо хорошо.

В контроллере dsPIC используется единственный АЦП с последовательным приближением, который для проведения оцифровки переключается между всеми входными каналами. В зависимости от разрешающей способности АЦП и конкретного устройства, входные каналы перед преобразованием в цифровое значение мультиплексируются через один, два или четыре усилителя выборки/хранения (рис. 3.7).

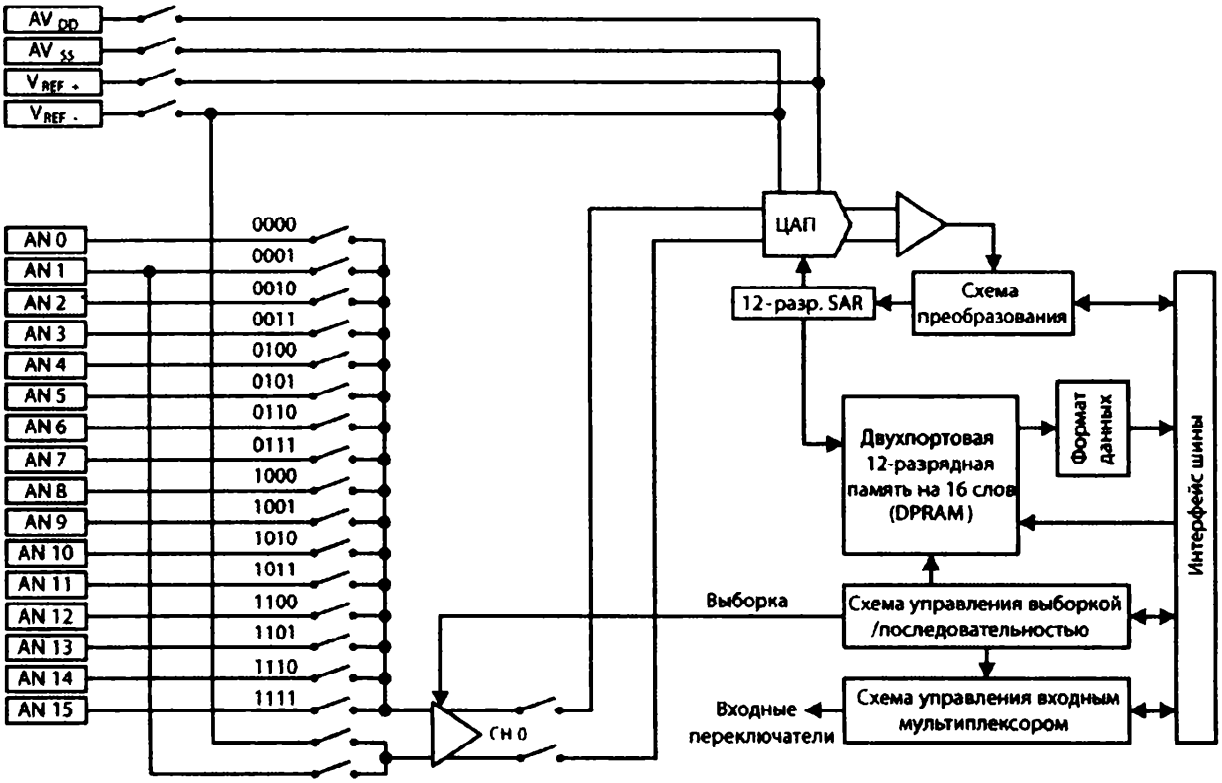


Рис. 3.7. *Схема 12-разрядного высокоскоростного аналого-цифрового преобразования в dsPIC*

Только 10-разрядные АЦП поддерживают несколько усилителей выборки/хранения — 12-разрядные АЦП ограничены одним таким усилителем. На первый взгляд, способность использовать несколько усилителей выборки/хранения не столь важна, поскольку выходные сигналы усилителей все равно должны быть преобразованы единственной схемой преобразования. Однако несколько таких усилителей предоставляет важную возможность выполнять несколько одновременных выборок. Эта возможность используется не так часто, однако в некоторых сферах применения dsPIC она чрезвычайно полезно.

Внутренний АЦП может производить до 200 000 выборок в секунду при полном уровне питающего напряжения, однако эта полоса пропускания распределена по всем каналам. Таким образом, при использовании всех каналов каждый канал может выполнять дискретизацию со

скоростью максимум 12 000 выборок в секунду (200 000 / 16 каналов). На практике, разумеется, частота дискретизации обычно снижаться для того, чтобы предоставить достаточно времени на обработку алгоритмов формирования и анализа. Частота дискретизации зависит также от напряжения питания микросхемы и может достигать 200 000 выборок в секунду только в том случае, если напряжение питания находится в диапазоне 4,5–5,5 В. Если микросхема работает на более низком напряжении, то максимальная частота дискретизации падает до 100 000 выборок в секунду.

Как показано на рис. 3.7, схема АЦП предоставляет значительную гибкость. Однако для того, чтобы правильно воспользоваться этой гибкостью, требуется существенный объем конфигурирования со стороны приложения. Модуль АЦП использует шесть 16-разрядных регистров состояния и управления и дополнительный, доступный только для чтения (от ADCBUF0 до ADCBUFF) двухпортовой буфер объемом в 16 слов, содержащий результаты преобразований.

Вместо того, чтобы просто перечислить назначение каждого разряда в регистрах состояния и управления, мы будем изучать вопрос конфигурации путем последовательного исследования поставленных перед нами задач в логическом порядке их выполнения. Надеемся, это поможет читателю лучше понять весь процесс, хотя, для полноты изложения, в завершение мы представим и общий перечень конфигурационных разрядов.

Итак, прежде, чем конфигурировать АЦП, необходимо определить следующее.

- Какие выводы порта должны быть аналоговыми входами, какие — цифровыми входами, и какие — цифровыми выходами?
- Какой должна быть частота дискретизации аналогового сигнала?
- Требуется ли чередовать дискретизацию и обработку, или же можно просто вызывать единственное прерывание после преобразования всех сигналов в течение заданного периода дискретизации? Требует ли пропускная способность мультиплексирования буфера результатов?
- В каком формате нам нужны преобразованные данные?

Ответив на эти вопросы, можно приступать к фактическому конфигурированию модуля.

Шаг 1. Конфигурация тракта сигнала

Не удивительно, что первый шаг в конфигурировании АЦП — это выяснение выводов, которые будут выполнять роль аналоговых и циф-

ровых входов, а также цифровых выходов. Хотя эту конфигурацию можно изменить в любой момент в ходе выполнения программы, для простоты будем предполагать, что мы конфигурируем типы сигналов и направления передачи данных только один раз в начале программы. Если, вдруг, приложению потребуется изменить типы сигналов и/или направление передачи позже, мы можем это сделать, выполнив рассмотренные здесь действия, но только *после* того, как будут запрещены прерывания.

Шаг 1а. Конфигурация выводов порта ввода-вывода

Как определено во всех устройствах Microchip, приложение задает направление сигнала для порта ввода-вывода с помощью регистра TRISx, где "x" соответствует конкретному порту (TRISA для PORTA, TRISB для PORTB и т. д.). Каждая пара разрядов с одинаковыми порядковыми номерами в регистрах TRISx и PORTx соответствует выводу с тем же номером. Для того чтобы сконфигурировать вывод как входной (будь то цифровой или аналоговый), приложение устанавливает соответствующий разряд в регистре TRISx в "1", а для того чтобы сконфигурировать вывод как выходной, приложение очищает этот разряд (записывает в него "0"). Распределение разрядов регистра TRISx показано на рис. 3.8.

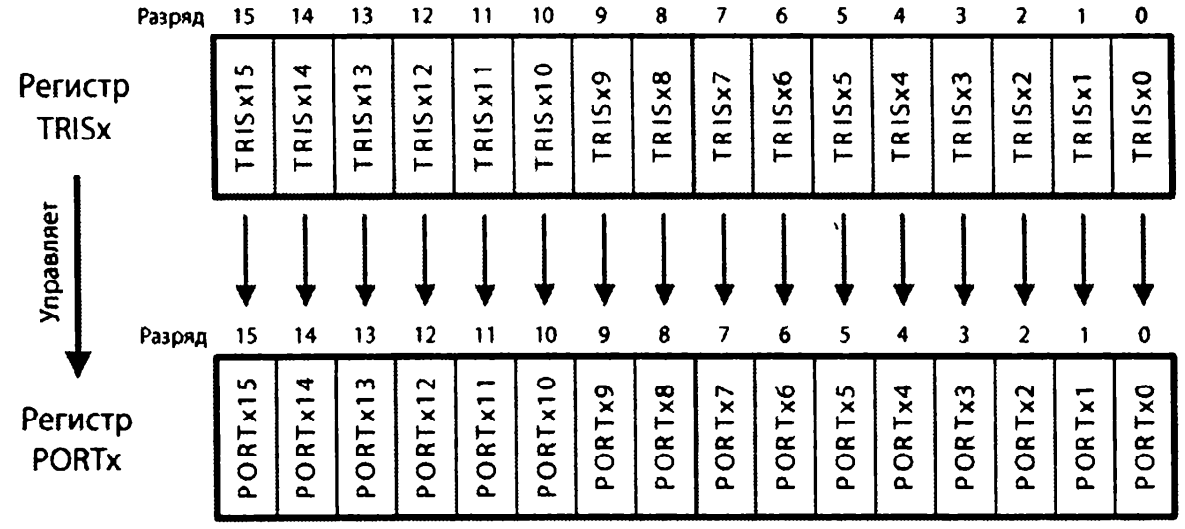


Рис. 3.8. *Распределение разрядов регистра TRISx*

В контроллере dsPIC30F6014A все аналоговые входы (обозначенные в техническом описании как AN0–AN15) определены в порту B, поэтому нас будет интересовать регистр TRISB. Если мы хотим оцифровывать сигнал на определенной линии порта B, необходимо установить в "1" соответствующий разряд в регистре TRISB. Например, если мы

хотим оцифровывать сигналы на выводах AN0, AN1 и AN8, используя в качестве цифровых входов линии 4 и 6 порта B, а остальные выводы — определить как цифровые выходы, то в регистр TRISB необходимо записать значение 0x0153 — значение, полученное путем объединения с помощью логической операции "ИЛИ" значений 0x0103 (аналоговые входы) и 0x0050 (цифровые входы).

Задав направление прохождения сигнала, необходимо указать, каким является тот или иной вход: аналоговым или цифровым. Для этого соответствующим образом конфигурируют 16-разрядный регистр ADPCFG (A/D Pin Configuration — конфигурация выводов АЦП). Каждый разряд в этом регистре соответствует одному из аналоговых входов (от AN0 до AN15). Установка такого разряда в "1" конфигурирует соответствующий вывод как цифровой вход, а запись в него "0" — как аналоговый вход. Распределение разрядов в этом регистре показано на рис. 3.9.

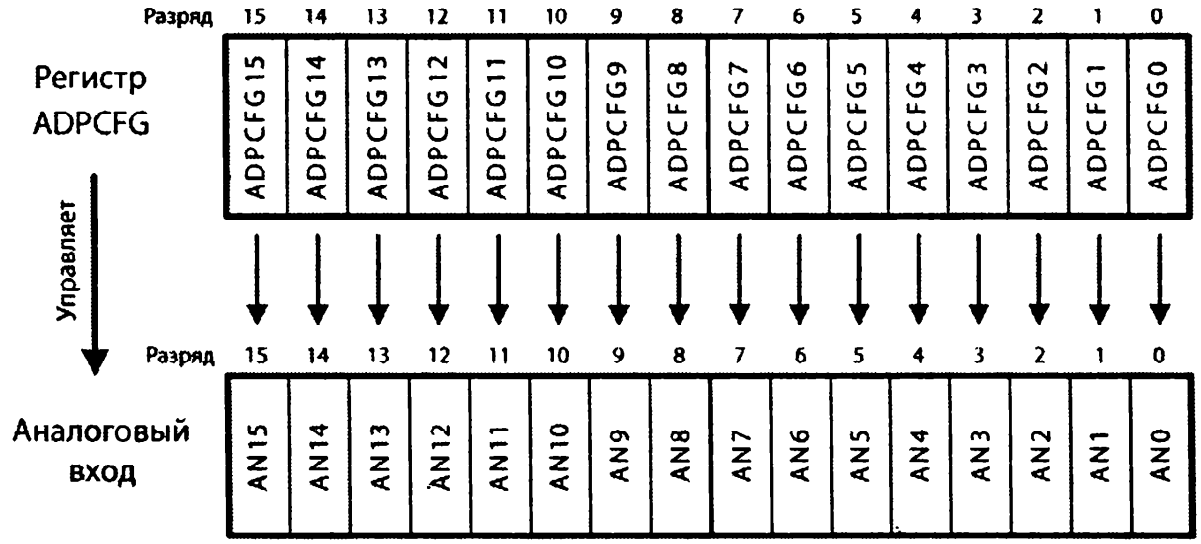


Рис. 3.9. *Распределение разрядов в регистре конфигурации выводов АЦП*

Продолжая рассматривать предыдущий пример, нам необходимо записать в регистр ADPCFG значение 0xFEFC, оставляя очищенными только разряды AN0, AN1 и AN8 (аналоговые входы AN0, AN1 и AN8).

А что произойдет, если приложение сконфигурирует какой-нибудь вывод с помощью регистра TRISB как цифровой выход, но в регистре ADPCFG определит его как аналоговый вход? В этом случае вывод будет функционировать как цифровой выход, но, кроме того, появляется возможность выполнять оцифровку уровня напряжения на этом выводе и считывать его через модуль АЦП. Это свойство может пригодиться в приложениях, критичных к безопасности, чтобы программа могла определять, находится ли на самом деле данный цифровой выход в кор-

ректном состоянии или нет (например, закорочен ли он на "землю" или на него подано напряжение VDD).

Хотя далее мы рассмотрим конфигурирование самого модуля АЦП, в большинстве приложений разработчик обычно вначале завершает конфигурирование остальных портов ввода-вывода (т.е. портов A, C и т. д.), чтобы аппаратное обеспечение как можно быстрее перешло в безопасное состояние. Важно отметить, что уровни аналогового напряжения на выводе, сконфигурированном как цифровой вход, могут вызывать потребление входным буфером большего тока, чем это регламентируется для устройства (если говорить нетехническим языком, то это означает, что мы можем повредить саму микросхему).

Шаг 1б. Выбор источников опорного напряжения

Когда направление прохождения и тип сигнала сконфигурированы, мы переходим к выбору опорных напряжений, которые будут использованы для оцифровки. Для простоты разработки, преобразования часто производятся в полном диапазоне питающего напряжения (от AV_{SS} до AV_{DD}), однако для улучшения разрешающей способности измерений, опорные напряжения преобразований могут ограничиваться диапазоном от V_{REF-} до V_{REF+}. Это может существенно улучшать способность устройства измерять входные сигналы из ограниченного диапазона, поскольку преобразование происходит только в интересующем диапазоне напряжений. Например, если разработчик знает, что уровни входного сигнала будут находиться в диапазоне 0,5–1,5 В, то установка нижнего и верхнего опорного напряжения на эти уровни позволит АЦП производить преобразования с шагом 0,24 мВ ((V_{REF+} – V_{REF-}) / 2^{12} уровней = (1,5 В – 0,5 В) / 4096 уровней = 0,24 мВ/уровень). Если бы преобразование выполнялось с использованием V_{REF-} = 0 В и V_{REF+} = 5 В, как могло бы быть в случае стандартной системы с питанием 5 В, то разрешающая способность была бы в пять раз хуже (1,2 мВ/уровень). Возможно, на первый взгляд, в этом нет ничего плохого, однако для низковольтных входных сигналов, наподобие тех, которые формируются термопарой, это различие может быть очень ощутимым.

Также существует возможность сконфигурировать модуль таким образом, чтобы одно входное напряжение преобразовывалось относительно другого входного напряжения, производя дифференциальное измерение. Однако, этот режим имеет ограничение: подобное преобразование — униполярное, т.е. опорное входное напряжение всегда должно быть меньше или равно напряжению на другом входе. Тем не менее,

даже с учетом этого ограничения, возможность дифференциации может пригодиться для уменьшения синфазного шума[15] на входах.

Напоминаем, что верхний и нижний уровни опорного напряжения определяют диапазон аналоговых входных напряжений, которым соответствуют 10- или 12-разрядные выходные значения АЦП. Наша задача — использовать диапазон опорных напряжений, который максимально соответствует диапазону входного сигнала, чтобы получить наилучшую разрешающую способность преобразования в цифровые значения. Контроллер dsPIC предоставляет четыре возможных комбинации верхнего и нижнего опорных напряжений, поскольку мы можем выбрать в качестве нижнего опорного напряжения AV_{SS} или V_{REFL}, а в качестве верхнего — AV_{DD} или V_{REFH}.

Задать источники опорного напряжения очень просто: приложение просто устанавливает разряды VCFG2..VCFG0 (Voltage Configuration — конфигурация напряжения) в регистре ADCON2 (A/D Control Register — регистр управления АЦП). Хотя, ввиду наличия трех разрядов VCFG, теоретически можно определить восемь конфигураций опорного напряжения, установка в "1" старшего разряда (VCFG2) всегда дает в качестве верхнего уровня опорного напряжения AV_{DD}, а в качестве нижнего уровня — AV_{SS}. В табл. 3.1 показаны конфигурации опорного напряжения, соответствующие различным сочетаниям разрядов VCFG, а на рис. 3.10 — размещение разрядов VCFG в регистре ADCON2.

Разряд	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Регистр ADCON 2	VCFG2	VCFG1	VCFG0	-	-	CSCNA	-	-	BUFS	-	SMPI3	SMPI2	SMPI1	SMPI0	BUFM	ALTS

Рис. 3.10. Размещение разрядов VCFG в регистре ADCON2

---

[15] *Синфазный шум* — электрический шум, который возникает на обоих входах дифференциального сигнала. Обычно он проявляется в витых парах, в которых переплетаются два провода, в результате чего шум, проникающий на один провод, проникает также и на другой. Теоретически, дифференциальный сигнал невосприимчив к этому типу шума, поскольку приемник считывает только разность между напряжениями на двух проводниках, так что любое напряжение (включая напряжение шума), которое присутствует в одном из проводников, вычитается из напряжения в другом проводнике. Хотя на практике так бывает не всегда, это, все же, — очень полезная методика существенного ослабления воздействия нежелательного шума.

Таблица 3.1. Конфигурационные значения опорного напряжения АЦП

VCFG<2:0>	V_{REFH}	V_{REFL}
000	AV_{DD}	AV_{SS}
001	V_{REF+}	AV_{SS}
010	AV_{DD}	V_{REF-}
011	V_{REF+}	V_{REF-}
1xx	AV_{DD}	AV_{SS}

V_{REFH} — верхний уровень опорного напряжения АЦП;
V_{REFL} — нижний уровень опорного напряжения АЦП;
V_{REF+} — внешний верхний уровень опорного напряжения;
V_{REF-} — внешний нижний уровень опорного напряжения;
AV_{DD} — положительная аналоговая шина питания;
AV_{SS} — отрицательная аналоговая шина питания ("земля")

Шаг 1в. Выбор аналоговых входов для оцифровки

Для сокращения потребления процессорных ресурсов приложение может определить для преобразования конкретные аналоговые входы, а также последовательность (по крайней мере, в значительной степени) и число преобразований, которые необходимо выполнить до прерывания работы процессора. Это высвобождает существенное количество времени процессора в приложениях, где необходима только часть входных каналов АЦП, поскольку в таком случае микропрограммам не нужно обслуживать каналы, данные которых никогда не будут востребованы.

Во всех режимах дискретизации оцифрованные данные сохраняются в буфере ADCBUF размером в 16 слов в том порядке, в котором они квантуются, начиная с ячейки ADCBUF0 и до тех пор, пока число выборок не достигнет значения, определенного набором разрядов SMPI (Sample Per Interrupt — выборок за прерывание) в регистре ADCON2, и с этого момента значение следующей выборки опять сохраняется по адресу ADCBUF0. Учитывая такую схему, обработчик прерываний АЦП должен обеспечить выгрузку всех необходимых данных из буфера ADCBUF и их обработку за время одной выборки, поскольку в противном случае значение выборки, хранимое в ячейке ADCBUF0, будет утеряно (затерто следующим значением выборки).

Модуль АЦП может преобразовывать сигналы по одному из двух методик, конфигурируемых пользователем (для большей гибкости их даже можно комбинировать). Первая методика, известная как *дискретизация с чередованием каналов*, подразумевает следующую последовательность операций.

1. Оцифровка заданного входного канала (в документации он обозначается как вход Mux A).
2. Сохранение оцифрованного значения в ячейке ADCBUF0.
3. Оцифровке второго входного канала (обозначается как Mux B), который обычно отличается от первого.
4. Сохранение полученного значения в ячейке ADCBUF1.

Эта последовательность циклично повторяется до тех пор, пока в буфере ADCBUF не накопится число выборок, заданное набором разрядов SMPI. По достижению этого момента модуль АЦП формирует запрос на прерывание (при условии, что оно разрешено). Значение следующей выборки, полученное после прерывания, сохраняется в ячейке ADCBUF0, и процесс повторяется сначала.

Для того чтобы активизировать дискретизацию с чередованием (т.е. попеременно на двух входах), приложение должно установить в "1" разряд ALTS (Alternate Sampling — поочередная дискретизация) в регистре ADCON2. Затем должны быть назначены определенные каналы для дискретизации в качестве входов Mux A и Mux B путем записи в четырехразрядное поле CH0SA (Channel 0 Select A) и поле CH0SB (Channel 0 Select B) регистра ADCHS (A/D Channel Select Register — регистр выбора канала АЦП) номера (начиная с нуля) требуемых входов Mux A и Mux B. Той же операцией записи в регистр ADCHS приложение должно выбрать источник для отрицательного входа схемы выборки/хранения для входов Mux A и Mux B, записав соответствующие значения в разряды флагов CH0NA (Channel 0 Negative Input Mux A) и CH0NB (Channel 0 Negative Input Mux B). Сброс флага CH0NA или CH0NB в "0" приводит к выбору в качестве отрицательного входа для соответствующего канала Mux напряжения V_{REF-}, а установка в "1" — к выбору напряжения на входе AN1. Таким образом модуль АЦП может поддерживать униполярный дифференциальный режим. При этом выполняется важное требование: напряжение на AN1 всегда меньше или равно напряжению на выбранном входе Mux A или Mux B.

Вторая методика известна как *сканирование каналов*. Она предполагает дискретизацию сигналов в группе полупоследовательных каналов, когда модуль формирует запрос на прерывание после того, как будет выполнено соответствующее число преобразований. Термин *полупоследовательный* означает, что хотя преобразования в каналах и происходят в установленном последовательном порядке (от наименьшего номера к наибольшему), отдельные каналы могут исключаться из последовательности дискретизации, если в них нет необходимости.

Как и в случае дискретизации с чередованием, оцифрованные данные после преобразования сохраняются в буфере ADCBUF, начиная с ячейки ADCBUF0. После того как в ADCBUF сохраняется SMPI выборок, формируется запрос на прерывание, чтобы приложение могло считать данные из буфера. Новые выборки, следующие за прерыванием, сохраняются в буфере ADCBUF, начиная с ячейки ADCBUF0, даже если буфер в предыдущий раз не был заполнен полностью.

Наконец, ЦКС dsPIC поддерживает сочетание дискретизации с чередованием и сканирования каналов с тем ограничением, что сканирование каналов может выполняться только одновременно с дискретизацией канала Mux A (т.е. Mux B — всегда отдельный вход). Это может пригодиться при сканировании ряда динамических входов и входа опорного напряжения (например, температуры компенсации холодного спая термопары[16]).

Поясним эти три методики дискретизации на простых примерах.

Пример 1. Базовая дискретизация с чередованием

Допустим, нам необходимо выполнить попеременную дискретизацию каналов 2 (AN2) и 14 (AN14). На вход канала 2 должно быть подано опорное напряжение V_{REF-}, на вход канала 14 — опорное напряжение со входа AN1 (униполярный дифференциальный режим), а модуль АЦП должен формировать запрос на прерывание после трех выборок в каждом канале (всего — шесть выборок на прерывание). Преобразования должны выполняться для полного диапазона напряжений, т.е. с использованием AV_{SS} и AV_{DD} в качестве нижнего и верхнего опорных напряжений для схемы преобразователя (в противоположность опорному напряжению для схемы выборки/хранение). В этом случае получаем следующую конфигурацию:

CH0SA = 0x02	(Mux A — канал 2)
CH0NA = 0	(на Mux A подано опорное напряжение V_{REF-})
CH0SB = 0x0E	(Mux B — канал 14)
CH0NB = 1	(на Mux B подано опорное напряжение со входа AN1)
ADCHS = 0x1E02	(CH0NB — в разряде 12, CH0SB — в разрядах 11..8, CH0NA — в разряде 4, CH0SA — в разрядах 3..0)
ALTS = 1	(Разрешение поочередной дискретизации)

[16] Компенсация холодного спая — методика, применяемая для учета зависящей от температуры разницы напряжений, которая возникает в месте спая двух разнородных металлов. Этот вопрос будет рассмотрен подробнее позже на примере многоканальной системы мониторинга температуры.

BUFM = 0	(Не разбивать ADCBUF)
SMPI = 5	(Число выборок за прерывание — 6)
CSCNA = 0	(Не сканировать входы)
VCFG = 0	($ADV_{REFH} = AV_{DD}$, $ADV_{REFL} = AV_{SS}$)
ADCON2 = 0x0019	(VCFG — в разрядах 15..13, CSCNA — в разряде 13, SMPI — в разрядах 5..2, BUFM — в разряде 1, ALTS — в разряде 0)
ADCSSL = 0x0000	(Сканирование отсутствует, поэтому это значение, по сути, ничего не означает)

Пример 2. Сканирование каналов

Допустим, мы хотим сканировать каналы 1–3, 6, 8, 10 и 12 с формированием запроса на прерывание после фиксации двух выборок для каждого канала (т.е. всего — после 14 выборок). Данные будут получаться от преобразований в диапазоне напряжений от V_{REF-} до V_{REF+}. В таком случае конфигурация модуля АЦП будет выглядеть следующим образом:

CH0SA = 0	(MuxA = 0, хотя это значение, по сути, роли не играет)
CH0NA = 0	(на Mux A подано опорное напряжение V_{REF-}, хотя, по сути, это не имеет значения)
CH0SB = 0	(MuxB = 0, хотя это значение, по сути, роли не играет)
CH0NB = 0	(на Mux B подано опорное напряжение V_{REF-}, хотя, по сути, это не имеет значения)
ADCHS = 0x0000	(CH0NB — в разряде 12, CH0SB — в разрядах 11..8, CH0NA — в разряде 4, CH0SA — в разрядах 3..0, хотя, по сути, это не имеет значения)
ALTS = 0	(Запрет поочередной дискретизации)
BUFM = 0	(Не разбивать ADCBUF)
SMPI = 13	(Число выборок за прерывание — 14)
CSCNA = 1	(Разрешение сканирования каналов)
VCFG = 3	($ADV_{REFH} = V_{REF+}$, $ADV_{REFL} = V_{REF-}$)
ADCON2 = 0xC434	(VCFG — в разрядах 15..13, CSCNA — в разряде 10, SMPI — в разрядах 5..2, BUFM — в разряде 1, ALTS — в разряде 0)
ADCSSL = 0x154E	(Установлены в “1” разряды 12, 10, 8, 6 и 3–1 для дискретизации каналов 1–3, 6, 8, 10 и 12)

Пример 3. Комбинирование дискретизации с чередованием и сканирования каналов

Теперь допустим, что нам необходимо сканировать каналы 2, 5 и 7, чередуя эту группу с дискретизацией канала 14, с формированием запроса на прерывание после фиксации четырех выборок для каждого канала (т.е. всего — после 16 выборок). Данные будут получаться от преобразований в диапазоне напряжений от V_{REF-} до V_{REF+}, а на отрицательный вход схемы выборки/хранения будет подано напряжение V_{REF-}. В этом случае конфигурация модуля АЦП будет выглядеть следующим образом:

CH0SA = 0	(MuxA = 0, хотя это значение, по сути, роли не играет)
CH0NA = 0	(на Mux A подано опорное напряжение V_{REF-})
CH0SB = 0	(MuxB = 0, хотя это значение, по сути, роли не играет)
CH0NB = 0	(на Mux B подано опорное напряжение V_{REF-})
ADCHS = 0x0400	(CH0NB — в разряде 12, CH0SB — в разрядах 11..8, CH0NA — в разряде 4, CH0SA — в разрядах 3..0)
ALTS = 1	(Разрешение поочередной дискретизации)
BUFM = 0	(Не разбивать ADCBUF)
SMPI = 15	(Число выборок за прерывание — 16)
CSCNA = 1	(Разрешение сканирования каналов)
VCFG = 3	($ADV_{REFH} = V_{REF+}$, $ADV_{REFL} = V_{REF-}$)
ADCON2 = 0xC43D	(VCFG — в разрядах 15..13, CSCNA — в разряде 10, SMPI — в разрядах 5..2, BUFM — в разряде 1, ALTS — в разряде 0)
ADCSSL = 0x00A4	(Установлены в "1" разряды 7, 5 и 2 для дискретизации каналов 2, 5 и 7)

Итак, мы сконфигурировали порты ввода-вывода для конкретных требований приложения, выбрали источники опорного напряжения и определили, для каких каналов АЦП будут выполнены преобразования. У нас осталось еще два шага конфигурирования сигнальной цепи: определить частоту тактирования АЦП и инициатора преобразования.

Шаг 1г. Определение тактовой частоты АЦП

Каждый полный цикл оцифровки (снятие и преобразование сигнала в одном канале) требует одного цикла синхронизации АЦП (его период обозначается как T_{AD}) для получения уровня аналогового сигнала, по одному тактовому циклу АЦП на каждый разряд преобразования и еще

один цикла на то, чтобы переместить оцифрованное значение в ячейку ADCBUFx и произвести установки для подготовки к следующей оцифровке. Это означает, что для выполнения оцифровки одного канала 10-разрядному АЦП требуется 13 T_{AD}, а 12-разрядному — 15 T_{AD}. В большинстве моделей ЦКС dsPIC источник тактовых импульсов и их период можно конфигурировать программно, однако аппаратное обеспечение микросхемы требует, чтобы значение T_{AD} составляло как минимум 333,3 нс при питающем напряжении V_{DD} минимум 4,5 В или как минимум 666,7 нс, если V_{DD} ниже 4,5 В. Невыполнение этого требования приведет к тому, что считанные АЦП данные будут существенно занижены, поскольку усилителю выборки/хранения будет недостаточно времени для загрузки между сменой каналов.

Значение T_{AD} определяется в зависимости от T_{CY} (периода системной синхронизации) из уравнения:

$$T_{AD} = T_{CY} * (0{,}5 * (ADCS<5:0> + 1)) \qquad (3.1)$$

где T_{AD} — период синхронизации АЦП; T_{CY} — период системной синхронизации; ADCS<5:0> — числовое значение шестиразрядного регистра ADCS (ADC Clock Select — выбор тактирования АЦП).

Отсюда мы можем вычислить максимальную частоту дискретизации при V_{DD}>= 4,5 В:

f_{smax} = 1/(15 * T_{ADmin}) = 1/(15 * 333,3 нс) = 200 020 выборок в секунду.

Если V_{DD} < 4,5 В, то f_{smax} в два раза меньше этого значения, т.е. равно 100 010 выб/с.

Для того чтобы определить корректное значение, которое необходимо записать в регистр ADCS, следует просто решаем уравнение (3.1) относительно ADCS:

$$ADCS<5:0> = 2 * (T_{AD} / T_{CY}) - 1.$$

Например, если необходимо оцифровать все 16 каналов со скоростью 10 000 выб/с для каждого сигнала (совокупная частота дискретизации составляет 16 х 10 000 = 160 000 выб/с), то на это потребовалось бы в общем 160 000 х 15 = 2 400 000 тактовых циклов АЦП в секунду. Отсюда T_{AD} = 1 / 2 400 000 = 416,7 нс. Допуская, что ЦКС dsPIC работает на своей максимальной скорости 30 MIPS (млн. команд в секунду), период командного цикла T_{CY} = 1 / 30 000 000 = 33,33 нс. В таком случае, значение для регистра ADCS равно:

$$ADCS = 2 * (416{,}7 \text{ нс} / 33{,}3 \text{ нс}) - 1 = 24.$$

Важно заметить, что регистр ADCS поддерживает только шесть разрядов, а значит его значение ограничено числом 63. Таким образом, если ЦКС dsPIC, работающий на скорости 30 MIPS, выполняет 12-разрядное преобразование, то максимальное значение T_{AD} составляет

$$T_{ADmax} = 33{,}33 \text{ нс} * 0{,}5 * (63 + 1) = 1{,}07 \text{ мкс},$$

что соответствует минимальной частоте дискретизации:

$$f_{smin} = 1 / (15 * T_{ADmax}) = 1 / (15 * 1{,}07 \text{ мкс}) = 62\,507 \text{ выб/с}.$$

Если алгоритмы, используемые приложением, требуют меньшей скорости дискретизации, то приложение должно или прореживать значения выборок[17], или комбинировать методики опроса и управления по прерываниям. В последнем случае будет инициирована отдельная группа оцифровок для всех каналов, которая выполняется независимо с более высокой частотой, однако останавливается после однократной дискретизации всех каналов. Для создания следующей группы оцифровок с более низкой частотой, можно задействовать какой-нибудь другой механизм (например, программный таймер).

Последний подход не так уж и плох, как может показаться на первый взгляд, хотя он и влечет за собой дополнительный программный код. Этот метод позволяет нам выполнять дискретизацию во всех каналах относительно одновременно — возможность, которая очень выгодна для систем, где сигналы в одном канале могут быть связаны с другим каналом. Например, в руки робота часто встраивают так называемые *силомоментные датчики*, что позволяет роботу безопасно перемещать объекты без того, чтобы ронять (если не прилагается достаточной удерживающей силы) или повреждать их (если прилàгается слишком большая сила). Как сила, так и вращающий момент обычно измеряются в трех направлениях. Среди силомоментных датчиков очень популярна

---

[17] *Прореживание* — процесс, при котором данные, квантованные при одной частоте, преобразовываются в выборку, соответствующую меньшей частоте. Когда более высокая частота дискретизации превышает более низкую частоту в целое число раз, прореживание легко реализовать путем игнорирования выборок в потоке данных, номер которых не кратен заданному числу. Однако, если более высокая частота дискретизации выше не в целое число раз, необходимо вначале *интерполировать* данные потока, полученные на высокой скорости, для получения третьего потока данных с частотой дискретизации, равной общему наименьшему кратному исходной высокой частоты дискретизации и желаемой низкой частоты. После этого можно надежно проредить этот третий поток для извлечения необходимых низкочастотных данных.

разновидность, в которой измеряется нагрузка на трех, точно позиционированных физических "пальцах", так что сигналы от всех трех "пальцев" взаимосвязаны. Если сигналы измеряются со слишком большой разницей во времени между выборками в каждом канале, то система теряет информацию из-за того, что данные одного сигнала применяются к состоянию системы в один момент времени, а данные другого сигнала применяются к этому же состоянию системы с небольшим смещением во времени. Для получения максимально достоверной информации необходимо добиться как можно более одновременного измерения сигналов. Сокращая время между дискретизациями в каждом канале, мы можем существенно повысить эффективность системы, поскольку выборки в рамках одной группы становятся "более одновременными".

Шаг 1д. Выбор инициаторов дискретизации и преобразования

Заключительный шаг в процессе конфигурирования модуля АЦП — выбор инициаторов дискретизации и преобразования. Как уже было отмечено ранее, каждый цикл оцифровки состоит из фазы дискретизации, когда усилитель выборки/хранения накапливает уровень аналогового сигнала на его входе, и фазы преобразования, когда схема АЦП преобразует в числовое значение уровень аналогового сигнала, сохраненного в усилителе выборки/хранения. В завершение фазы преобразования оцифрованное значение сохраняется в соответствующей ячейке буфера ADCBUF. Контроллер dsPIC позволяет определить способ активизации каждой из этих двух фаз, предоставляя разработчику большую гибкость в выборе вариантов.

Для активизации процесса дискретизации можно выбрать один из двух вариантов:

- ручная инициализация, при которой приложение каждый раз, когда необходимо преобразовать входной сигнал, устанавливает соответствующий флаг;
- автоматическая инициализация, при которой дискретизация следующего канала начинается сразу же после завершения преобразования для текущего канала.

Ручная инициализация подходит, когда за наблюдаемым сигналом можно следить через непериодические отрезки времени (например, фоновое наблюдение за уровнем жидкости с целью отслеживания переполнения), или когда требуется частота дискретизации меньше, чем в автоматическом режиме преобразований. В качестве примера для последнего случая можно привести дискретизацию сигнала о температуре,

полученного от устройства с большой тепловой массой[18]. Поскольку изменение температуры — относительно медленное, приложение может производить выборки нечасто, чтобы сократить потребление ресурсов процессора. Хотя частота дискретизации — невысокая, для правильной реализации алгоритмов ЦОС выборки все же должны происходить периодически. Для этого приложение может выполнять оцифровку в подпрограмме обслуживания прерывания от таймера, возникающего с требуемым периодом.

Приложение выбирает инициатора дискретизации с помощью разряда ASAM (A/D Sample Auto-Start — автостарт дискретизации в АЦП) регистра ADCON1 (разряд 2). В том случае, если в ASAM записывается "0", модуль АЦП конфигурируется для ручной инициализации дискретизации. В таком случае каждый раз, когда необходимо квантовать входной сигнал, приложение должно устанавливать в "1" разряд SAMP (A/D Sample Enable — разрешение дискретизации в АЦП) регистра ADCON1 (разряд 1). И наоборот, при записи "1" в разряд ASAM модуль конфигурируется для выполнения автоматической дискретизации, когда квантование текущего уровня сигнала начинается сразу же по завершении предыдущего преобразования. При этом разряд SAMP устанавливается аппаратно, без вмешательства приложения.

Как при ручной, так и при автоматической дискретизации, усилитель выборки/хранения продолжает квантовать входной сигнал до тех пор, пока не начнется фаза преобразования.

Приложение определяет метод инициализации фазы преобразования (и, следовательно, — окончания фазы дискретизации), с помощью разрядов SSRC2.. SSRC0 (Conversion Trigger Source Select — выбор инициатора преобразования) регистра ADCON1 (разряды 7–5). Хотя три разряда SSRC позволяют определить восемь возможных инициаторов преобразования, три комбинации — зарезервированы (табл. 3.2).

Примечание

Однажды установленная комбинация разрядов SSRC имеет силу до момента отключения питания от микросхемы или записи в регистр ADCON1 новых данных.

---

[18] *Тепловая масса* — понятие, которым оперируют в терморегуляции. Тепловая масса подобна физической массе в механике. Чем она выше, тем труднее изменить температуру тела. В качестве примера объекта с большой тепловой массой можно привести большой лист стали, для нагрева которого требуется затратить значительную энергию, и на остывание которого требуется продолжительное время. Тонкая игла — объект с относительно малой тепловой массой, и потому она быстро нагревается и охлаждается.

Таблица 3.2. Соответствие комбинаций разрядов SSRC регистра ADCON1 инициаторам преобразования

SSRC<2:0>	Инициатор преобразования
000	Запись "0" в разряд SAMP
001	Активное переключение уровня на выводе INT0
010	Событие совпадения при сравнении в Timer 3
011	Истечение интервала ШИМ при управлении двигателем
100	Зарезервировано
101	Зарезервировано
110	Зарезервировано
111	Завершение счета внутренним счетчиком (автоматическое преобразование)

В случае ручной инициализации преобразования за определение окончания фазы дискретизации и активизацию фазы преобразования отвечает приложение. Для этого вначале в разряды SSRC записывается значение 000b, а потом обнуляется разряд SAMP. Затем приложение ожидает завершения преобразования, реализуя подходящую временную задержку, или следя за состоянием флага DONE (A/D Conversion Status — состояние преобразования АЦП) регистра ADCON1 (разряд 0), который устанавливается в "1" сразу же по завершении преобразования. Последующие выборки производятся путем повторения последовательности "SAMP-DONE".

Разработчик может также активизировать фазу преобразования наступлением одного (и только одного) из трех событий:

- активного переключение уровня на выводе INT0 (внешний источник);
- завершения счета таймером Timer 3 ЦКС dsPIC (внутренний источник);
- истечения интервала ШИМ при управлении электродвигателем (также внутренний источник).

Для того чтобы воспользоваться этим режимом, приложение должно записать в разряды SSRC комбинацию, соответствующую требуемому источнику события (001b, 010b или 011b), и сконфигурировать соответствующий ресурс (внешнее прерывание, таймер или ШИМ). Как только наступает выбранное событие, модуль прекращает дискретизацию текущего сигнала и начинает фазу преобразования.

Наконец, приложение может использовать режим автоматического преобразования, когда ЦКС dsPIC автоматически активизирует фазу преобразования по истечении заданного числа циклов тактирования

АЦП (от 1 до 32) от начала фазы дискретизации. В этом режиме для слежения за длительностью фазы дискретизации модуль использует внутренний счетчик, а фаза преобразования начинается по истечении определенного интервала (обозначенного как T_{AD}). Комбинируя автоматическую дискретизацию с автоматическим преобразованием, приложение может управлять системой в полностью автоматическом режиме, не требующего никакого вмешательства пользователя.

Контрольный перечень этапов для использования модуля АЦП

Для реализации преобразования с помощью модуля АЦП приложение должно выполнить следующие семь основных операций:

1. Сконфигурировать модуль АЦП для конкретных требований, предъявляемых к тракту сигнала, включая:
   - конфигурирование соответствующих выводов порта ввода-вывода в качестве аналоговых входов, выбор подходящих источников опорного напряжения и конфигурирование любых других выводов порта в качестве цифровых входов или выходов;
   - выбор входных каналов АЦП, которые будут преобразованы;
   - выбор требуемой частоты тактирования и инициаторов процессов АЦП;
   - активизация АЦП.
2. Сконфигурировать прерывания от АЦП, если приложение собирает данные по прерыванию:
   - сбросить флаг ADIF (A/D Interrupt Flag — флаг прерывания АЦП);
   - установить приоритет прерывания от АЦП.
3. Начать дискретизацию аналоговых данных.
4. Выждать необходимое время сбора данных.
5. Инициировать завершение сбора данных и начало преобразования.
6. Выждать завершение преобразования данных путем:
   - ожидания прерывания от АЦП;
   - отслеживания момента установки флага DONE.
7. Считать преобразованное значение(я) из буфера ADCBUF и обнулить флаг ADIF, если сбор данных производится по прерыванию.

Модуль таймеров/счетчиков

Модуль таймеров/счетчиков позволяет ЦКС dsPIC формировать точные внутренние синхроимпульсы, которые можно использовать для

реализации часов реального времени, для контроля и протоколирования особых программных событий или для подсчета переключений уровня определенных сигналов от внешних устройств. В зависимости от модели ЦКС dsPIC, доступны три или пять независимых таймеров/счетчиков, которые могут работать независимо или в особых предопределенных комбинациях, предоставляя чрезвычайную гибкость в распределении времени и подсчете событий.

Существует три основных конфигурации таймера/счетчика, которые обозначены в документации как типы A, B и C. Эти три типа обладает некоторыми общими характеристиками, однако каждый из них также имеет и уникальные свойства, рассчитанные на выполнение специфических задач. Мы вначале рассмотрим характеристики, общие для всех таймеров/счетчиков dsPIC, а затем исследуем отличительные свойства каждого из типов.

Обращаем внимание на то, что далее будут часто встречаться ссылки на регистры или названия выводов вида PRx, TMRx, TxCON и т.п. В таких случаях просто замените "x" требуемым номером таймера (1–5), чтобы получить реальное название регистра или вывода. Например, регистры таймера 1 называются PR1, TMR1 и T1CON, тогда как эти же регистры таймера 2 называются PR2, TMR2 и T2CON.

Общие характеристики таймеров/счетчиков

Каждый таймер/счетчик обладает следующим минимальным набором характеристик:

- 16-разрядный регистр периода PRx;
- 16-разрядный счетчик TMRx;
- 16-разрядный компаратор, который в каждом цикле синхронизации сравнивает значение в счетчике со значением в регистре периода;
- предварительный делитель, управляющий частотой обновления значения в регистре таймера/счетчика;
- два или больше вариантов выбора источников синхроимпульсов, один из которых — всегда системный генератор (T_{CY});
- возможность измерять промежуток времени, когда на соответствующем выводе синхронизации таймера присутствовал или отсутствовал импульс;
- возможность устанавливать флаг прерывания и формировать запрос на прерывание, когда компаратор обнаруживает совпадение значения счетчика со значением, сохраненным в регистре периода;
- регистр конфигурации таймера TxCON, который позволяет приложению конфигурировать работу соответствующего таймера.

Базовый принцип работы таймера действительно очень прост: приложение очищает счетный регистр TMRx, загружает необходимое значение в регистр периода PRx и активизирует таймер установкой флага TON (Timer On) в регистре управления TxCON. Начав работу, таймер приращивает содержимое своего счетного регистра счета на единицу по каждому нарастающему фронту синхроимпульса от выбранного источника синхронизации таймера. Когда значение в счетном регистре совпадает со значением в регистре периода, таймер устанавливает свой флаг прерывания и формирует запрос на прерывание (если оно разрешено). Установка флага прерывания также приводит к автоматическому обнулению счетного регистра без вмешательства со стороны программы, и описанный выше процесс повторяется до тех пор, пока пользователь не отключит таймер, обнулив флаг TON.

Хотя это обобщенное описание работы таймера само по себе корректно, в нем не были упомянуты некоторые важные аспекты, которые необходимо понимать для правильного конфигурирования таймера:

- выбор для таймера/счетчика подходящего генератора синхроимпульсов;
- работа предварительного делителя частоты синхронизации;
- вычисление значения для загрузки в регистр периода, которое зависит как от выбранного генератора синхроимпульсов, так и от коэффициента деления его частоты.

Рассмотрим эти три аспекта подробнее. Приложение может сконфигурировать каждый таймер для использования внутреннего синхросигнала, сформированного на основе системного синхросигнала (с периодом T_{CY}), или внешнего тактового генератора (с периодом T_{XCK}), подключенного к выводу TxCK. В любом случае, сигнал выбранного генератора проходит через предварительный делитель, который делит его частоту на 1, 8, 64 или 256 (т.е. период следования синхроимпульсов умножается соответственно на 1, 8, 64 или 256) до подачи на счетный регистр. Это дает разработчику возможность контролировать более длинные периоды времени за счет меньшего временного разрешения. Проиллюстрируем это простым примером.

Если в качестве источника синхронизации для таймера/счетчика назначить системный тактовый ЦКС dsPIC, работающего со скоростью 30 MIPS (T_{CY} = 33,33 нс), при коэффициенте деления 1 (PSF — Pre-scale Factor), то мы получим следующую разрешающую способность таймера:

$$T_{RES} = T_{CY} * PSF = 33{,}33 \text{ нс} * 1 = 33{,}33 \text{ нс},$$

и наш 16-разрядный счетный регистр будет поддерживать любой период вплоть до

$$T_{MAX} = 65\,536 * 33{,}33 \text{ нс} \approx 2{,}18 \text{ мс}.$$

С другой стороны, если мы используем тот же источник синхроимпульсов, но с коэффициентом деления 256, то разрешающая способность таймера ухудшается до

$$T_{RES} = T_{CY} * PSF = 33{,}33 \text{ нс} * 256 = 8{,}5325 \text{ мкс},$$

однако 16-разрядный счетный регистр теперь может поддерживать период до

$$T_{MAX} = 65\,536 * 8{,}5325 \text{ мкс} \approx 559{,}18 \text{ мс}.$$

После выбора источника синхроимпульсов и коэффициента предварительного деления значение, которое необходимо загрузить в регистр периода, вычисляется по простой формуле:

$$PR = T_{PERIOD} / T_{RES},$$

где PR — 16-разрядное значение, подлежащее загрузке в регистр периода PRx; T_{PERIOD} — желаемый период в секундах; T_{RES} — разрешающая способность таймера в секундах.

Разумеется, значения T_{PERIOD} и T_{RES} должны быть такими, чтобы полученное значение для регистра периода могло поместиться в 16-разрядный регистр, т.е. PR не должно превышать 65 535. Во многих приложениях именно это требование обуславливает выбор источника синхроимпульсов и значения предварительного делителя частоты.

Теперь, когда мы рассмотрели общие характеристики всех трех типов таймеров/счетчиков, перейдем к их отличительным особенностям.

Таймер/счетчик типа A

Таймер/счетчик типа A может применяться 16-разрядный таймер (о чем уже было сказано выше), синхронный счетчик или асинхронный счетчик. При работе в роли 16-разрядного синхронного счетчика, таймер/счетчик ведет себя во многом сходно с 16-разрядным таймером. Единственное отличие состоит в том, что в режиме таймера ожидается, что тактовые импульсы будут стабильными и периодичными, а в режиме счетчика импульсы, считываемые на соответствующем входе от внешнего источника, могут поступать апериодически. Аппаратное обеспечение таймера синхронизирует счетный сигнал по фазе с внутренним синхроимпульсом, после чего подает его на счетчик, что приводит к приращению содержимого счетного регистра на единицу по каж-

дому нарастающему фронту тактового сигнала. Как и в режиме таймера, при совпадении значения в регистре TMR1 со значением в регистре PR1, регистр TMR1 обнуляется, и модуль формирует запрос на прерывание.

Кроме режима асинхронного счетчика, уникальным для таймера/счетчика типа A является еще и возможность использовать в качестве внешнего источника тактовых импульсов маломощный генератор с частотой 32 кГц. Эти два свойства делают его идеальным для применения в качестве часов реального времени (RTC — Real-Time Clock) для приложений на базе ЦКС dsPIC, поскольку таймер может работать даже тогда, когда системный генератор синхроимпульсов неактивен. Это означает, что таймер продолжает работать, даже когда микросхема находится в "спящем" режиме, — ключевое требование для настоящих часов реального времени. В любом ЦКС dsPIC присутствует только один таймер типа A — Timer 1. Его структурная схема показана на рис. 3.11.

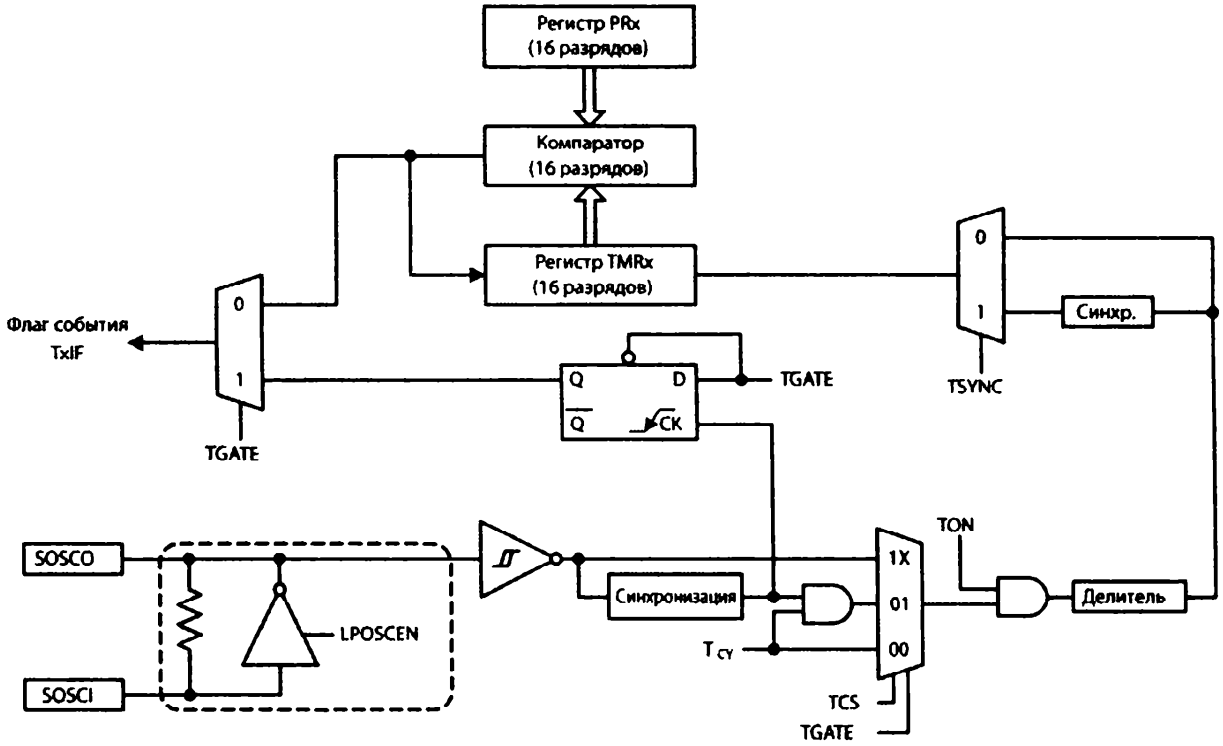


Рис. 3.11. Структурная схема таймера/счетчика типа A

В этой схеме следует отметить несколько важных моментов. Во-первых, ввиду того, что таймер 1 имеет специальную схему между входами T1CKI (Timer 1 Clock Input — тактовый вход) и SOSCI (Secondary Oscillator Input — вход вторичного генератора), для формирования синхроимпульсов таймера приложение может использовать внешний маломощный генератор с частотой 32 кГц.

Во-вторых, какой бы ни использовался источник синхроимпульсов, его сигнал перед необязательной синхронизацией с внутренним системным тактом проходит через предварительный делитель, чтобы можно было использовать входные сигналы синхронизации с частотой, по меньшей мере, 0,5 * (F_{CY} * *PSF*) (т.е. их частота могла превышать системную частоту синхронизации, умноженную на ½ * *PSF*). Однако, из-за временных ограничений комбинаторной логики, применяемой для реализации асинхронного тактового входа, частота внешнего синхросигнала таймера/счетчика типа A для его правильного функционирования всегда должна быть меньше 25 МГц (даже если это ограничивает частотный диапазон, которого можно было бы ожидать после деления частоты предварительным делителем). Разумеется, как и для всех типов таймеров/счетчиков ЦКС dsPIC, внешний синхросигнал должен соответствовать требуемой длительности переднего и заднего фронтов, которая составляет максимум 10 нс.

Теперь, когда мы познакомились с возможностями таймера/счетчика типа A, какие действия необходимо выполнить для его конфигурирования? Во-первых, мы должны определить его режим работы: таймер, синхронный счетчик или асинхронный счетчик (режим, применяемый для реализации часов реального времени). Задав рабочий режим, необходимо определить, когда таймер/счетчик должен формировать запрос на прерывание, т.е. при каком значении в счетчике должна быть прервана работа микросхемы.

Инициализация режима таймера

Для инициализации таймера 1 для работы в режиме таймера необходимо выполнить следующую последовательность действий.

1. Остановить таймер на время инициализации, обнулив управляющий разряд TON в регистре T1CON (разряд 15). Также следует сбросить флаг T1IF (Timer 1 Interrupt Flag) в регистре IFS0 и флаг T1IE (Timer 1 Interrupt Enable) в регистре IEC0, чтобы устранить все условия, которые могут привести к прерыванию от таймера 1. Это гарантирует, что последовательность инициализации не будет прервана.
2. Сбросить флаг TCS (Timer Clock Select — выбор источника тактирования таймера) в регистре T1CON (разряд 1), чтобы определить для подачи на вход предварительного делителя таймера сигнал от системного тактового генератора.
3. Задать значение для коэффициента деления частоты, сконфигурировав разряды TCKPS (Timer Clock Prescale) в регистре T1CON (разряды 5 и 4):

TCKPS = 00b — коэффициент 1:1 (отсутствие деления);
TCKPS = 01b — коэффициент 1:8;
TCKPS = 10b — коэффициент 1:64;
TCKPS = 11b — коэффициент 1:256.

4. Сбросить флаг TSYNC (Timer External Clock Input Synchronization Select — выбор синхронизации на входе от внешнего тактового генератора) в регистре T1CON (разряд 2), поскольку мы используем внутренний генератор синхроимпульсов и, таким образом, нет необходимости синхронизировать внешний синхросигнал.
5. Установить флаг TSIDL (Timer Stop in Idle) в регистре T1CON (разряд 13) в зависимости от того, необходимо ли приложению останавливать таймер в ждущем режиме, или нет:

   TSIDL = 0 — позволяет таймеру работать в ждущем режиме;
   TSIDL = 1 — останавливает таймер, когда микросхема переходит в ждущий режимю
6. Установить флаг TGATE (Timer Gated Time Accumulation Enable) в регистре T1CON (разряд 6) в зависимости от того, использует ли приложение режим стробированного временного накопления:

   TGATE = 0 — отключает режим стробированного временного накопления;
   TGATE = 1 — активизирует режим стробированного временного накопления (для этого режима также должен быть выбран внутренний источник синхроимпульсов, поэтому флаг TCS должен быть сброшен)
7. Записать в регистр периода PR1 значение, которое соответствует требуемому периоду возникновения прерываний (его расчет был представлен ранее в разделе "Общие характеристики таймеров/счетчиков").
8. Если приложение необходимо прерывать по завершению счета таймером (т.е. когда значение в регистре TMR1 равно значению в регистре PR1, заданному в п. 7, или по ниспадающему фронту сигнала на внешнем входе таймера, если активен режим стробированного временного накопления), то установите флаг T1IE в регистре IEC0 для разрешения прерываний от этого таймера. Также установите флаг общего разрешения прерываний, если он еще не установлен.
9. Запустите таймер записью "1" в разряд TON регистра T1CON.

На практике шаги 2–6 при инициализации обычно сводятся к выполнению единственной операции записи в регистр T1CON.

Инициализация режима синхронного счетчика

Режим синхронного счетчика похож на режим таймера с тем отличием, что сигнал, по которому производится счет, подается на внешний тактовый вход и синхронизируется с внутренним синхросигналом. В этом режиме принцип стробированного временного накопления не применяется, поэтому представленный ниже процесс очень похож на инициализацию режима таймера.

1. То же, что и п. 1 подраздела "Инициализация режима таймера".
2. Записать "1" в флаг TCS регистра T1CON, чтобы выбрать в качестве источника синхроимпульсов внешний источник.
3. То же, что и п. 3 подраздела "Инициализация режима таймера".
4. Установитm флаг TSYNC в регистре T1CON, чтобы синхронизировать входной синхросигнал с системным тактовым сигналом.
5. То же, что и п. 5 подраздела "Инициализация режима таймера".
6. Сбросить флаг TGATE в регистре T1CON, поскольку мы не используем режим стробированного временного накопления.
7. То же, что и п. 7 подраздела "Инициализация режима таймера".
8. То же, что и п. 8 подраздела "Инициализация режима таймера".
9. То же, что и п. 9 подраздела "Инициализация режима таймера".

Как и в случае инициализации режима таймера, выполнение пунктов 2–6 обычно сводится к выполнению единственной операции записи в регистр T1CON.

Режим асинхронного счетчика

Режим асинхронного счетчика идентичен режиму синхронного счетчика за тем исключением, что счетчик продолжает счет в "спящем" режиме, а на его тактовый вход может подаваться сигнал от маломощного кварцевого генератора с частотой 32 кГц. Если используется кварцевый генератор, то один его вывод подключается ко входу для подачи внешнего синхросигнала, а второй — к выводу SOSCI. Кроме того, должен быть установлен в "1" флаг LPOSCEN (Low-Power Oscillator Enable — разрешение маломощного генератора) в регистре OSCCON (Oscillator Control — управление генератором), чтобы включить схему, необходимую для управления формированием импульсов с помощью кварцевого генератора.

Как можно было бы предполагать, последовательность инициализации очень напоминает инициализацию режима синхронного счетчика.

1. То же, что и п. 1 подраздела "Инициализация режима синхронного счетчика".
2. То же, что и п. 2 подраздела "Инициализация режима синхронного счетчика".
3. То же, что и п. 3 подраздела "Инициализация режима синхронного счетчика".
4. Обнулить флаг TSYNC в регистре T1CON (разряд 2>) для асинхронного считывания внешнего синхросигнала.
5. То же, что и п. 5 подраздела "Инициализация режима синхронного счетчика".
6. То же, что и п. 6 подраздела "Инициализация режима синхронного счетчика".
7. То же, что и п. 7 подраздела "Инициализация режима синхронного счетчика".
8. То же, что и п. 8 подраздела "Инициализация режима синхронного счетчика".
9. То же, что и п. 9 подраздела "Инициализация режима синхронного счетчика".

Как и в других вариантах инициализации, пункты 2–6 обычно сводится к единственной операции записи в регистр T1CON.

Таймеры/счетчики типа B

Как и их собратья типа A, таймеры/счетчики типа B могут выполнять роль 16-разрядных таймеров и синхронных счетчиков. Однако, кроме этого, они также могут объединяться с таймером/счетчиком типа C для формирования 32-разрядного таймера или 32-разрядного синхронного счетчика. В такой комбинации таймер/счетчик типа B формирует младшие 16 разрядов 32-разрядного значения, а таймер/счетчик типа C — старшие 16 разрядов. Однако это повышение гибкости имеет определенную цену: таймеры/счетчики типа B не могут работать асинхронно и к ним нельзя подключить внешний кварцевый генератор. Впрочем, для большинства сфер применения это — не такие уж и серьезные ограничения.

В устройствах, поддерживающих таймеры/счетчики типа B, к этому типу всегда относятся таймеры 2 и 4 (если такой присутствует). Кроме того, при объединении таймеров/счетчиков типа B и типа C разрешены только определенные их комбинации. Таймер 2 всегда объединяется только с таймером 3, а таймер 4 — только с таймером 5.

Когда образован 32-разрядный узел, регистр TxCON (Timer Control) таймера/счетчика типа B служит управляющим регистром для всего 32-разрядного таймера/счетчика. Поскольку ЦКС dsPIC не поддерживает непосредственного обмена 32-разрядным данными с объединенными регистрами TMR3/TMR2 или TMR5/TMR4, микросхема использует регистр TMRxHLD для временного хранения значений из старшего слова регистра таймера/счетчика. Для того чтобы записать 32-разрядное значение в объединенный регистр таймера/счетчика, приложение вначале записывает старшие 16 разрядов в регистр TMRxHLD, а затем — младшие 16 разрядов в регистр TMRx таймера/счетчика типа B. Одновременно с записью в регистр TMRx ЦКС dsPIC загружает значение из регистра TMRxHLD в регистр TMRx соответствующего таймера/счетчика типа C. Считывание 32-разрядного значения из объединенного счетного регистра происходит в обратном порядке: когда приложение считывает младшее слово из регистра счетчика типа B, ЦКС dsPIC одновременно загружает старшее слово в регистр TMRxHLD. Затем приложение может выполнить чтение регистра TMRxHLD для извлечения старшего слова.

Структурная схема таймера/счетчика типа B показана на рис. 3.12.

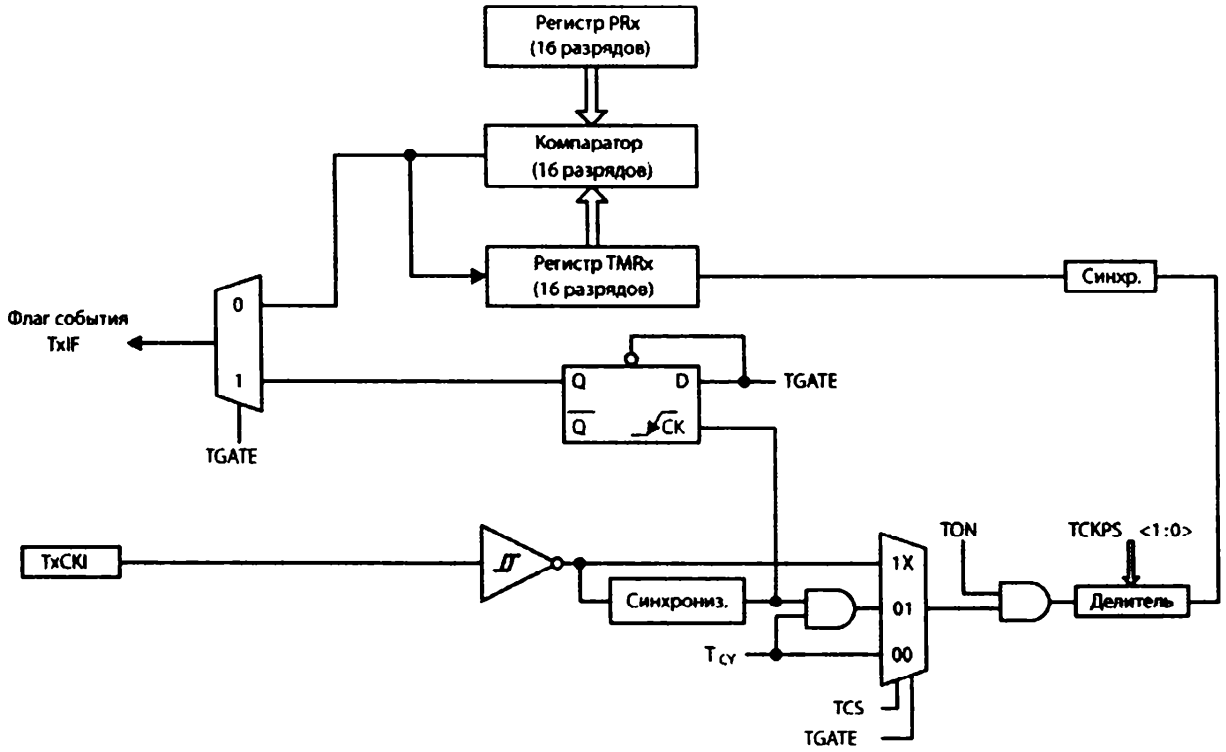


Рис. 3.12. Структурная схема таймера/счетчика типа B

Единственное существенное отличие этой структуры от структуры таймера/счетчика типа A — это отсутствие схемы формирования сигна-

ла, поддерживающей подключение кварцевого генератора, а также отсутствие тракта прохождения асинхронного тактового сигнала от выхода предварительного делителя до схемы счетчика. Как и в таймере/счетчике типа А, предварительный делитель таймера/счетчика типа В предшествует схеме синхронизации, а значит возможен прием внешних тактовых сигналов с частотой до 0,5 * F_{CY} * *PSF*. Поскольку во входной цепи сигнала тактирования отсутствует комбинаторная логика, нет ограничения максимальной частоты 25 МГц, как в случае с таймером/счетчиком типа А.

Таймеры/счетчики типа В имеют дополнительный флаг, определенный в регистре TxCON, который отсутствует в этом же регистре таймеров/счетчиков типа А или С. Приложение должно записать "1" в разряд T32 (32-bit Timer Mode Select — выбор режима 32-разрядного таймера), чтобы объединить таймер/счетчик типа В с соответствующим таймером/счетчиком типа С, или обнулить этот разряд, чтобы эти два таймера/счетчика функционировали как независимые 16-разрядные элементы. Поскольку таймеры/счетчики типа В всегда работают в синхронном режиме, отпадает необходимость в разряде TSYNC в регистрах TxCON, поэтому данный флаг не определен.

Инициализация 16-разрядного таймера и 16-разрядного синхронного счетчика

Процедура инициализация таймера/счетчика типа В для работы в качестве 16-разрядного таймера или счетчика идентична процедуре соответствующей инициализации таймера/счетчика типа А за тем исключением, что в регистре TxCON должен быть обнулен разряд T32, а разряд TSYNC вообще отсутствует.

Инициализация 32-разрядного таймера или синхронного счетчика

Процедура инициализации таймера/счетчика типа В для работы в 32-разрядном режиме идентична процедуре его инициализации как 16-разрядного таймера за тем исключением, что в регистре TxCON должна быть записана "1" в разряд T32, а также должен быть задействован флаг разрешения прерывания от таймера/счетчика. Обратите внимание на то, что запись производится в регистр PR как для таймера/счетчика типа В, так сопряженного с ним таймера/счетчика типа С.

Таймеры/счетчики типа С

Таймеры/счетчики типа С сходны с таймерами/счетчиками типа В и имеют те же основные режимы работы, однако структура на рис. 3.13

показывает, что схема синхронизации внешнего тактового сигнала размещена перед предварительным делителем, поэтому максимальная частота внешнего тактового сигнала для таймера/счетчика типа C составляет 0,5 * F_{CY} (в отличие от потенциально более высокой частоты 0,5 * F_{CY} * *PSF* у таймеров/счетчиков типа B). Как и в случае с таймерами/счетчиками типа B, отсутствие асинхронной комбинаторной логики на входе внешнего тактового сигнала исключает ограничение по максимальной тактовой частоте 25 МГц, как в случае с таймером/счетчиком типа A.

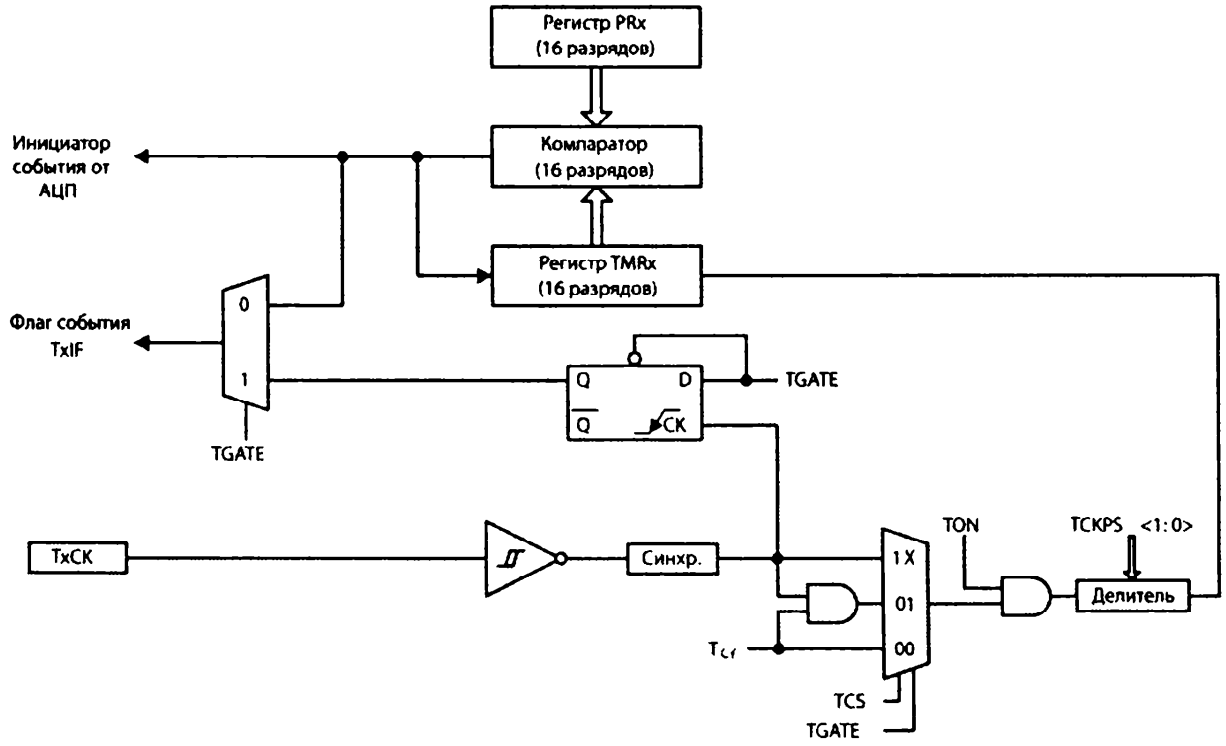


Рис. 3.13. *Структурная схема таймера/счетчика типа C*

Процедура инициализации таймеров/счетчиков типа C практически идентична процедуре инициализации таймеров/счетчиков типа B для каждого из четырех рабочих режимов с одним важным отличием. Используя таймер/счетчик типа C в 16-разрядном режиме, флаг T32 в регистре управления объединенного с ним таймера/счетчика типа B должен быть сброшен, чтобы перевести оба таймера/счетчика в 16-разрядный режим.

3.4. Резюме

В этой главе архитектура ЦКС dsPIC была рассмотрена довольно подробно, однако, разумеется, — далеко не в полной мере с точки зрения возможностей этой микросхемы. Дополнительную информацию по

этому вопросу можно найти в документах от компании Microchip: "dsPIC30F Family Reference Manual" и "dsPIC30F Programmer's Reference Manual", а также в технических описаниях конкретных моделей dsPIC. Все эти документы можно найти на официальном Web-сайте компании `www.microchip.com` наряду с многочисленными рекомендациями по применению, советами по разработке и Internet-семинарами, которые описывают и анализируют конкретные компоненты системы и их применение для решения реальных задач.

В следующей главе мы продолжим исследование периферийных модулей, доступных в ЦКС dsPIC, но на этот раз углубимся в изучение коммуникационных ресурсов микросхемы.

4

Умение быть хорошим коммуникатором

Чем выше вы поднимаетесь, тем шире ваш круг общения, который может или поднять, или погубить вас.

— Дональд Уолтон

Есть один аспект, который пронизывает все, что мы делаем с интеллектуальными сенсорными системами, и который на самом деле во многом определяет качество этих систем. Речь идет о способности обмениваться данными с другими компонентами — как внутри локальной среды, так и дистанционно. Как будет показано позже, ЦКС dsPIC предлагает обширный спектр средств связи с возможностью выбрать подходящий коммуникационный метод в зависимости от характера данных, а также задать скорость и физическую среду передачи. Данная глава разделена на два основных раздела, в первом из которых изложена вводная информация относительно коммуникационных требований, часто возникающих в областях применения, связанных с интеллектуальными датчиками. Во втором разделе главы рассматриваются коммуникационные интерфейсы, предоставляемые семейством устройств dsPIC, определяются ситуации, в которых лучше всего применить тот или иной интерфейс, а также разъясняется использование соответствующих аппаратных модулей.

4.1. Виды связи

Приступая к исследованию вопроса организации связи, хотелось бы сразу отметить, что для интеллектуальных датчиков не существует какой-либо "правильной" формы связи — точно так же, как не существует какой-то одной наилучшей формы общения для людей. Когда мы хотим высказать что-либо, то подсознательно регулируем громкость нашего голоса и скорость речи, а также определяем количество передаваемой информации, учитывая ее важность, окружающую обстановку и характер собеседника. Если вы блуждаете залами Лувра с четырехлетней пле-

мянницей, то вполне достаточным комментарием для нее будет фраза "красивая картина", но та же самая экскурсия с другом-живописцем может вылиться в долгую дискуссию о движениях кисти, красках и стиле. В обоих случаях беседа формируется с учетом возможностей одновременно говорящего и слушающего — в терминах как количества передаваемой информации, так и уровня ее детализации.

То же самое справедливо и для ситуации с интеллектуальными датчиками. Задействованные методы связи зависят от типа передаваемых данных, их срочности, а также физической среды, используемой для их передачи от источника к приемнику. Так, например, команды конфигурирования передают от центральной системы в сенсорную, как правило, до того, как сенсорная система начнет сообщать результаты параметрических измерений. Если объем командных данных не будет чрезмерно большим, то скорость их передачи в данном случае не играет особой роли, чего нельзя сказать о требовании сохранения точной последовательности передаваемых команд, т.е. в данном случае связь не особо критична ко времени. На противоположном конце спектра могут быть распознанные датчиком аварийные условия, информация о которых должна быть доставлена адресату надежно и достаточно быстро, чтобы центральная система смогла надлежащим образом отреагировать и предотвратить нежелательные (и, возможно, опасные) рабочие состояния. Само собой, последняя ситуация требует более надежного и скоростного метода связи, чем предыдущая. К счастью, ЦКС dsPIC предлагает несколько различных коммуникационных интерфейсов, способных восполнить требования самых разнообразных задач по передаче данных.

Определение характеристик канала связи

Мы уже затронули некоторые важные характеристики каналов связи, и сейчас — самое время рассмотреть их подробнее. Это позволит нам лучше классифицировать различные подходы и даст некий критерий, на основании которого мы сможем оценивать различные варианты организации связи. Далее будем исходить из предположения об использовании проводных электронных каналов связи (в противоположность оптическим, беспроводным или другим средствам). Это предположение не следует воспринимать как "естественное", поскольку существует множество систем, использующих оптическую или беспроводную передачу данных, и мы избираем проводную связь, только исходя из возможностей ЦКС dsPIC.

Перечислим ключевые критерии для принятии решения относительно организации связи:

- требуемая непрерывная (в противоположность пакетной) пропускная способность канала связи, выраженная в единицах бит/с;
- должна ли связь быть двухточечной (т.е. только между двумя устройствами) или многоточечной (т.е. в одном канале связи присутствует более двух устройств);
- физическая длина канала связи и уровень электрического шума в нем;
- должен ли канал связи быть синхронным, асинхронным или любым из этих двух вариантов;
- требуется ли аппаратный контроль ошибок.

Рассмотрим каждый из этих критериев подробнее прежде, чем перейти к изучению коммуникационных интерфейсов, доступных в ЦКС dsPIC .

Пропускная способность канала передачи данных

Одна из наиболее важных коммуникационных характеристик — это *пропускная способность канала передачи данных*, т.е. то объем данных, которые могут быть переданы за заданный период времени. Существуют два принципиально различных вида пропускной способности: *пакетная* и *непрерывная*. Очень важно понимать различие между этими двумя терминами и знать, который из них является ограничивающим фактором в конкретной сфере применения.

Пакетная пропускная способность — это самая быстрая скорость передачи, которую может поддерживать канал для небольших порций данных (пакетов). Однако пакетной пропускной способности, самой по себе, недостаточно. Разработчик также должен знать максимальный объем данных, который может быть передан на этой скорости. Хотя пакетная пропускная способность может быть очень высокой, этот режим не годится, если он не в состоянии передавать достаточно большие пакеты данных для соответствия требованиям конкретной ситуации.

Непрерывная пропускная способность — это максимальная скорость, поддерживаемая каналом для непрерывной передачи данных. Обычно это — более предпочтительная единица измерения пропускной способности канала, хотя она и может оказаться значительно ниже, чем пакетная пропускная способность, поскольку отражает гарантированную скорость, независимо от объема передаваемых данных.

В наших проектах сенсорных систем мы будем опираться на непрерывную пропускную способности канала, хотя те случаи, в которых пакетный режим может оказаться более приемлемым, также будут оговорены.

Сравнение двухточечных и многоузловых сетей

Системы связи относятся к одной из двух категорий. Системы, использующие двухточечную топологию[19], содержат, по сути, два узла, которые обмениваются данными (рис. 4.1).

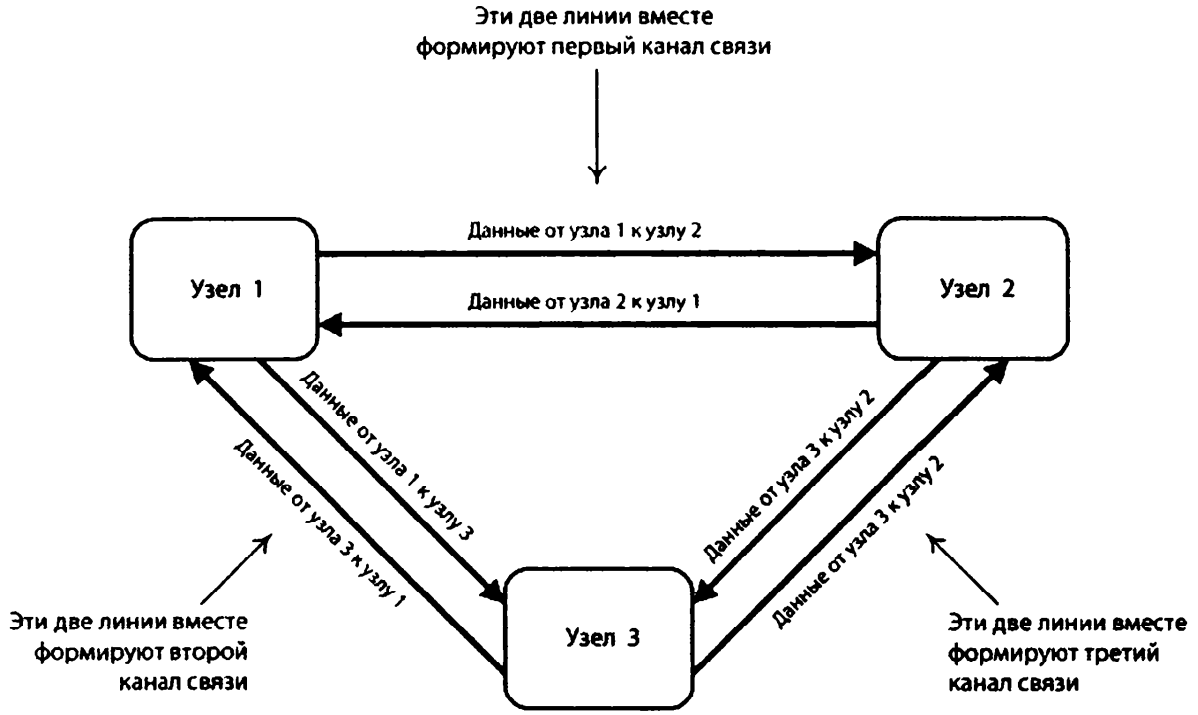


Рис. 4.1. Двухточечная коммуникационная сеть

Хотя можно использовать повторители, передающие данные от одного узла к другому, ключевым момент заключается в том, что в каждом канале связи существует только одна пара "передатчик-приемник" для каждого направления передачи. Это упрощает задачу разработчика, поскольку, ввиду наличия единственного передатчика, ему не нужно беспокоиться о распределении доступа на передачу данных по каналу связи. Кроме того, использование двухточечной топологии значительно упрощает концевую заделку кабелей, которая является обязательным требованием для протяженных линий связи.

Основной недостаток двухточечных систем — ограниченная способность использовать данные совместно с другими системами. Образно говоря, им трудно разговаривать даже с одним собеседником, и потому во многих системах используют *многоузловую* топологию, в которой несколько узлов соединены одним каналом связи (рис. 4.2).

---

[19] В данном случае *топология* — это просто технический термин, описывающий расположение узлов сети.

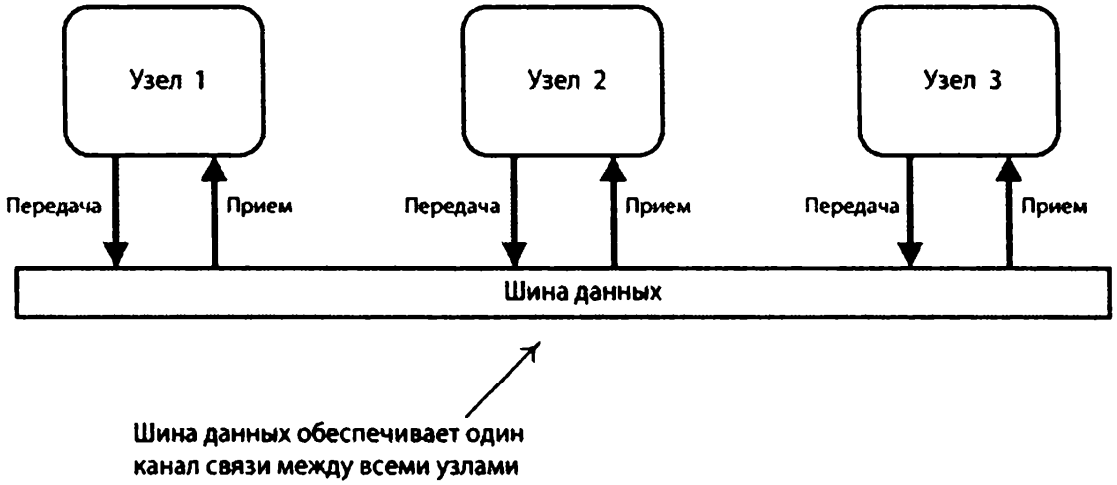


Рис. 4.2. Пример многоузловой сети

При таком подходе из одного узла можно обращаться ко всем узлам сразу или устанавливать связь между несколькими узлами. Однако обратите внимание на то, что при наличии многих *принимающих* узлов одновременно с этим только один узел может фактически *передавать* данные. Так возникает одна из основных проблем рассматриваемой топологии: планирование передачи из отдельных узлов таким образом, чтобы несколько узлов не пытались передавать данные одновременно.

Для того чтобы избежать *конфликтов* передачи (т.е. ситуаций, когда несколько передатчиков пытаются одновременно отправить данные), обычно используют один из двух подходов. Первый из них заключается в попытке полностью избежать конфликтов путем передачи по сети программного или аппаратного *маркера*, который извещает отдельный передатчик о возможности безопасно передавать данные. Когда передатчик получает маркер, он может передавать любые данные, а затем передать маркер следующему узлу в сети. Если маркер оказывается у узла, не имеющего никаких данных для передачи, то он просто передается следующему узлу без выполнения каких-либо действий.

Хотя этот метод и предотвращает конфликты, он делает это за счет дополнительного программного обеспечения и/или аппаратных средств, необходимых для последовательной передачи маркера по сети. При этом могут проявляться (относительно) длительные периоды времени, когда маркер передается между узлами, не имеющими данных для передачи, в то время как узел, имеющий такие данные, вынужден ожидать.

Второй подход использует несколько иной метод. Вместо того чтобы избегать конфликтов, он их допускает, однако быстро и надежно их распознает, после чего пытается перепланировать передачу для участников конфликта таким образом, чтобы доминирующий узел (т.е. тот,

который первым начал передачу) завершил передачу данных, а другие в это время ожидали своей очереди. Этот подход может значительно повысить пропускную способность в тех сетях, где отдельные узлы должны передавать данные апериодически, поскольку разрешает узлам предпринимать попытки передачи сразу же после освобождения канала, без длительных периодов ожидания. Конечно же, алгоритм разрешения конфликтов в таком случае должен гарантировать, что все узлы в конечном счете получат право на передачу.

Физические свойства канала передачи данных

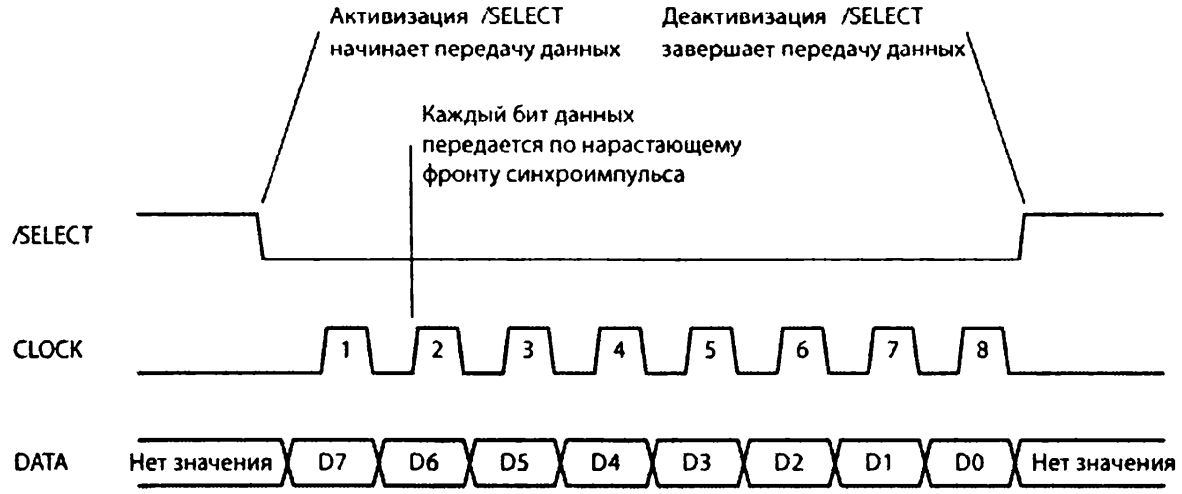
Такие физические свойства канала связи, как протяженность и уровень электрического шума его рабочей среды, ограничивают максимальную скорость передачи и могут потребовать использования специальных схем для адекватного управления электронными сигналами и/или обнаружения ошибочных ситуаций. Эти проблемы можно в некоторой мере преодолеть с помощью дифференциальных сигналов, однако дополнительные требования по потребляемой мощности и финансовые издержки в этом случае могут свести на нет все "плюсы" такого метода. Само собой разумеется, чем длиннее линия связи или чем сильнее зашумлена среда, тем медленнее будут передаваться данные (если, конечно, мы хотим обеспечить их надежную доставку).

Асинхронная и синхронная передача данных

В общем случае данные могут быть переданы *синхронно* с сопровождающим их синхросигналом или же *асинхронно*, т.е. без сопровождающего их синхросигнала. Как всегда, каждый из методов имеет свои достоинства и недостатки, и выбор одного из них зависит от конкретной ситуации.

Преимущество синхронной передачи заключается в том, что передатчик может генерировать синхросигналы, точно сообщающие приемнику, когда следует произвести выборку входных данных. Это значительно упрощает точное распознавание логических "0" и "1" (если, конечно, предположить, что мы имеем дело с двоичными данными) на принимающей стороне и гарантирует правильную последовательность информационного обмена. Однако за удобства приходится платить. Кроме того, что линия должна обеспечивать передачу дополнительного синхросигнала (или двух, если синхросигнал посылается дифференциальным образом), становится крайне важным "чистота" синхросигнала, поскольку любой значительный "всплеск" в нем может быть ошибочно интерпретирован как переключение синхроимпульса. Что еще хуже, с

целью четкой идентификации соответствующих значений битов данных синхросигнал обычно имеет крутые фронты, что генерирует высокочастотный шум, который может превысить допустимые пределы. Пример простой схемы синхронной передачи показан на рис. 4.3.



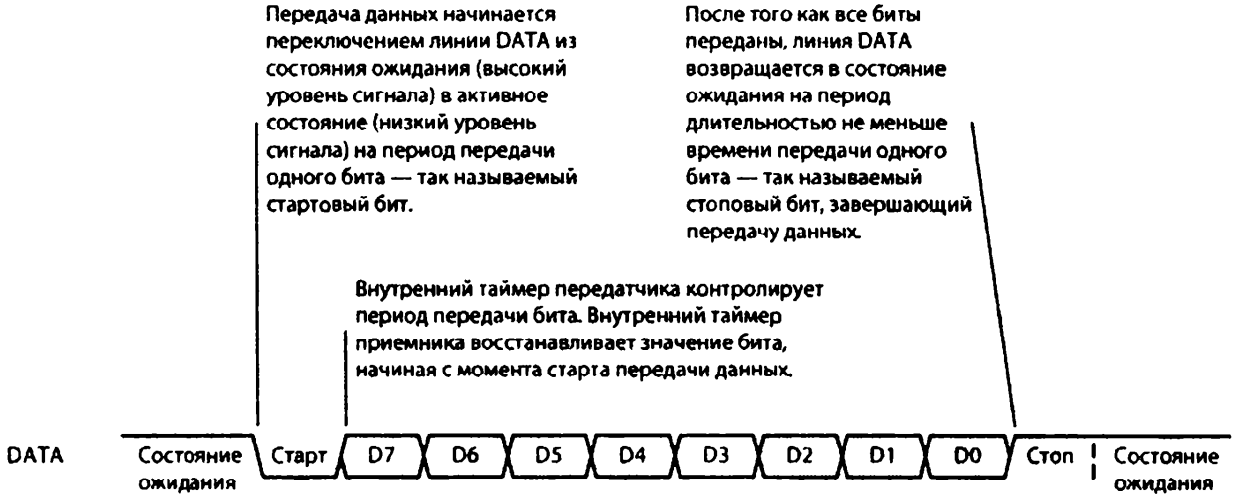
Ключевые моменты

1. Сигнал CLOCK синхронизирует передачу данных, обеспечивая их отправку передатчиком и считывание приемником по одному и тому же сигналу.
2. Сигнал /SELECT определяет передачу данных (т.е. идентифицирует ее легитимность). Без этого спецификатора всплески в сигнале CLOCK могли бы привести к ошибкам передачи.
3. Полярность синхроимпульсов и фаза на диаграмме показаны в иллюстративных целях и в фактической реализации могут отличаться. Кроме того, схема следования сигналов на практике может оказаться более сложной. Например, многие стандарты синхронной передачи используют отдельные сигналы "Data Receive" ("Прием данных") и "Data Transmit" (Передача данных), позволяющие передавать данные одновременно в обоих направлениях.

Рис. 4.3. Пример схемы синхронной передачи данных

Если принять во внимание все вышеупомянутые проблемы, то может сложиться впечатление, что наилучший вариант — асинхронная передача данных, и зачастую это действительно так. Большим преимуществом асинхронной линии связи является то, что она не требует никакого внешнего синхросигнала — синхроимпульс, требуемый для декодирования поступающих битов, содержится в самих данных. Это устраняет все трудности, связанные со внешним синхросигналом, включая прокладку для него отдельного кабеля. Тем не менее, читатель уже, наверное, понял, что за подобное "счастье" обычно приходится чем-то платить, и данный случай — не исключение. Точное восстановление синхроимпульса, внедренного в поток данных, требует синхронизации вы-

сокоскоростного синхрогенератора в приемнике (частота которого в четыре–шестнадцать раз выше частоты, используемой при формировании данных) со входным потоком данных, специальной дополнительной аппаратуры и эффективного ограничения верхнего предела скорости передачи значением, которое меньше (и зачастую — намного) допустимой скорости в синхронной системе. Пример асинхронной передачи сигналов иллюстрирует рис. 4.4.



Ключевые моменты

1. Как передатчик, так и приемник содержат в высокой степени согласованные внутренние тактовые генераторы, которые синхронизируются по ниспадающему фронту стартового бита. В случае слишком большой разницы между частотами этих генераторов приемник будет производить считывания в некорректные моменты времени и получит ошибочные данные.
2. Синхронизация потока данных улучшается за счет обязательного присутствия в линии DATA начальных и конечных состояний (обозначены здесь как стартовый и стоповый биты). Любая передача данных, не соответствующая этим минимальным требованиям, может быть распознана как некорректная.
3. Добавив в поток данных дополнительные информационные биты, состояния которых определяются значениями конкретных передаваемых данных, можно реализовать механизм обнаружения ошибок и, возможно, — их исправления. Если приемник определяет, что значения считанных им дополнительных битов не соответствуют ожидаемым, то он может пометить передачу как ошибочную. Биты, используемые для обнаружения ошибок, обычно называются битами контроля четности.
4. Специфические аспекты этого примера, включая уровни сигнала в линии DATA, используемые для идентификации конкретных состояний и порядка следования битов данных, представлены исключительно для иллюстративных целей, и в фактических реализациях могут отличаться. Кроме того, реальная схема передачи сигналов может оказаться более сложной. Например, многие стандарты асинхронной передачи данных предусматривают использование отдельных сигналов "Data Receive" ("Прием данных") и "Data Transmit" (Передача данных), позволяющих передавать данные одновременно в обоих направлениях.

Рис. 4.4. Пример асинхронной передачи сигналов

Аппаратное обнаружение ошибок

Какие бы меры предосторожности не были предприняты для того, чтобы избежать ошибок, их возникновение в любой коммуникационной системе неизбежно. Учитывая эту объективную реальность, разработчик обязательно должен встроить в систему средства, позволяющие если и не откорректировать возникшие ошибки, то по крайней мере обнаружить их, чтобы ошибочные данные не привели к нежелательному поведению системы. В зависимости от быстродействия канала передачи данных, обнаружение ошибок можно реализовать программно, хотя и за счет дополнительной нагрузки на процессор.

Однако в некоторых случаях из-за быстродействия канала связи или требований надежности для обеспечения корректной связи между передатчиком и приемником может потребоваться аппаратная реализация обнаружения ошибок. К сожалению, такие аппаратные решения, как правило, сложны и дорогостоящи, если только не входят в состав стандартных коммуникационных устройств. Как вскоре будет показано, ЦКС dsPIC содержит, как минимум, коммуникационный интерфейс CAN (Control Area Network). Этот интерфейс полностью соответствует вышеупомянутым критериям и обеспечивает высокую степень надежности канала связи, характерную для аппаратного обнаружения ошибок.

4.2. Коммуникационные интерфейсы, доступные в dsPIC30F

Контроллер dsPIC предоставляет разнообразные интерфейсы, поддерживающие обмен данными как с устройствами на той же печатной плате, так и с удаленными системами:

- SPI — последовательный периферийный интерфейс;
- I^2C — межсхемный интерфейс интегральных схем;
- UART — универсальный асинхронный приемопередатчик;
- CAN — интерфейс области управления.

Мы подробно рассмотрим интерфейсы SPI, UART и CAN. Интерфейс I^2C использует многие из принципов интерфейса SPI, поэтому отдельно здесь не рассматривается.

Порт интерфейса SPI

Последовательный периферийный интерфейс SPI — синхронный, и предназначен в первую очередь для передачи данных между устройствами, размещенными на одной печатной плате (хотя и не обязатель-

но). Этот интерфейс довольно прост. В нем используются следующие сигналы:

- SDO (Serial Data Out) — отправка последовательных данных;
- SDI (Serial Data Input) — прием последовательных данных;
- SCK (Serial Clock) — последовательные синхроимпульсы;
- CS (Chip Select) — выбор кристалла;
- SS (Slave Select) — выбор ведомого устройства.

Для каждого из этих сигналов отводят отдельную цифровую линию — одна из причин, по которой интерфейс SPI не подходит для создания длинных каналов передачи данных и для работы в среде, насыщенной электрическими шумами. Тем не менее, этот интерфейс используется многими устройствами — как микропроцессорами, так и микросхемами периферийных устройств — поскольку связь по SPI очень легко реализовать и контролировать. Например, плата dsPICDEM использует один из двух портов SPI для обмена данными со встроенным датчиком температуры, посылая на датчик конфигурационные данные, и считывая показания температуры и значения регистра состояния.

С годами интерфейс SPI эволюционировал и стал поддерживать четыре основных режима работы, которые отличаются только временны*м* соотношением между фронтом синхроимпульса SCK и сигналами данных SDO и SDI для определения момента передачи данных и их корректности на принимающей стороне. Большинство устройств поддерживает только некоторые из этих режимов, поэтому важно удостовериться в том, чтобы передающее и принимающее устройства имели хотя бы один общий режим работы.

Четыре режима интерфейса SPI показаны на рис. 4.5. В наших примерах мы используем режим 0, поскольку ему соответствует одна из наиболее популярных конфигураций интерфейса.

Компания Microchip предоставляет прекрасную 16-разрядную библиотеку обслуживания периферийных устройств, содержащую полезные функции контроля и доступа к портам SPI в ЦКС dsPIC. В отличие от интерфейсов для UART и шины CAN, которые мы рассмотрим чуть позже, SPI обычно используют для передачи небольших порций данных (зачастую — один байт или слово). Поскольку скорость передачи данных по SPI очень высока, ее, фактически, можно трактовать как встроенную операцию реального времени, которая завершается в пределах времени выполнения соответствующего программного кода. Например, если мы считываем два байта данных о температуре из датчика, подключенного через интерфейс SPI, то можем позволить себе ожидание ответа на запрос, поскольку передача данных работу значительно не за-

медлит. Совершенно иначе обстоит дело при пересылке больших объемов данных через другие коммуникационные порты (или даже через SPI). Обычно в такой ситуации необходимо реализовать буферизированную, управляемую по прерываниям структуру. К счастью, в 16-разрядной библиотеке обслуживания периферийных устройств от компании Microchip уже реализована вся функциональность, требуемая для использования порта SPI.

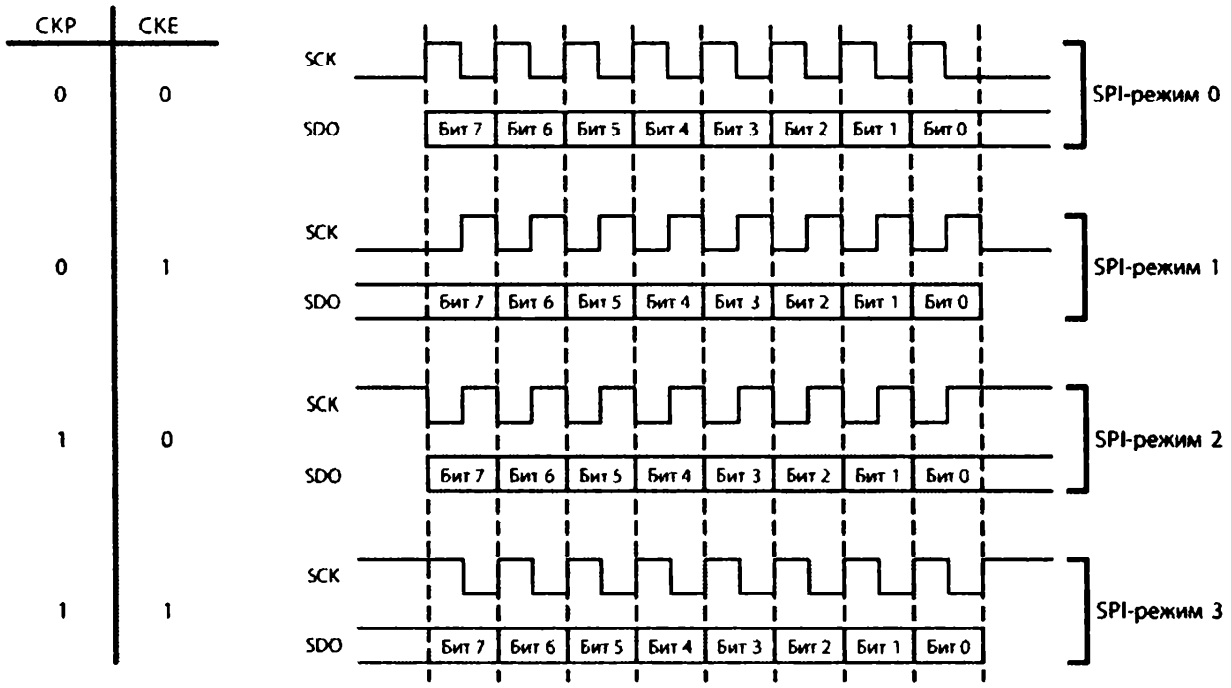


Рис. 4.5. *Четыре режима интерфейса SPI*

Для того чтобы воспользоваться портом SPI, вначале необходимо сконфигурировать соответствующий модуль dsPIC, задав скорость передачи данных, полярность и фазу синхроимпульсов (эти два параметра, по сути, определяют режим работы SPI), а также приоритетность прерывания.

Приемопередатчик UART

В то время как интерфейсы SPI и I^2C — синхронные, приемопередатчик UART использует асинхронную передачу данных. UART, наверное, — наиболее популярное средство связи (по крайней мере, — между компьютерами), и наилучшим примером его использования является старый добрый последовательный порт RS-232, который вплоть до недавнего времени был неотъемлемой частью любого персонального компьютера. Последовательные порты RS-232, кроме того, широко используются и в промышленных системах. Это был один из самых первых

и распространенных интерфейсов наряду с его близкими "родственниками": интерфейсами RS-422 и RS-485 (по сути — дифференциальные версии того же базисного интерфейса). Можно уверенно сказать, что буква "U" в слове "UART" — вполне заслуженна, учитывая универсальность этого средства обмена данными.

Устройства на базе UART могут работать в составе как двухточечной, так и многоузловой топологии при условии, что будут использованы дополнительные схемы управления, способные блокировать передатчик каждого узла. Для организации связи по интерфейсу UART достаточно трех сигналов:

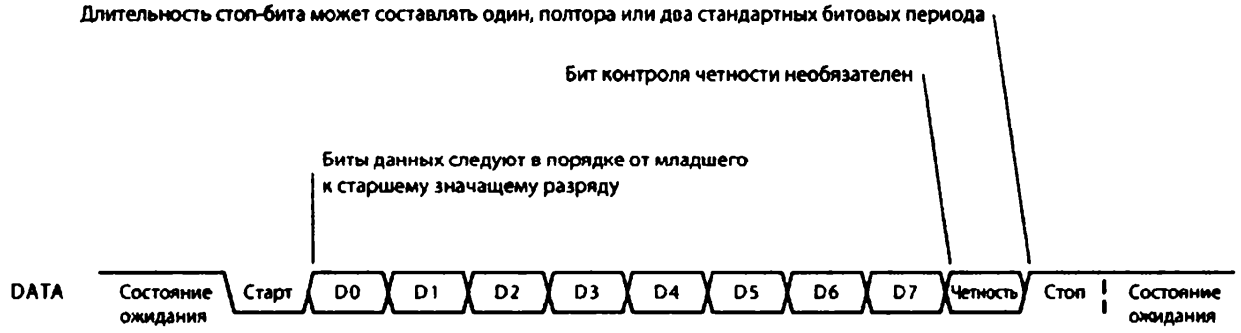
- RxD (Receive Data) — прием данных;
- TxD (Transmit Data) — передача данных;
- сигнал "земли", служащий опорным для предыдущих двух сигналов.

На практике сигнал RxD одного узла соединен с сигналом TxD другого узла и наоборот. Как уже было отмечено выше, асинхронные линии связи для обеспечения корректного извлечения информационных битов внедряют в передаваемые данные синхросигнал или хотя бы предлагают некоторый способ синхронизации быстродействующего внутреннего генератора приемника со входным потоком данных. В случае с интерфейсом UART эта синхронизация реализована путем вставки в начале каждого байта данных стартового бита. Сами данные следуют после стартового бита, начиная с младшего и заканчивая старшим значащим разрядом. После данных следует необязательный бит контроля четности и стоповый период длительностью в один, полтора или два периода передачи информационного бита. Пример передачи данных через UART показан на рис. 4.6.

Обратите внимание, что неактивное состояние линии RxD соответствует высокому уровню сигнала. Стартовый бит низкого уровня выводит линию из состояния ожидания и сигнализирует приемнику о том, что сейчас последуют данные. В результате UART активизирует свой внутренний генератор побитной синхронизации, а также аппаратный конечный автомат, который идентифицирует время выборки каждого бита из входящего потока данных. Очевидно, что оба конца линии связи должны быть сконфигурированы так, чтобы работать на одной и той же скорости, иначе схемы выборки на принимающей стороне будут считывать отдельные разряды в неправильные моменты времени, в результате чего принятые данные окажутся искаженными.

Необязательный бит контроля четности может использоваться для базового обнаружения ошибок передачи, хотя обычно его просто игно-

рируют. Значение этого бита устанавливается передатчиком, а приемник должен проверить его соответствие содержимому переданных данных.



Ключевые моменты

1. Передача данных через UART характеризуется следующими параметрами:
   А. Скорость передачи — в битах за секунду.
   Б. Тип четности — способ формирования бита контроля четности.
   В. Объем данных — число бит, передаваемых за одну передачу.
   Г. Количество стоповых битов.
2. Скорость передачи задает ширину каждого бита данных и измеряется в числе битов, которые канал связи может передать за секунду. Некоторые распространенные значения скорости передачи: 9 600 бит/с, 19,2 Кбит/с, 38,4 Кбит/с, 57,6 Кбит/с и 115,2 Кбит/с. Хотя не существует никаких жестких требований по использованию именно этих значений скорости передачи (самое главное, чтобы приемник и передатчик использовали одну и ту же скорость), для обеспечения связи с другими системами лучше выбирать одно из них.
3. Три типа контроля четности: на четность, на нечетность и отсутствие контроля. В случае контроля на четность контрольный бит содержит "1", если данные (не включая стартовый и стоповые биты) содержат четное число единичных разрядов, и "0", если данные содержат нечетное число единичных разрядов.

   В случае контроля на нечетность, контрольный бит содержит "1", если данные содержат нечетное число единичных разрядов, и "0", если данные содержат четное число единичных разрядов.

   Если контроль четности отсутствует, то контрольный бит в поток данных не включается.
4. Объем данных может составлять восемь или девять бит, однако применение девятиразрядных данных требует дополнительного программного обеспечения для их формирования и обработки.
5. Длительность стопового периода может составлять один, полтора или два стандартных битовых периода. Хотя при низкоскоростной передаче допускается несоответствие в числе битов данных, на которые сконфигурированы передатчик и приемник, при более высоких скоростях такое несоответствие вызовет ошибки передачи.

Рис. 4.6. Пример передачи данных через UART

Существует три варианта конфигурации контроля четности:

- отсутствие контроля — бит контроля четности передатчиком не формируется и в поток данных не включается;

- контроль на четность — контрольный бит содержит "1", если "чистые" данные содержат четное число единичных разрядов, и "0", если данные содержат нечетное число единичных разрядов;
- контроль на нечетность — контрольный бит содержит "1", если "чистые" данные содержат нечетное число единичных разрядов, и "0", если данные содержат четное число единичных разрядов (по сути, — проверка, противоположная проверке на четность).

Поскольку контроль четности связан с дополнительным программным кодом, его следует использовать только в том случае, когда он реализован на принимающей стороне. Контроль четности особенно полезен в условиях сильно зашумленной среды, когда существует вероятность того, что данные могут быть искажены.

В заключение отметим, что стоп-биты, передаваемые за байтом данных, важны тем, что гарантируют переход линии в неактивное состояние на период времени, достаточный для перевода в исходное состояние аппаратных средств синхронизации приемника. Количество стоп-битов обычно стараются свести к минимуму, чтобы увеличить пропускную способность канала, однако в зашумленной среде или в узлах, требующих немного дополнительного времени для обработки полученных данных, без использования дополнительных стоп-битов не обойтись.

Приемопередатчик UART хорошо работает не только при соедини двух печатных плат, но и с гораздо более длинными линиями связи. Типичная максимальная длина для каналов связи RS-232 составляет около 15 метров, однако для дифференциальных каналов RS-422 и RS-485 это значение возрастает до 4 000 метров. Скорость передачи данных по интерфейсу UART обычно находится в диапазоне 300 бит/с..115 Кбит/с.

Базовая структура работы интерфейса UART



Программный код для интерфейса UART можно найти на прилагаемом к книге компакт-диске в папке `Коды` в файле `CommIF.c`. При этом прототипы и соответствующие определения находятся в заголовочном файле `CommIFDef.h`.

Реализованный как буферизуемая, управляемая по прерываниям структура, этот интерфейс поддерживает передачу и прием данных из кольцевых буферов приема/передачи, поддерживаемых приложением. Благодаря буферизованному интерфейсу, приложение освобождено от необходимости блокировать передачу или прием сообщений, длина которых превосходит размер четырехбайтовых внутренних буферов приема и передачи UART — характеристика, обязательная для критичных ко

времени систем. Кроме того, данный интерфейс позволяет проектировщику при работе с ЦКС dsPIC, поддерживающим два порта UART, использовать как UART 1, так и UART 2.

Для использования предлагаемого интерфейса приложение должно вначале инициализировать требуемый коммуникационный порт, вызвав функцию `CommInit()` с параметрами, задающими порт и его рабочие свойства (скорость, тип контроля четности и число стоп-битов). Поскольку эта функция инициализирует глобальные переменные состояния, используемые для управления интерфейсом и конфигурирования аппаратных средств UART, то отсутствие вызова функции `CommInit()` перед вызовом других функций интерфейса UART приведет к сбою системы. Программный код функции `CommInit()` представлен в листинге 4.1.

Листинг 4.1. Функция `CommInit()`

```
*****************************************************************
* ФУНКЦИЯ:   CommInit(UintS ui8Port, Uint16 ui16BaudRate,
*                     Uint16 ui16Parity, Uint16 ui16StopBits)
*
* ОПИСАНИЕ:  Эта функция конфигурирует коммуникационный(е)
*            канал(ы) системы. Она должна быть настроена на
*            параметры конкретных модулей и каналов связи,
*            используемые в конкретном приложении.
*
*            Данная функция использует глобальные системные
*            параметры коммуникационной конфигурации. Если
*            какой-либо из этих параметров будет иметь
*            недопустимое значение, то будут применены
*            значения коммуникационных параметров,
*            используемые по умолчанию, чтобы гарантировать
*            минимальную работоспособность канала связи.
*
*            ПРИМЕЧАНИЕ: Хотя аппаратные средства dsPIC
*                        допускают использование 9-разрядных
*                        данных, эта функция поддерживает
*                        только 8-разрядные данные, поскольку
*                        такой формат более распространен
*                        и с ним легче работать.
*
* ПАРАМЕТРЫ: uiSPort - номер конфигурируемого порта UART
*                      (UART_1 или UART_2)
*            ui16BaudRate - скорость обмена данными в бит/с
*            ui16Parity - тип контроля четности
*                         (ДОЛЖЕН быть в виде PARITY_*)
```

Листинг 4.1. Продолжение

```
*            ui16StopBits - количество стоп-битов
*                           (STOP_BITS_1 либо STOP_BITS_2)
*
* ВОЗВРАЩАЕТ:       Одно из следующих значений кода состояния:
*                   ST_OK - операция выполнена успешно
*                   ST_INV_PARM - обнаружен неверный
*                         коммуникационный параметр, были
*                         использованы значения параметров,
*                         устанавливаемые по умолчанию
*                   ST_COMM_INIT - не удалось инициализировать
*                         указанный(е) канал(ы) связи
*
* РЕДАКЦИЯ: 0, версия 1.0        ДАТА: 18 мая 2006 года
*
* Оригинальный вариант.
*****************************************************************

Uint16 CommInit(Uint8 ui8Port, Uint16 ui16BaudRate,
                Uint16 ui16Parity, Uint16 ui16StopBits)
{
// Локальные переменные
Uint16
   ui16BRGValue,    // Значение для генератора скорости
                    // обмена данными
   ui16Mode,        // Конфигурационные данные для
                    // регистра UxMODE
   ui16Status;      // Конфигурационные данные для
                    // регистра UxSTA

// Регистрируем UART, который используем
// для коммуникационного порта

g_ui8CommPort = ui8Port;

// Отключаем заданные модуль UART и запрещаем связанное с ним
// прерывание, чтобы завершить инициализацию без прерываний

if (ui8Port == UART_2)
   CloseUART2();    // Закрываем UART 2
else
   CloseUART1();    // Закрываем UART 1

// Вычисляем значение для генератора скорости обмена данными
// (BRG) на основании частоты системной синхронизации
// и желаемой скорости обмена данными.
```

Листинг 4.1. Продолжение

```
// По спецификации 30F6014A значение BRG вычисляется как
// BRG = ((FCY / скорость обмена) / 16) - 1
// где: FCY - частоте системной синхронизации в Гц,
//            скорость обмена - желаемая скорость в бит/с

ui16BRGValue = ((FCY / ui16BaudRate) / 16) - 1;

// Устанавливаем значение регистра UxMODE, чтобы задать,
// активен ли UART, как он будет обрабатывать состояние
// ожидания, а также других конфигурационные данные

ui16Mode = UART_EN              &    // Активизируем UART
           UART_IDLE_CON        &    // Останов UART, когда
                                     // линия - в состоянии
                                     // ожидания
           UART_DIS_WAKE        &    // Запрет пробуждения UART
                                     // по стартовому биту
           UART_DIS_LOOPBACK    &    // Отменить режим цикла
           UART_DIS_ABAUD;           // Не использовать режим
                                     // автоустановки скорости

// Добавляем необходимую информацию по конфигурированию,
// задав количество битов данных и требуемый режим
// контроля четности

switch (ui16Parity)
   {
   case PARITY_EVEN:
      ui16Mode &= UART_EVEN_PAR_8BIT; // Контроль на
      break;                    // четность, 8 разрядов данных

   case PARITY_ODD:
      ui16Mode &= UART_ODD_PAR_8BIT;  // Контроль на
      break;                  // нечетность, 8 разрядов данных

   default:
      ui16Mode &= UART_NO_PAR_8BIT; // Нет контроля
      break;                    // четности, 8 разрядов данных
   }

// Добавляем необходимую конфигурационную информацию
// о количестве используемых стоп-битов
```

Листинг 4.1. Продолжение

```
if (ui16StopBits == STOP_BITS_2)
   ui16Mode &= UART_2STOPBITS;  // Используем 2 стоп-бита
else
   ui16Mode &= UART_1STOPBIT;   // Используем 1 стоп-бит

// Устанавливаем значение регистра UxSTA, чтобы задать, как
// будут генерироваться прерывания при передаче и приеме
// данных, а также указываем другие конфигурационные
// управляющие данные

ui16Status = UART_INT_TX          &  // Прерывание от
                                     // регистра Tx
             UART_TX_PIN_NORMAL   &  //Использовать нормальное
                                     // состояние выводов Tx
                                     // (не передавать разрыв)
             UART_TX_DISABLE      &  // Отключить передатчик
                                     // с текущего момента
             UART_INT_RX_CHAR     &  // Прерывание по
                                     // приему символа
             UART_ADR_DETECT_DIS  &  // Не использовать режим
                                     // распознавания адреса
             UART_RX_OVERRUN_CLEAR;  // Очистить флаг
                                     // переполнения UART Rx

// Инициализируем глобальные переменные состояния,
// имеющие отношение к каналу связи

g_ui16CommRxAppIndex = 0;          // Инициализируем номер
                                   // буфера приема данных со
                                   // стороны приложения
g_ui16CommRxISRIndex = 0;          // Инициализируем номер
                                   // буфера приема данных
                                   // со стороны подпрограммы
                                   // обслуживания прерывания
//Инициализируем буфер приема данных
memset(g_ui8CommRxData, 0, COMM_RX_BUFF_SZ);

g_ui16CommTxAppIndex = 0;          // Инициализируем номер
                                   // буфера передачи данных
                                   // со стороны приложения
g_ui16CommTxISRIndex = 0;          // Инициализируем номер
                                   // буфера передачи данных
                                   // со стороны подпрограммы
                                   // обслуживания прерывания
//Инициализируем буфер передачи данных
```

Листинг 4.1. Окончание

```
memset(g_ui8CommTxData, 0, COMM_TX_BUFF_SZ);

// Фактическая инициализация соответствующего UART
if (ui8Port == UART_2)
   {
   // Разрешаем прерывание при приеме данных
   // и запрещаем прерывание при передаче
   ConfigIntUART2(UART_RX_INT_EN & UART_RX_INT_PR6 &
                  UART_TX_INT_DIS & UART_TX_INT_PR3);

   // Открываем UART 2
   OpenUART2(ui16Mode, ui16Status, ui16BRGValue);
   }
else
   {
   // Разрешаем прерывание при приеме данных
   // и запрещаем прерывание при передаче
   ConfigIntUART1(UART_RX_INT_EN & UART_RX_INT_PR6 &
                  UART_TX_INT_DIS & UART_TX_INT_PR3);

   // Открываем UART 1
   OpenUART1(ui16Mode, ui16Status, ui16BRGValue);
   }

return ui16Status;
}
```

Как только коммуникационный интерфейс инициализирован функцией `CommInit()`, и установлен достаточно низкий уровень прерываний ЦКС dsPIC для разрешения прерываний от последовательного порта (уровень 6), приложение может обмениваться данными с выбранным UART. Для того чтобы прочитать данные из инициализированного последовательного порта, приложение вначале проверяет наличие данных, ожидающих в глобальной очереди приема. Для этого вызывается функция `CommGetRxPendingCount()`, которая возвращает число непрочитанных байт данных в очереди приема. Если это значение отлично от нуля, то в очереди присутствуют доступные для чтения данных. В таком случае приложение считывает следующий символ из очереди, вызвав функцию `CommGetRxChar()`. Блок-схема алгоритма, иллюстрирующая эту последовательность действий, показана на рис. 4.7.

Если функция `CommGetRxChar()` возвращает значение кода состояния `ST_OK`, то это означает, что приложение может безопасно выполнять чтение данных из UART. Если же значение кода состояния не

будет равно ST_OK, то приложение не должно использовать данные, находящиеся в ui8NewData, поскольку они не будут корректными.

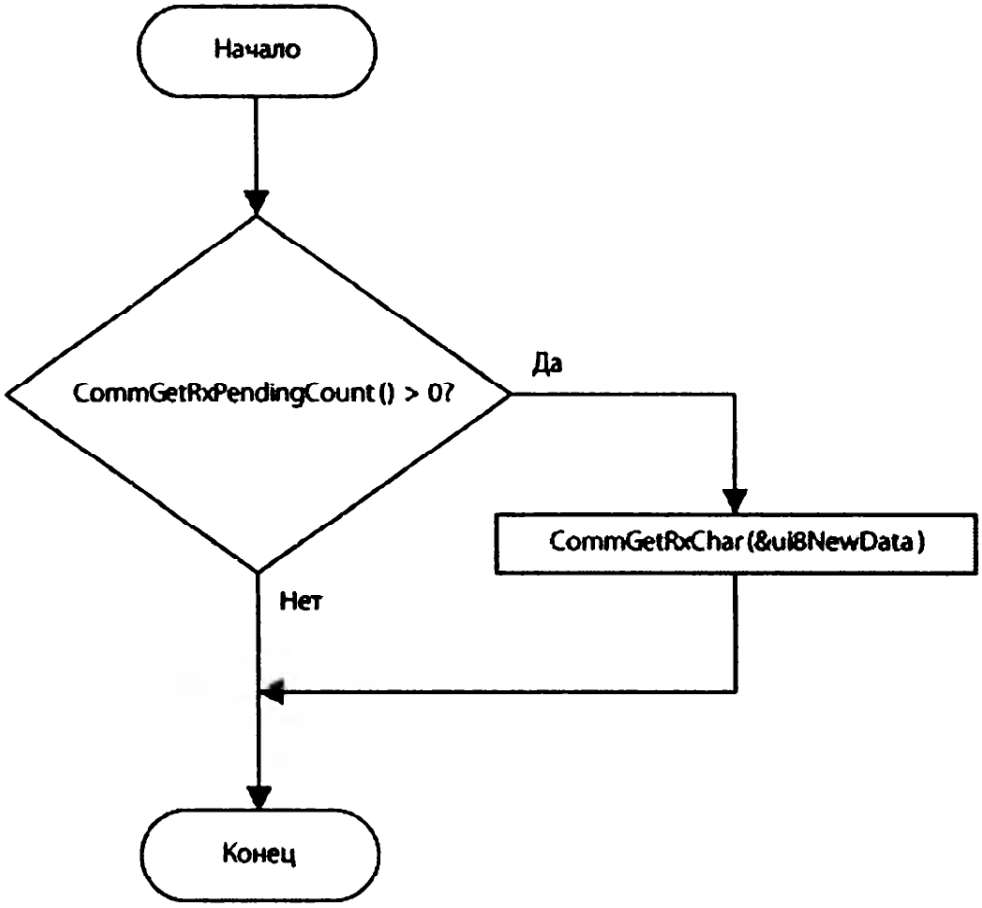


Рис. 4.7. Блок-схема алгоритма чтения данных из UART

Алгоритм передачи данных так же прост. При условии, что коммуникационный интерфейс уже был инициализирован с помощью функции CommInit(), приложение использует функции CommPutChar() и CommPutBuff() для передачи одного или нескольких байт данных соответственно. Функция CommPutBuff() просто вызывает в цикле функцию CommPutChar() для загрузки в кольцевой буфер передачи данных полного содержимого сообщения. Программный код функции CommPutChar() (листинг 4.2) полезно рассмотреть подробнее.

Листинг 4.2. Функция CommPutChar()

```
*****************************************************************
* ФУНКЦИЯ:   CommPutChar(Uint8 ui8Data)
*
* ОПИСАНИЕ:  Эта функция добавляет символ в глобальный буфер
*            данных передачи для последующей передачи
*            подпрограммой обслуживания прерывания от UART.
```

Листинг 4.2. Продолжение

```
*           Если буфер полон, то символ в него не заносится,
*           а функция возвращает состояние ошибки.
*
* ПАРАМЕТРЫ: ui8Data - данные, которые должны быть добавлены
*           к буферу с целью их последующей передачи UART
*
* ВОЗВРАЩАЕТ: Одно из следующих значений кода состояния:
*           ST_OK - операция выполнена успешно;
*           ST_BUFFER_FULL - буфер передачи заполнен,
*                  данные для передачи в буфер не занесены
*
* РЕДАКЦИЯ: 0, версия 1.0        ДАТА: 18 мая 2006 года
*
* Оригинальный вариант.
*****************************************************************

Uint16 CommPutChar(Uint8 ui8Data)
{
// Локальные переменные
Uint16
   ui16AppIndex,    // Временный буфер для обработки номера
                    // для буфера передачи данных
   ui16DataCount,   // Количество символов, находящихся
                    // в буфере передачи данных
   ui16Status;      // Состояние выполнения функции

// Проверяем, есть ли в буфере передачи данных
// место для новых данных

ui16Status = ST_BUFFER_FULL;     // Предполагаем, что буфер
                                 // передачи заполнен

ui16DataCount = CommGetTxBuffCount();  // Получаем количество
                                       // символов, уже
                                       // размещенных
                                       // в буфере передачи

if (ui16DataCount < (COMM_TX_BUFF_SZ - 1))
   {
   // В буфере есть место, поэтому заносим в него символ
   g_ui8CommTxData[g_ui16CommTxAppIndex]= ui8Data;

   // Модифицируем указатель приложения так, чтобы он
   // указывал на следующий символ в буфере передачи.
   // Используем локальную переменную для манипуляций
   // и последующей передачи значения в глобальную переменную
```

Листинг 4.2. Окончание

```
    // состояния для предотвращения проблем с неоднородностью
    // данных при вызове подпрограммы обслуживания прерывания.

    ui16AppIndex = g_ui16CommTxAppIndex; // Получаем текущий
                                         // индекс приложения
    ui16AppIndex++;                      // Прибавляем единицу
    // Выполняем быстрый подсчет модуля COMM_TX_BUFF_SZ
    // (предполагаем, что COMM_TX_ BUFF_SZ - это степень 2)
    ui16AppIndex &= (COMM_TX_BUFF_SZ - 1);

    // Обновляем переменную состояния, которая контролирует
    // доступ к глобальной очереди передачи данных

    g_ui16CommTxAppIndex = ui16AppIndex;

    // Убеждаемся в том, что аппаратура может и будет
    // передавать данные. Поскольку они фактически загружены
    // в буфер передачи UART подпрограммой обслуживания
    // прерывания, необходимо убедиться в том, что эта
    // подпрограмма вызывается, если в буфере передачи есть
    // свободное место. Если свободного места нет, то в этот
    // раз генерировать прерывание нет необходимости (новые
    // данные уже были добавлены в глобальную очередь
    // передачи данных и будут загружены в буфер передачи
    // UART при следующем прерывании).

    CommEnableTransmitter(); // ПРЕЖДЕ ВСЕГО убеждаемся в том,
                             // что передатчик активирован

    // В буфере передачи UART есть свободное место, поэтому
    // выполняем прерывание передачи. Это должно быть сделано
    // ПЕРЕД тем, как убедиться, что прерывание от UART при
    // передаче разрешено, во избежание конфликта с
    // подпрограммой обслуживания прерывания при передаче

    if (CommTxBuffAvail()) CommForceTxInt();

    CommEnableTxInt();      // И, наконец, гарантируем, что
                            // прерывание от UART разрешено

    // Устанавливаем флаг успешного выполнения операции
    ui16Status = ST_OK;
    }
return ui16Status;
}
```

Интерфейс CAN

Интерфейс CAN — пожалуй, самый сложный из всех последовательных интерфейсов в ЦКС dsPIC. В его состав входит передовой аппаратный контроллер, поддерживающий скорость передачи данных до 1 Мбит/с со встроенным аппаратным обнаружением ошибок, сложная схема присвоения приоритета сообщениям и средства фильтрации, позволяющие получать только представляющие интерес сообщения. И все это — при совсем незначительном потреблении процессорного времени. Широко используемая в автомобилестроении и производстве, архитектура CAN обеспечивает надежный способ связывания множества узлов в рамках одной сети.

Но неужели существуют какие-либо причины, *не позволяющие* использовать интерфейс CAN? Да, и их — целых две: сложность и стоимость. Одно из больших преимуществ CAN — развитые возможности конфигурирования, однако они настолько развиты, что это же становится и недостатком. Благодаря своей гибкости, топология CAN может использоваться для решения самых разнообразных задач с применением, по сути, неизменного набора аппаратных средств. Но, к сожалению, эта гибкость требует очень точного конфигурирования, иначе канал будет или ненадежен, или полностью неработоспособен, а на отладку канала порой уходит немало времени и сил.

Введение в архитектуру CAN

Разработанная компанией Bosch в начале 1980-х годов, архитектура CAN довольно проста. Хотя сам по себе стандарт CAN намеренно нейтрален по отношению к среде передачи информации[20], одна из наиболее распространенных его реализаций использует одну дифференциальную последовательную шину, работающую на скорости передачи данных 1 Мбит/с (или ниже) и соединяет несколько узлов. С учетом соответствующего сигнала "земли", надежный интерфейс может состоять всего лишь из трех проводов!

Коммуникационный протокол CAN — представитель семейства протоколов CSMA/CD. Эта аббревиатура расшифровывается как "Carrier Sense Multiple Access/ Collision Detection" ("Множественный доступ

---

[20] *Нейтральность по отношению к среде* означает, что протокол не определяет физическую среду, требуемую для его реализации. Этот аспект был намеренно вынесен за рамки спецификации CAN, чтобы протокол мог работать с различными физическими средами (лишь бы они поддерживали доминантное и рецессивное битовые состояния).

с контролем несущей/обнаружением коллизий". Хотя это название выглядит громоздким, стоящие за ним концепции довольно просты для понимания. В системе *с контролем несущей* все узлы, прежде чем попытаться передать сообщение, должны проверить сеть на предмет отсутствия активности в течение определенного периода. Как только этот период истек, любой из узлов в сети может передавать данные, — отсюда и термин "множественный доступ" в названии семейства протоколов. Само собой разумеется, будут возникать ситуации, когда несколько попытаются отправить данные одновременно, — состояние известное как *конфликт* — поэтому система должна реализовать какой-нибудь метод *обнаружения конфликтов*. Разные представители семейства CSMA/CD решают эти задачи по-разному, однако все протоколы определенного типа (например, CAN) делают это одинаково.

Из названных задач (контроль несущей и обнаружение коллизий) гораздо более сложной является задача обнаружения коллизий. Разработчики CAN нашли оригинальное решение этой проблемы, позволяющее разработчику назначать приоритеты сообщениям в системе. Более важные сообщения всегда получают доступ к шине перед менее важными (аналогично тому, как реализована приоритетность прерываний в ЦКС dsPIC). Интерфейс CAN не только допускает присвоение приоритетов сообщениям — его схема сетевого *арбитража*[21] *недеструктивна*[22] по отношению к сообщениям с более высоким приоритетом и гарантирует, что сообщения с более высоким приоритетом не будут испытывать никаких задержек с передачей. Поскольку арбитраж сообщений весьма важен, мы немного позже рассмотрим его подробнее.

Другой ключевой особенностью CAN является наличие встроенных схем обнаружения ошибок, которые сигнализируют о проблемах с шиной и постепенно отключают отдельные сетевые узлы от шины, если эти узлы создают слишком много ошибок. Хотя протокол CAN и не поддерживает коррекцию ошибок, встроенная возможность обнаруже-

---

[21] В данном случае *арбитраж* — это процесс, в ходе которого нескольким узлам, конкурирующим за доступ к сети, разрешается передавать данные. *Арбитраж прерываний* — это процесс, в ходе которого контроллер прерываний ЦКС dsPIC определяет, какой обслужить запрос на прерывание.

[22] *Недеструктивный арбитраж* подразумевает, что сообщение, которое будет передано по шине, остается неповрежденным. Деструктивный арбитраж определил бы, какое из сообщений следует передать по шине, но при этом само сообщение было бы разрушено, а значит узлу, которому разрешена передача, пришлось бы опять выполнить передачу с самого начала. Это увеличивает общее время передачи и снижает результирующую пропускную способность.

ния ошибок помогает избежать серьезных проблем, связанных с блокировкой сбойным узлом всей сети.

Если вся эта информация о функциональных возможностях CAN наводит на мысль о серьезной нагрузке на процессор, то можете вздохнуть с облегчением — основная часть интерфейса CAN реализована в двух аппаратных компонентах:

- конечный автомат контроллера CAN, выполняющий весь арбитраж и обнаружение ошибок;
- драйвер шины CAN, управляющий физической средой шины CAN и контролирующий ее состояние.

В большинстве систем эти два компонента реализованы в отдельных интегральных схемах[23]. Это относится и к ЦКС dsPIC, в который встроен один или два модуля контроллера CAN (в зависимости от конкретной модели dsPIC), и который требует подключения к шине внешней микросхемы драйвера CAN. Как только схема интерфейса CAN сконфигурирована, она просто выдает полностью сформированные информационные сообщения и биты состояния приемнику, а также передает сообщения другим узлам. Поскольку все обнаружение ошибок и управление реализовано аппаратно, нагрузка на процессор, связанная с интерфейсом CAN, — минимальна.

И напоследок упомянем о допустимой длине шины CAN, которая зависит от скорости передачи данных. Рекомендуемую максимальную длину шины можно определить по табл. 4.1[24].

Таблица 4.1. Рекомендуемые максимальные значения длины шины CAN

Скорость передачи данных, Кбит/с	Длина шины, м
1 000	30
500	100
250	250
125	500
62,5	1 000

Как ясно видно из табл. 4.1, максимальная длина шины быстро уменьшается с увеличением скорости передачи данных, однако даже при скорости 1 Мбит/с максимальная длина шины довольно велика.

---

[23] Интегральные схемы — это кремниевые кристаллы, содержащие большую часть электроники системы.

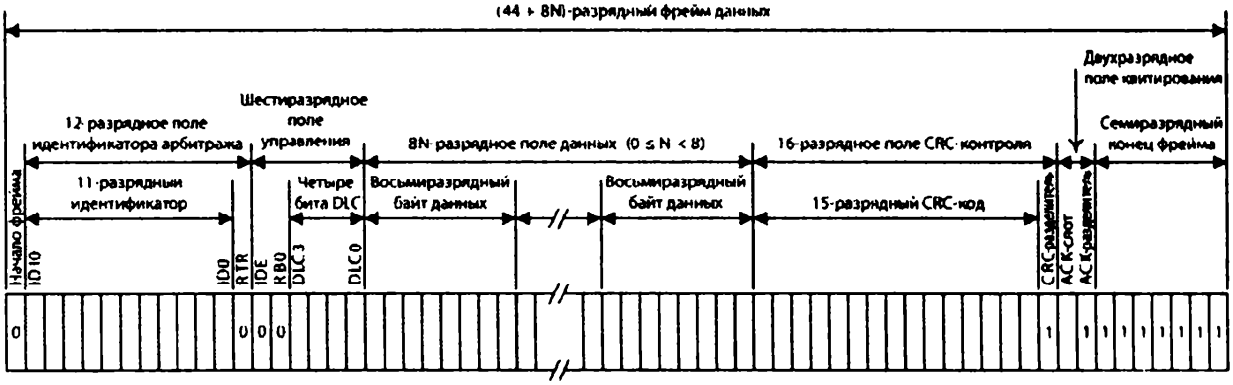
[24] Таблица взята из документа DS00713A, "Controller Area Network (CAN) Basics", опубликованного на Web-сайте компании Microchip.

Форматы данных CAN

Согласно спецификации CAN 2.0[25], данные, передаваемые по шине CAN, могут иметь один из следующих базовых форматов (*фреймов*):

- фрейм данных — передает данные от одного узла всем остальным узлам, подключенным к шине;
- фрейм удаленной передачи — запрашивает данные у другого узла на шине;
- фрейм ошибки — сообщает об обнаружении ошибки передачи;
- фрейм перегрузки — сообщает о том, что передающий узел занят обработкой предыдущего сообщения и в текущий момент времени не может принять данные.

В нашем примере основное внимание будет уделено самому распространенному формату: фрейму данных. Он имеет две разновидности: стандартный фрейм (рис. 4.8) и расширенный фрейм (рис. 4.9).



Ключевые моменты

1. Каждое сообщение состоит из четырех основных полей и нескольких обрамляющих разрядов:
   А. 12-разрядное поле идентификатора арбитража.
   Б. Шестиразрядное поле управления.
   В. 8N-разрядное поле данных (из N байтов).
   Г. 16-разрядное поле CRC-контроля.

   Пользователь может управлять только первыми тремя полями, а значения поля CRC и обрамляющих разрядов устанавливает аппаратура контроллера CAN.

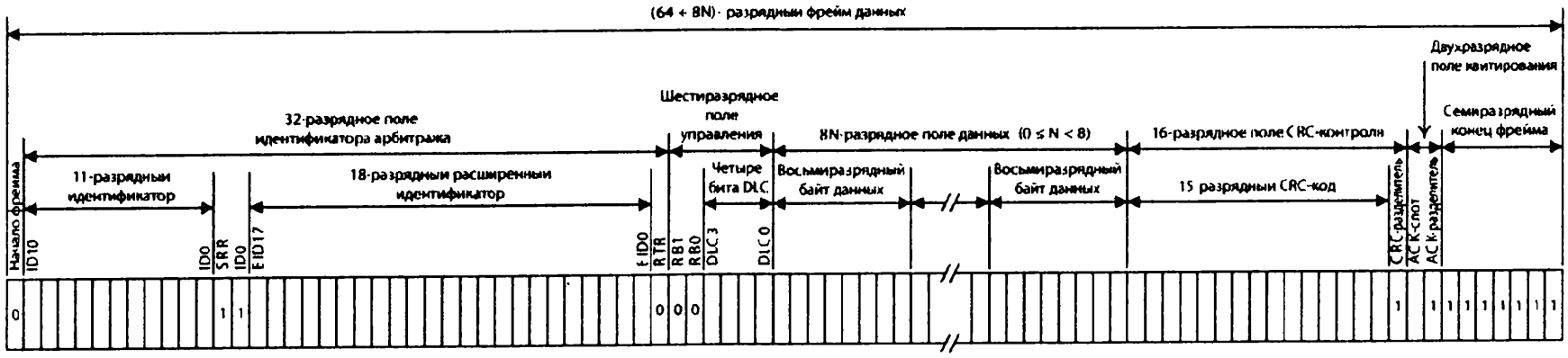
2. Битовое значение "1" рассматривается как состояние низкого уровня, а битовое значение "0" — высокого уровня сигнала.

3. Арбитраж шины, определяющий, какой узел может передавать сообщение, основан на значении поля идентификатора арбитража. При этом сообщение с первым доминантным разрядом в этом поле будет иметь приоритет, и ему будет разрешена передача по шине. На практике это означает, что чем меньше значение поля, тем выше приоритет сообщения.

4. Фреймы данных стандартного формата имеют приоритет над фреймами данных расширенного формата.

***Рис. 4.8.** Стандартный формат фрейма данных CAN*

[25] CAN Specification 2.0, Robert Bosch GmbH, 1991.



Ключевые моменты

1. Каждое сообщение состоит из четырех основных полей и нескольких обрамляющих разрядов:
   А. 32-разрядное поле идентификатора арбитража.
   Б. Шестиразрядное поле управления.
   В. 8N-разрядное поле данных (из N байтов).
   Г. 16-разрядное поле CRC-контроля.

   Пользователь может управлять только первыми тремя полями, а значения поля CRC и обрамляющих разрядов устанавливает аппаратура контроллера CAN.

2. Битовое значение "1" рассматривается как состояние низкого уровня, а битовое значение "0" — высокого уровня сигнала.

3. Арбитраж шины, определяющий, какой узел может передавать сообщение, основан на значении поля идентификатора арбитража. При этом сообщение с первым доминантным разрядом в этом поле будет иметь приоритет, и ему будет разрешена передача по шине. На практике это означает, что чем меньше значение поля, тем выше приоритет сообщения.

4. Поскольку используется схема арбитража шины, а нулевой (доминантный) разряд RTR стандартного фрейма данных выравнивается по единичному (рецессивному) разряду SRR расширенного фрейма данных, стандартные фреймы данных всегда имеют приоритет над расширенными.

Рис. 4.9. *Расширенный формат фрейма данных CAN*

Эти два формата фреймов данных, по сути, идентичны. Единственное реальное различие между ними заключается в более коротком идентификаторе арбитража у стандартного фрейма. Все форматы фреймов данных содержат следующие базовые элементы:

- поле идентификатора арбитража, размер которого изменяется в зависимости от типа фрейма;
- шестиразрядное поле управления;
- поле данных длиной от нуля до восьми байт;
- двухбайтовое поле CRC-контроля (контроль циклическим избыточным кодом);
- двухразрядное поле квитирования;
- семиразрядный маркер конца фрейма.

Среди перечисленных полей пользователь может управлять полями арбитража, управления и данных, в то время как содержимое полей CRC-контроля, квитирования и маркера конца фрейма определяется и контролируется автоматически аппаратными средствами контроллера CAN. Рассмотрим эти поля немного подробнее, прежде чем перейти к изучению методики арбитража CAN.

В стандартном фрейме данных поле идентификатора арбитража состоит из 11-разрядного идентификатора и одноразрядного флага запроса удаленной передачи RTR (Remote Transmission Request). Расширенный формат фрейма данных немного отличается, однако протокол CAN предусматривает более высокую приоритетность стандартного фрейма, если у него возникает конфликт с расширенным фреймом. В случае расширенного фрейма идентификатор арбитража состоит из 29 разрядов. При этом старшие 11 разрядов передаются после признака начала фрейма, за которым следует флаг замещения удаленного запроса SRR (Substitute Remote Request), одноразрядный флаг расширения идентификатора IDE (Identifier Extension), а затем — остальные 18 разрядов идентификатора с одноразрядным флагом RTR, завершающим поле идентификатора арбитража.

Присутствуют также небольшие различия в структуре поля управления для двух названных форматов фреймов данных, хотя оно в обоих случаях имеет длину, равную шести разрядам. В стандартном фрейме, первый передаваемый бит поля управления — это флаг IDE, за которым следует один резервный разряд, обозначаемый в спецификации CAN как "r0". Заключительные четыре разряда поля управления включают код длины данных DLC (Data Length Code), который задает число байтов данных, которые последуют в сообщении. Хотя длина поля DLC составляет четырем разряда, оно может принимать значения только от 0

до 8, поскольку протокол CAN поддерживает максимум восемь байт данных на сообщение.

Поскольку расширенный фрейм включает в себя флаг IDE, как часть поля идентификатора арбитража, его поле управления содержит два зарезервированных разряда: "rl" и "r0". Поле DLC в нем такое же, как и в стандартном фрейме данных, и для него справедливы те же ограничения.

Поле циклического избыточного кода (CRC) не представляет для разработчиков особого интереса, поскольку оно обрабатывается исключительно аппаратными средствами и, следовательно, "прозрачно" для программиста. Отметим только, что сам CRC-код — это 15-разрядное значение, кроме которого в поле CRC-контроля присутствует еще одноразрядный CRC-разделитель.

Последнее поле в сообщении CAN — это двухразрядное поле квитирования (ACK), которое состоит из ведущего разряда подтверждения, который устанавливается передающим узлом в состояние, соответствующее низкому уровню сигнала (более подробно рассматривается чуть позже), а затем устанавливается в состояние, соответствующее высокому уровню сигнала, всеми узлами, которые успешно получили данное сообщение вне зависимости, используют они его фактически или нет. Последний разряд поля ACK (и сообщения) — это ACK- разделитель, который просто переводит шину в состояние, соответствующее низкому уровню сигнала, чтобы сигнализировать о завершении передачи.

Хотя в нашем примере мы не будем использовать фреймы удаленной передачи, ошибки и перегрузки, следует отметить, что интерфейс CAN ЦКС dsPIC полностью их поддерживает. Фреймы удаленной передачи используются для выдачи запросов на автоматическую передачу данных из узла (когда данные уже загружены в модуль CAN в ожидании запроса), а фрейм ошибки формируется узлом при обнаружении им состояния сбоя в шине. Поскольку фреймы ошибки намеренно нарушают параметры синхронизации шины CAN, они заставляют все узлы, передающие данные, прекратить передачу, вернуться в исходное состояние и повторить передачу сначала.

Арбитраж шины

Как уже было отмечено выше, из-за асинхронности передачи данных необходимо реализовать арбитраж, определяющий, какой узел может передавать данные, если два узла попытались сделать это одновременно. Разработчики CAN придумали оригинальное решение этой проблемы, создав недеструктивную схему арбитража. Данная схема для оп-

ределения приоритетности узлов использует значения идентификаторов арбитража конфликтующих сообщений.

Прежде, чем рассмотреть принцип действия схемы арбитража, имеет смысл упомянуть о двух терминах, применимых к системам на базе CAN. Принято говорить, что данные в шине CAN могут находиться в *доминантном* (логический 0) или *рецессивном* состоянии (логическая 1). Когда одновременно передаются два бита, находящихся в разных состояниях, то доминантное состояние "побеждает", т.е. именно оно определяет состояние шины.

Интерфейс CAN использует это правило для арбитража доступа. Всякий раз, когда несколько узлов пытаются одновременно передать сообщения, доминантным состоянием считается то, которое уже присутствует в шине. Поскольку каждый узел передает данные в шину побитно, он проверяет, отражают ли данные в шине состояние бита, переданного последним.

Если передающий узел посылает рецессивный бит, но обнаруживает, что шина находится в доминантном состоянии, то это означает, что в этот момент свое сообщение передает какой-то другой узел. В результате узел, данные которого находятся в рецессивном состоянии, сразу же отключает свой передатчик и возобновляет попытку передать данные только, дождавшись завершения текущей передачи.

Выполняя подобный арбитраж, интерфейс CAN обеспечивает структурированный подход к доступу на передачу, а также гарантирует, что конфликты не приведут к потере данных и, как следствие, — принудительному повторению передачи сообщений на всех узлах. Поскольку доминантное состояние соответствует логическому нулю, разработчики систем на базе CAN выбирают идентификаторы арбитража таким образом, чтобы наиболее важные сообщения имели низкие значения, т.е. обладали более высоким приоритетом. Например, выбирая значения идентификатора арбитража 000H–01FH для аварийных сообщений и значения 020H–7FFH для нормальных рабочих сообщений, разработчик гарантирует, что приоритет аварийных сообщений будут всегда выше приоритета нормальных рабочих сообщений.

На рис. 4.10 показан пример, в котором аварийное сообщение с идентификатором арбитража 010H передается одновременно с обычным рабочим сообщением с идентификатором арбитража 040H.

Данная схема позволяет находиться в шине как стандартным, так и расширенным фреймам данных. При этом стандартные фреймы будут иметь приоритет над расширенными.

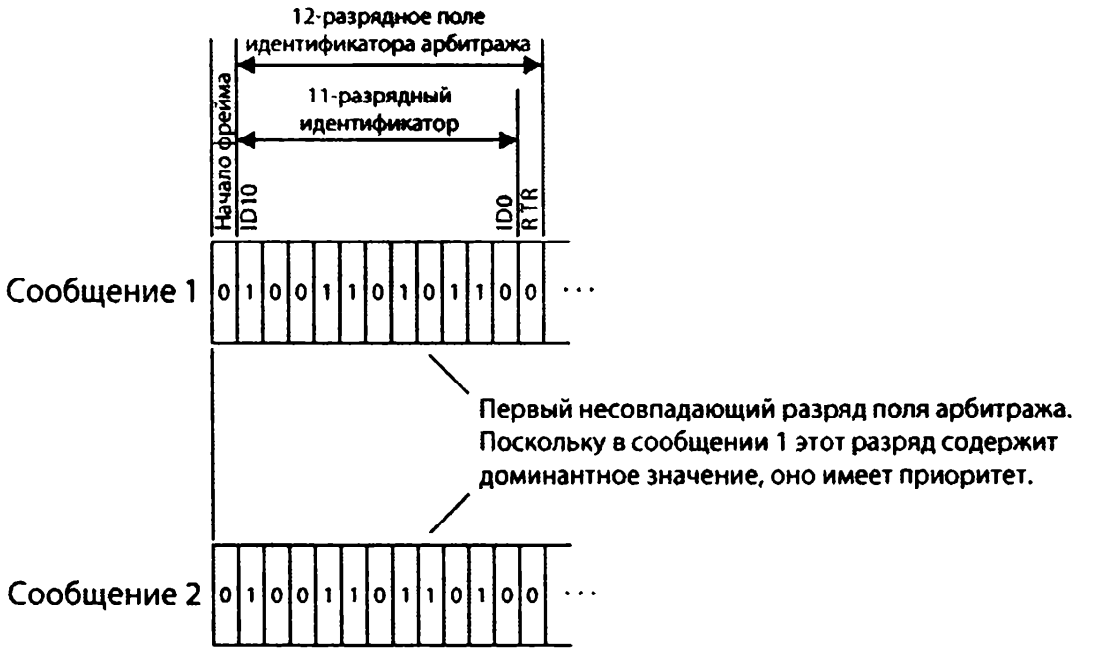


Рис. 4.10. Пример арбитража для двух сообщений, посылаемых одновременно

Фильтры принятия сообщений

Один из дополнительных аспектов протокола CAN, который реализован во всех контроллерах CAN, — это *фильтрация сообщений*, позволяющая контроллеру принимать только те сообщения, поле идентификатора арбитража которых соответствует запрограммированному значению фильтра. В данном случае, говоря о фильтре, мы имеем в виду не цифровой фильтр, обрабатывающий цифровой сигнал, а процесс, в результате которого для обработки контроллером CAN выбирается только ограниченная группа сообщений, соответствующих некоторым критериям. Обратите внимание на то, что даже в случае игнорирования сообщения контроллером, он все равно надлежащим образом отвечает на него, записывая соответствующее значение в разряд квитирования.

Фильтрация — это методика среднего уровня, с помощью которой можно уменьшить нагрузку на процессор, ограничив типы сообщений, подлежащих обработке. В то же время, процесс квитирования сообщений — выполнение требований низкого аппаратного уровня, направленных на гарантирование безошибочной доставки сетевого потока ко всем узлам.

Фильтрация сообщений CAN разбита на два этапа, каждый из которых можно сконфигурировать программно, хотя они и реализуются аппаратными средствами контроллера CAN. Вначале необходимо установить значения фильтра принятия сообщений (ЦКС dsPIC поддерживает до шести различных фильтров). Значение фильтра поразрядно пере-

множается с помощью логической операции "И" с полем идентификатора арбитража каждого полученного сообщения. Результирующее значение затем поразрядно сравнивается с маской принятия сообщений. Если результат применения фильтра соответствует маске принятия, то входящее сообщение добавляется в буфер приема интерфейса CAN (предполагается, что в этом буфере есть свободное место).

Рассмотрим суть фильтрации на примере. Предположим, мы хотим принимать все стандартные фреймы данных CAN, у которых значение поля идентификатора арбитража находится в диапазоне от 300H до 3FFH. В этом случае фильтр принятия сообщений будет равен F00H, а маска — 300H, поскольку логическое перемножение фильтра с 12-разрядным полем идентификатора арбитража любого полученного сообщения будет приводить к игнорированию младшего байта идентификатора арбитража (он логически умножается на нули). Таким образом, маске принятия сообщений будут соответствовать только те поля идентификаторов арбитража, в которых старшие полубайты равны 3.

Базовая структура взаимодействия с интерфейсом CAN

Компания Microchip предоставляя в распоряжение программистов 16-разрядные библиотеки с функциями обслуживания периферийных устройств. В их состав входят и средства, необходимые для организации взаимодействия с контроллерами CAN ЦКС dsPIC. Однако, как и в случае с интерфейсом UART, они не реализуют напрямую буферизированный ввод-вывод сообщений (не символов, поскольку CAN — протокол, ориентированный на сообщения) с управлением по прерываниям. К счастью, мы можем просто подкорректировать программный код, уже разработанный для UART, создав аналогичную структуру для интерфейса CAN. Это превосходный пример ценности обобщенного программного кода, когда одни и те же базовые принципы применимы как к UART, так и к CAN.

4.3. Протоколы высокого уровня

До сих пор мы рассматривали базовые каналы связи, по сути, как механизмы ввода-вывода, передающие данные между ЦКС dsPIC и другими устройствами. При этом мы рассматривали уровень битов/байтов. Даже когда речь шла о протоколе CAN, мы, в действительности, исследовали аппаратный интерфейс низкого уровня и правила его корректного конфигурирования. Все это, конечно, необходимо, однако этого недостаточно. Мы также должны определить фактическое значение передаваемых данных, что имеет отношение к *протоколам высокого уровня.*

В предшествующих главах уже было упомянуто о необходимости преобразования данных в информацию — именно той задаче, за которую отвечают протоколы высокого уровня. Это можно сравнить с пересылкой письма от одного человека другому. Для того чтобы письмо достигло адресата, необходимо задать определенную информацию (адрес получателя). Кроме того, в зависимости от обстоятельств может потребоваться дополнительная необязательная информация (например, имя получателя). Поскольку теоретически может возникнуть проблема с доставкой, то неплохо было бы добавить информацию, предназначенную для оповещения о проблемной ситуации или же для выхода из этой ситуации (например, имя и адрес отправителя). То же самое происходит и с данными, передаваемыми между узлами в системе — будь то отдельными устройствами (например, между ЦКС dsPIC и внешним ЦАП) или же ЦКС dsPIC и удаленными системами. Благодаря применению к передаваемым данным определенной структуры, они будут верно поняты и эффективно обработаны.

Для последовательной передачи данных существует несколько популярных стандартных протоколов. В зависимости от типа оборудования, с которым датчик должен обмениваться данными, разработчику может не оставаться никакого выбора, кроме как реализовать в разрабатываемом приложении конкретный протокол. Если этот протокол — популярен, то вполне вероятно, что для него уже существуют готовые программные реализации, которые можно встроить в приложение и/или средства тестирования конечного программного кода. Однако при таком подходе возникают проблемы, если в приложении необходимо задействовать только небольшое подмножество функций выбранного протокола. В таком случае полная функциональность подразумевает крайне нерациональное использование драгоценных ресурсов процессора. Тем не менее, иногда это — единственный вариант (например, когда необходимо реализовать связь между микросхемами по интерфейсу SPI). Обычно отдельные устройства реализуют специфический протокол, что оставляет разработчику очень немного или вообще никакой гибкости.

Альтернативный подход, использованный, в частности, в нашем примере для организации связи между центральной системой и ЦКС dsPIC, заключается в создании специального частного протокола, максимально приспособленного к конкретному приложению. Если такой протокол правильно реализовать, то он обеспечит требуемую функциональность без лишнего расходования ресурсов на возможности, которые никогда не понадобятся приложению. Однако разработчик всегда должен быть уверен в том, что для приложения действительно требуется

(а команда разработчиков может себе позволить) потратить время и средства на решение задачи с помощью частного протокола[26].

Если после объективного анализа ситуации разработчик решает, что частный протокол необходим, то его необходимо сделать максимально *легким, надежным* и *расширяемым*. Под легкостью протокола подразумевается надежное решение задачи при минимальной нагрузке на процессор. Хотя концепция надежности сама по себе интуитивно понятна, мы также включаем в нее в качестве желаемой цели обнаружение ошибок. И, наконец, под расширяемостью протокола подразумевается его способность реализовать новые типы сообщений при минимальном изменении существующего программного кода.

Иллюстрация одной из таких реализаций — протокол, используемый в наших примерах. В нем создан хороший баланс между потреблением процессорного времени и реализацией обнаружения ошибок и расширяемости. Читатель должен понимать, что это — не единственный (и, возможно, — даже не самый лучший) способ реализации протокола для конкретного приложения; однако он хорошо зарекомендовал себя в самых разных обстоятельствах. Фактически, в наших примерах используются два протокола:

- для работы с сообщениями в общем;
- ориентированный на конкретные типы сообщений.

Для начала рассмотрим общий протокол обработки сообщений.

Общий протокол обработки сообщений

Общий протокол обработки сообщений — это простой протокол типа "команда/ответ", основная задача которого — доставить целевой системе командное сообщение переменной длины и обработать полученное в ответ сообщение переменной длины. И этот простой протокол, описание которого представлено на рис. 4.11 и рис. 4.12, справляется со своей задачей превосходно.

---

[26] Многие инженеры (и компании) страдают от синдрома НИЗ (Не Изобретено Здесь). Это настоящая болезнь, жертвы которой отказываются от проектных решений только потому, что они не созданы самими этими инженерами или их компаниями. Хотя многие склонны относиться к этому с юмором, последствия могут оказаться далеко не смешными: значительные задержки в разработке проекта, потеря рыночной ниши и выпуск второсортных изделий. Все это выливается для компаний в большие затраты и потерю прибыли, поэтому решение о необходимости разработки частного протокола должно базироваться на фактах, а не на эмоциях.

Маркер начала (1 байт)	Идентификатор команды (1 байт)	Длина поля данных (1 байт)	Данные (*N* байт)	Контрольная сумма (1 байт)
0x01 (SOH)	–	*N*	–	Вычисляется

Рис. 4.11. *Формат команды общего протокола обработки сообщений*

Описание полей на рис. 4.11:

- маркер начала — ASCII-символ начала заголовка 0x01;
- идентификатор команды — идентификатор посылаемой команды (старший разряд, установленный в "1", определяет команду);
- длина поля данных — количество байт данных, пересылаемых вместе с командой ($0 \le N \le 255$);
- данные — *N* байт данных, пересылаемых с командой;
- контрольная сумма[27] — вычисляется на основании идентификатора команды, длины поля данных и всех байтов данных.

Маркер начала (1 байт)	Идентификатор ответа (1 байт)	Состояние (1 байт)	Длина поля данных (1 байт)	Данные (*M* байт)	Контрольная сумма (1 байт)
0x01 (SOH)	–	–	*M*	–	Вычисляется

Рис. 4.12. *Формат ответа на сообщение общего протокола*

Описание полей на рис. 4.12:

- маркер начала — ASCII-символ начала заголовка 0x01;
- идентификатор ответа — идентификатор возвращаемого ответа (равен соответствующему идентификатору команды, однако старший разряд содержит "0", что соответствует ответу);
- длина поля данных — количество байт данных, пересылаемых вместе с ответом ($0 \le M \le 255$); *N* и *M* могут отличаться;
- данные — *M* байт данных, пересылаемых с ответом (не используется, если $M = 0$);
- контрольная сумма — вычисляется на основании идентификатора ответа, длины поля данных и всех байтов данных.

Поскольку можно (и нужно) предположить, что в процессе обмена данными возникнут ошибки, протокол использует символ начала пере-

---

[27] *Контрольная сумма* — это общая методика проверки соответствия данных, полученных адресатом, отправленным данным. Слово "сумма" обусловлено тем, что она обычно вычисляется как тот или иной вариант суммирования отдельных байтов передаваемых данных.

дачи, чтобы сообщить о начале вычисления нового значения контрольной суммы. Эта сумма позволяет убедиться в том, что полученные данные — это именно те данные, которые были переданы, что не только обеспечивает определенный уровень обнаружения ошибок, но и предлагает способ повторной синхронизации данных между передатчиком и приемником в случае возникновения ошибки, поскольку приемник может ожидать символа начала передачи.

Хотя существует много различных способов вычисления контрольной суммы, метод, используемый в наших примерах, заключается в том, чтобы сложить все байты данных между (но не включая) символом начала передачи и передаваемой контрольной суммой, а затем выполнить поразрядную инверсию результата суммирования. Преимущество инверсии суммы — в том, что длинная строка нулей породит, скорее, значение контрольной суммы FFH, чем 00H, что создаст различие между контрольной суммой и данными. Если такого различия не будет, то сигнал, "застрявший" на 0, не будет распознан, поскольку как данные, так и значения контрольной суммы будут нулевыми.

Для дополнительной идентификации команд старший разряд идентификатора устанавливается в "1", а соответствующий идентификатор ответа будет иметь то же содержимое, но с "0" в старшем разряде. Это немного упрощает для разработчика отладку обмена данными, поскольку теперь можно легко выбирать команды и ответные сообщения с помощью анализатора данных. Это также обеспечивает дополнительный способ проверки полученного командного сообщения.

Как уже было упомянуто ранее, очень важно реализовать анализатор протокола в форме конечного автомата, чтобы мы никогда не перешли в состояние, из которого не сможем выйти. Алгоритм работы конечного автомата анализатора протокола, который мы будем использовать, показан на рис. 4.13.

Протоколы со специальными командами

Каждая команда имеет собственный набор данных, которые она передает приемнику. Ответ, посылаемый приемником передатчику после обработки командного сообщения, также зависит от команды. Хотя специальные команды и ответы для разных приложений могут отличаться, всех три области применения, рассматриваемые в этой книге, имеют нечто общее, и дальше мы рассмотрим пример реализации протокола, учитывающего эти общие черты. Обратите внимание на то, что протокол со специальными командами не выполняет никакого контроля ошибок.

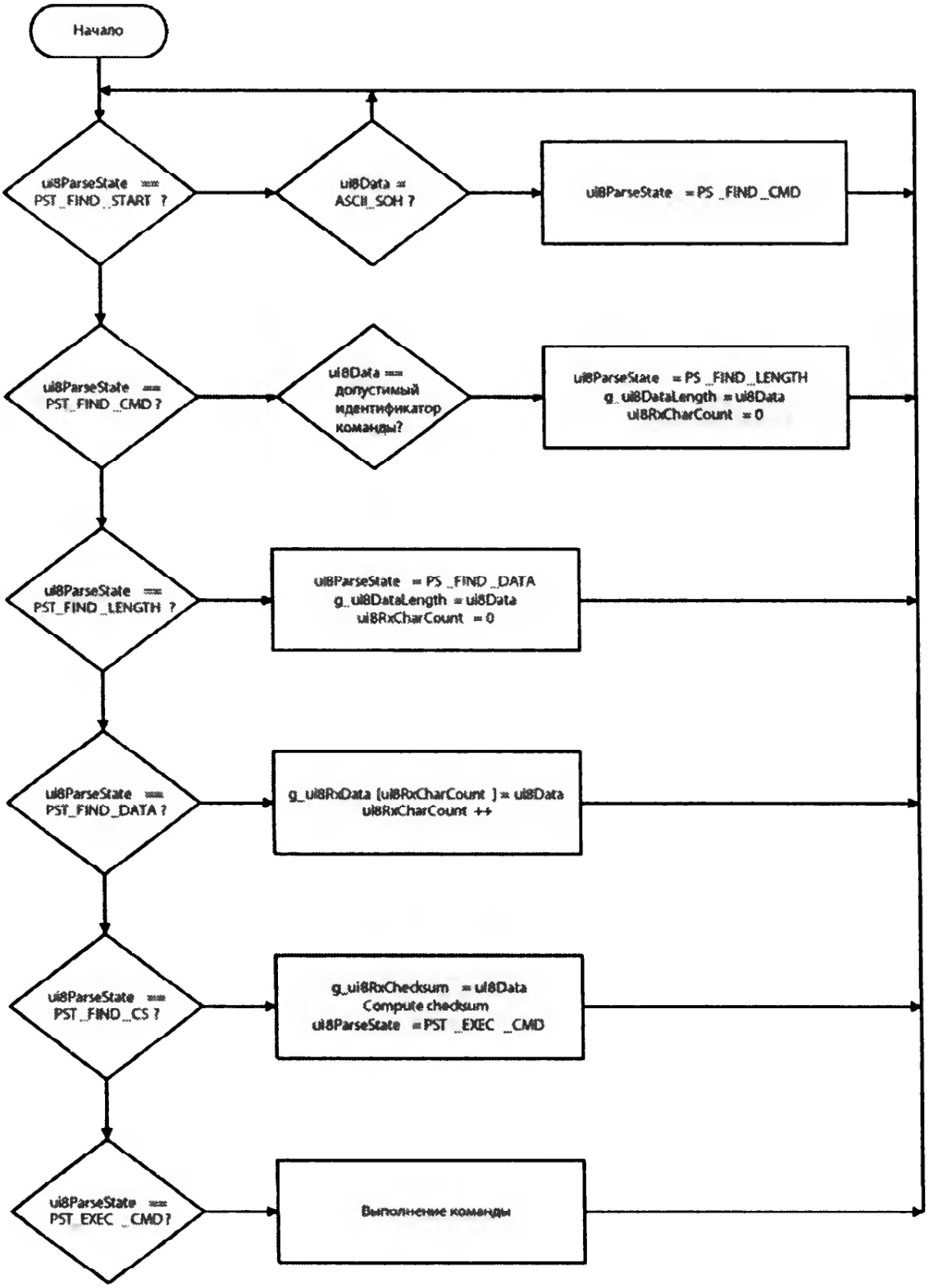


***Рис. 4.13.** Конечный автомат для обработки протокола*

Гарантия точной доставки данных от передатчика к приемнику — ответственность общего протокола обработки сообщений. Как только

данные доставлены, они считаются правильными. Функции, используемые для анализа, конечно же, должны проверять соответствие этих данных конкретной задаче. Например, если центральная система передала значение температурной контрольной точки, равное 1500 °F, а максимально допустимая контрольная точка составляет всего лишь 1000 °F, то полученное значение 1500 °F будет точным (т.е. оно будет правильно передано от передатчика приемнику), но недопустимым (для этого параметра). Форматы данных командных сообщений, которые будут общими для разрабатываемых далее датчиков, представлены в табл. 4.2, а форматы соответствующих ответных сообщений — в табл. 4.3.

Таблица 4.2. Форматы данных командных сообщений

Идентификатор команды	Длина поля данных	Значение параметра
0x80	0	Сообщение о номере версии
0x81	5	Регистрация нижнего калибровочного измерения: Parm 0 — восьмиразрядный номер датчика, начиная с 0; Parm 1 — определенное 32-разрядное нижнее калибровочное значение
0x82	5	Регистрация верхнего калибровочного измерения: Parm 0 — восьмиразрядный номер датчика, начиная с 0; Parm 1 — определенное 32-разрядное верхнее калибровочное значение
0x83	1	Вычисленное калибровочное усиление и смещение: Parm 0 — восьмиразрядный номер датчика, начиная с 0
0x84	1	Сообщение о калибровочном усилении: Parm 0 — восьмиразрядный номер датчика, начиная с 0
0x85	1	Сообщение о калибровочном смещении: Parm 0 — восьмиразрядный номер датчика, начиная с 0
0x86	6	Конфигурирование нижнего безопасного предела: Parm 0 — восьмиразрядный номер датчика, начиная с 0; Parm 1 — 32-разрядное значение безопасного предела; Parm 2 — восьмиразрядный флаг разрешения аварийного оповещения
0x87	6	Конфигурирование верхнего безопасного предела: Parm 0 — восьмиразрядный номер датчика, начиная с 0; Parm 1 — 32-разрядное значение безопасного предела; Parm 2 — восьмиразрядный флаг разрешения аварийного оповещения

Таблица 4.2. Окончание

Идентификатор команды	Длина поля данных	Значение параметра
0x88	1	Сброс аварийных оповещений: Parm 0 — восьмиразрядный номер датчика, начиная с 0
0x89	1	Сообщение о состояниях аварийного оповещения: Parm 0 — восьмиразрядный номер датчика, начиная с 0
0x8A	1	Сообщение о нижнем безопасном пределе: Parm 0 — восьмиразрядный номер датчика, начиная с 0
0x8B	1	Сообщение о верхнем безопасном пределе: Parm 0 — восьмиразрядный номер датчика, начиная с 0
0x8C	1	Сообщение о значении от датчика: Parm 0 — восьмиразрядный номер датчика, начиная с 0
0x8D	1	Установка значения цифрового потенциометра: Parm 0 — восьмиразрядное значение цифрового потенциометра

Таблица 4.3. Форматы данных ответных сообщений

Идентификатор ответа	Длина поля данных	Значение параметра
0x00	6	Информационный отчет о версии: Parm 0 — восьмиразрядное значение месяца выпуска; Parm 1 — восьмиразрядное значение дня выпуска; Parm 2 — восьмиразрядное значение старших двух цифр года выпуска; Parm 3 — восьмиразрядное значение младших двух цифр года выпуска; Parm 4 — восьмиразрядное значение старших двух цифр идентификатора версии; Parm 5 — восьмиразрядное значение младших двух цифр идентификатора версии
0x01	3	Регистрация нижнего калибровочного измерения: Parm 0 — восьмиразрядный номер датчика, начиная с 0; Parm 1 — старший байт двухбайтового дробного значения, замеренного в точке калибровки; Parm 2 — младший байт двухбайтового дробного значения, замеренного в точке калибровки

Таблица 4.3. Продолжение

Идентификатор ответа	Длина поля данных	Значение параметра
0x02	3	Регистрация верхнего калибровочного измерения: Parm 0 — восьмиразрядный номер датчика, начиная с 0; Parm 1 — старший байт двухбайтового дробного значения, замеренного в точке калибровки; Parm 2 — младший байт двухбайтового дробного значения, замеренного в точке калибровки
0x03	3	Вычисление усиления и смещения калибровки: Parm 0 — восьмиразрядный номер датчика, начиная с 0
0x04	5	Сообщение об усилении калибровки: Parm 0 — восьмиразрядный номер датчика, начиная с 0; Parm 1 — старший байт старшего слова четырехбайтового вещественного значения усиления по IEEE-754; Parm 2 — младший байт старшего слова четырехбайтового вещественного значения усиления по IEEE-754; Parm 3 — старший байт младшего слова четырехбайтового вещественного значения усиления по IEEE-754; Parm 4 — младший байт младшего слова четырехбайтового вещественного значения усиления по IEEE-754
0x05	5	Сообщение о смещении калибровки: Parm 0 — восьмиразрядный номер датчика, начиная с 0; Parm 1 — старший байт старшего слова четырехбайтового вещественного значения смещения по IEEE-754; Parm 2 — младший байт старшего слова четырехбайтового вещественного значения смещения по IEEE-754; Parm 3 — старший байт младшего слова четырехбайтового вещественного значения смещения по IEEE-754; Parm 4 — младший байт младшего слова четырехбайтового вещественного значения смещения по IEEE-754
0x06	3	Конфигурирование нижнего безопасного предела: Parm 0 — восьмиразрядный номер датчика, начиная с 0; Parm 1 — 32-разрядный безопасный предел; Parm 2 — восьмиразрядный флаг разрешения аварийного оповещения

Таблица 4.3. Продолжение

Идентификатор ответа	Длина поля данных	Значение параметра
0x07	6	Конфигурирование верхнего безопасного предела: Parm 0 — восьмиразрядный номер датчика, начиная с 0; Parm 1 — старший байт старшего слова четырехбайтового вещественного значения предела по IEEE-754; Parm 2 — младший байт старшего слова четырехбайтового вещественного значения предела по IEEE-754; Parm 3 — старший байт младшего слова четырехбайтового вещественного значения предела по IEEE-754; Parm 4 — младший байт младшего слова четырехбайтового вещественного значения предела по IEEE-754; Parm 5 — восьмиразрядный флаг разрешения аварийного оповещения
0x08	1	Сброс аварийного оповещения: Parm 0 — восьмиразрядный номер датчика, начиная с 0
0x09	3	Сообщение о состояниях аварийного оповещения: Parm 0 — восьмиразрядный номер датчика, начиная с 0; Parm 1 — старший байт двухбайтового слова с флагами аварийного оповещения; Parm 2 — младший байт двухбайтового слова с флагами аварийного оповещения
0x0A	6	Сообщение о нижнем безопасном пределе: Parm 0 — восьмиразрядный номер датчика, начиная с 0; Parm 1 — старший байт старшего слова четырехбайтового вещественного значения предела по IEEE-754; Parm 2 — младший байт старшего слова четырехбайтового вещественного значения предела по IEEE-754; Parm 3 — старший байт младшего слова четырехбайтового вещественного значения предела по IEEE-754; Parm 4 — младший байт младшего слова четырехбайтового вещественного значения предела по IEEE-754; Parm 5 — восьмиразрядный флаг разрешения аварийного оповещения

Таблица 4.3. Окончание

Идентификатор ответа	Длина поля данных	Значение параметра
0x0B	6	Сообщение о верхнем безопасном пределе: Parm 0 — восьмиразрядный номер датчика, начиная с 0; Parm 1 — старший байт старшего слова четырехбайтового вещественного значения предела по IEEE-754; Parm 2 — младший байт старшего слова четырехбайтового вещественного значения предела по IEEE-754; Parm 3 — старший байт младшего слова четырехбайтового вещественного значения предела по IEEE-754; Parm 4 — младший байт младшего слова четырехбайтового вещественного значения предела по IEEE-754; Parm 5 — восьмиразрядный флаг разрешения аварийного оповещения
0x0C	5	Сообщение о значении сигнала от датчика: Parm 0 — восьмиразрядный номер датчика, начиная с 0; Parm 1 — старший байт старшего слова четырехбайтового вещественного значения сигнала по IEEE-754; Parm 2 — младший байт старшего слова четырехбайтового вещественного значения сигнала по IEEE-754; Parm 3 — старший байт младшего слова четырехбайтового вещественного значения сигнала по IEEE-754; Parm 4 — младший байт младшего слова четырехбайтового вещественного значения сигнала по IEEE-754
0x0D	1	Установка значения цифрового потенциометра: Parm 0 — восьмиразрядное значение цифрового потенциометра

4.4. Резюме

В этой главе были рассмотрены три коммуникационных интерфейса, широко используемых при подключении ЦКС dsPIC к периферийным устройствам, сетям и удаленным системам. Хотя интерфейс CAN был затронут только вкратце, читатель получил общее представление о возможностях стандартных и частных протоколов, используемых при подключении датчиков. Вооружившись рассмотренными средствами, разработчик получает прочное основание для построения и расширения коммуникационных возможностей систем, создаваемых на базе dsPIC.

5

Базовые средства разработки для ЦКС dsPIC

Каждый день можно двигаться вперед. Каждый шаг может быть плодотворен. Но для того, чтобы стать на путь реального прорыва, подъема и совершенствования, приходится делать шаги пошире.
— Уинстон Черчилль

Получив в двух предыдущих главах понимание функционирования модулей ЦКС dsPIC, мы готовы сформировать базовый набор программных модулей, который можно использовать для создания гибкой тестовой среды общего назначения, обеспечивающей реализацию приложений для интеллектуальных датчиков. В процессе разработки мы будем применять различные программные средства, а также аппаратные платформы, которые можно получить бесплатно или за незначительную цену у компании Microchip. Конечно же, существуют и другие хорошие средства разработки, предлагаемые сторонними компаниями, однако избранные нами инструменты выдержали три важных практических испытания: они легкодоступны, недороги и эффективны!

5.1. Среда тестирования приложений

Одной из наиболее удручающих реалий нашей жизни заключается в том, что нам постоянно пытаются всучить новое, непроверенное программное обеспечение, работающее на не прошедших тестирование аппаратных средствах. Из-за программных или аппаратных ошибок вся система может накрепко "заглохнуть", и зачастую вы даже не догадываетесь, где искать источник проблем. Во избежание подобного, мы будем разрабатывать наши приложения с помощью макетной платы общего назначения Microchip dsPICDEM 1.1[28], в которой (что очень удобно) применена микросхема dsPIC30F6014A.

---

[28] Номер dsPICDEM 1.1 по каталогу Microchip — DM300014.

Разумеется, в большинстве случаев конечное изделие будет использовать другое аппаратное обеспечение, удовлетворяющее требованиям, которые предъявляются в конкретной постановке задачи к цене, габаритам, электропитанию и функциональности. Тем не менее, изначальная разработка на стандартной, проверенной аппаратной платформе позволяет начать процесс программирования еще до окончательного формирования аппаратного обеспечения системы. Кроме того, это устранит один из главных источников ошибок (аппаратные средства) при отладке приложений.

Мы должны располагать каким-либо способом ввода наших программ в макетную плату для тестирования. Кроме того, в процессе любой серьезной разработки необходимо каким-то образом исследовать работу приложения. Решить обе упомянутые выше задачи позволяет встроенный отладчик ICD 2 и интегрированная среда разработки MPLAB IDE от компании Microchip. Удобная среда MPLAB работает под управлением ПК и включает в себя редактор, имитатор и ассемблер. Хотя для ЦКС dsPIC существует несколько хороших компиляторов с языка C, рассматриваемый в примерах код рассчитан на компилятор Microchip C30 v2.02, учебную версию которого можно бесплатно загрузить с Web-сайта `www.microchip.com`[29]. Одним из преимуществ компилятора C30 является его прямая интеграция с MPLAB, хотя, конечно же, существуют и другие хорошие компиляторы, обладающие этим свойством. Отладчик ICD 2 бесплатно не распространяется[30], зато MPLAB можно загрузить бесплатно с Web-сайта Microchip[31].

Структурная схема среды тестирования на базе ЦКС dsPIC представлена на рис. 5.1.

5.2. Обзор микропрограммной структуры

Итак, мы сформировали среду тестирования. Теперь самое время создать микропрограммную структуру, которая служила бы в качестве основания для разработки приложений в оставшейся части книги.

---

[29] Учебная версия обеспечивает полную функциональность в течении первых 60 дней после установки, а затем поддерживает только минимальную оптимизацию программ. В большинстве случаев оптимизация этого уровня приводит к увеличению объема кода, однако функциональность компилятора не изменяется. Номер полнофункциональной версии компилятора по каталогу Microchip — SW006012.

[30] Номер ICD 2 по каталогу Microchip — DV164005.

[31] Номер MPLAB по каталогу Microchip — SW007002.

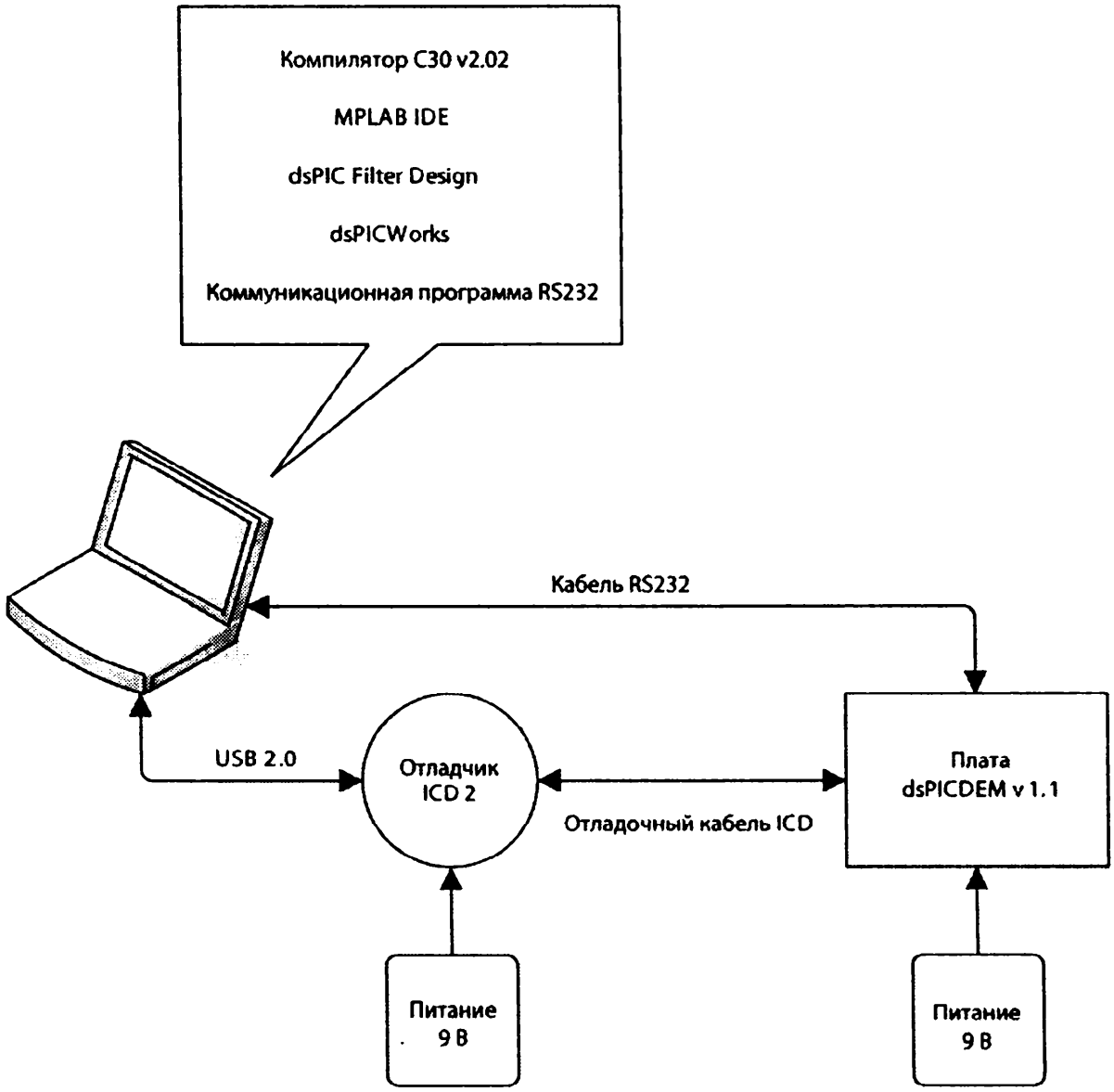


***Рис. 5.1.** Структурная схема среды тестирования на базе dsPIC*

При написании программ мы будем придерживаться перечисленных ниже принципов разработки, гарантирующих, что программные коды надежны и учитывают как нормальный, так и аномальный характер работы системы.

- Коды будут использовать программные конечные автоматы, которые отслеживают ошибки и способны к самовосстановлению в случае возникновения проблем. Эти конечные автоматы должны быть максимально независимы и реализовать простые, четко определенные и надежные интерфейсы с другими программными компонентами приложения.

- Для сокращения времени, необходимого для программирования и отладки приложений, микропрограммное обеспечение должно по возможности использовать стандартные библиотеки, предоставляемые компанией Microchip. Такой подход также облегчает обслуживание приложений, поскольку в них используются заранее оттестированные программные коды[32].
- Код должен быть разбит на уровни — особенно в части коммуникационных интерфейсов и доступа к низкоуровневым аппаратным модулям. Этому гарантирует, что изменения, внесенные в одной части программы, не нарушат другую часть, никак не связанную с первой. Кроме того, разработчик получает максимум гибкости при программировании отдельных областей задачи.
- Весь код должны хорошо документироваться с помощью достаточно обширных, понятных комментариев и стандартизованных соглашений о присвоении имен. Это поможет тем, кто будет сопровождать программы в будущем. Хотя в листингах этой книги многие комментарии изъяты с целью экономии места, в реальных программах, включенных в состав прилагаемого к книге компакт-диска, комментарии представлены полностью.

Из перечисленных принципов особенно важны первый и последний, однако ими часто пренебрегают, поскольку они требуют слишком много времени или кажутся второстепенными. Реализовав программные конечные автоматы, которые способны к самовосстановлению и контролируют ход выполнения операций, мы гарантируем (насколько это возможно), что программа никогда на перейдет в такое рабочее состояние, из которого уже не сможет выйти. Однако для этого требуется воображение и дисциплина, поскольку разработчик должен предвидеть все возможные аномальные условия и создать методики, направленные как на обнаружение этих состояний, так и на восстановление в случае их

---

[32] Как и в любом другом случае, связанном с использованием средств разработки, опытный программист воспринимает выражение "оттестированный код" с долей скептицизма. Хотя программисты обычно безапелляционно заявляют, что проблема — в чьем угодно коде, но только не в их, будет гораздо разумнее предположить, что именно ваш недостаточно протестированный программный код является вероятным виновником возникших сбоев. Если после тщательного тестирования программы вы пришли к выводу, что ошибок в ней нет, то следует проверить, насколько корректно работает "протестированное" программное обеспечение сторонних разработчиков. Помните: присутствие ярлыка "протестировано" еще не означает, что тестирование было выполнено именно в тех специфических условиях, с которыми может быть связана ваша проблема.

возникновения. Но зато потом вы пожнете огромные дивиденды в виде надежности, гибкости и удобства в сопровождении полученных программ.

Одним из ключей эффективной разработки программных конечных автоматов является надлежащее документирование самого конечного автомата. *Внутренним документированием кода* (т.е. информацией, включенной в сам исходный код) зачастую пренебрегают или же чрезмерно сокращают — особенно те программисты, которые ощущают нехватку времени. Как правило, недостаток комментирования сопровождается обещанием сделать это "по окончании проекта", — обещание, которое выполняется далеко не всегда. Даже если по завершении работы над микропрограммным обеспечением комментарии и добавляются, то им обычно не достает глубины. Комментарии получаются самыми правильными, если они создаются непосредственно в процессе программирования, когда программист размышлял над выбором тех или иных команд.

Фактически, отказ от документирования программы в момент ее написания, в лучшем случае, — безответственность, и характеризует программиста как слишком ленивого или слишком подозрительного (если он считает, что недостаток комментариев сохранит его идеи в секрете). Такого программиста вряд ли можно привлекать к разработке серьезных проектов. Временные затраты, связанные с надлежащим документированием программ, окупаются сторицей за счет снижения усилий, необходимых для сопровождения кода в будущем. Кроме того, хорошие комментарии снижают вероятность возникновения несоответствия вносимых изменений принятым по умолчанию (но не задокументированным) соглашениям.

Если вы от кого-то услышите, что документировать программы необязательно, то так и знайте: сказавший это на самом деле не понимает, как нужно разрабатывать программное обеспечение. Программисту очень важно не поддаваться давлению, направленному на увеличение объема незадокументированного кода. В конце концов, когда (именно "когда", а не "если"!) из-за плохо комментированных программ возникнут проблемы, пострадает именно его репутация.

Взяв за основание перечисленные общие принципы разработки, мы переходим к самой структуре приложения. Для упрощения ее понимания мы рассмотрим ее с двух точек зрения:

- с точки зрения потока данных через приложение;
- с точки зрения временных взаимосвязей между различными элементами системы.

Это позволит увидеть как способ преобразования "сырых" сигнальных данных в смысловую информацию, так и последовательность операций, необходимых для реализации такого преобразования. Для начала исследуем поток данных через приложение.

Поток данных через приложение

Поток данных через приложение имеет абсолютно прямолинейный характер (рис. 5.2).

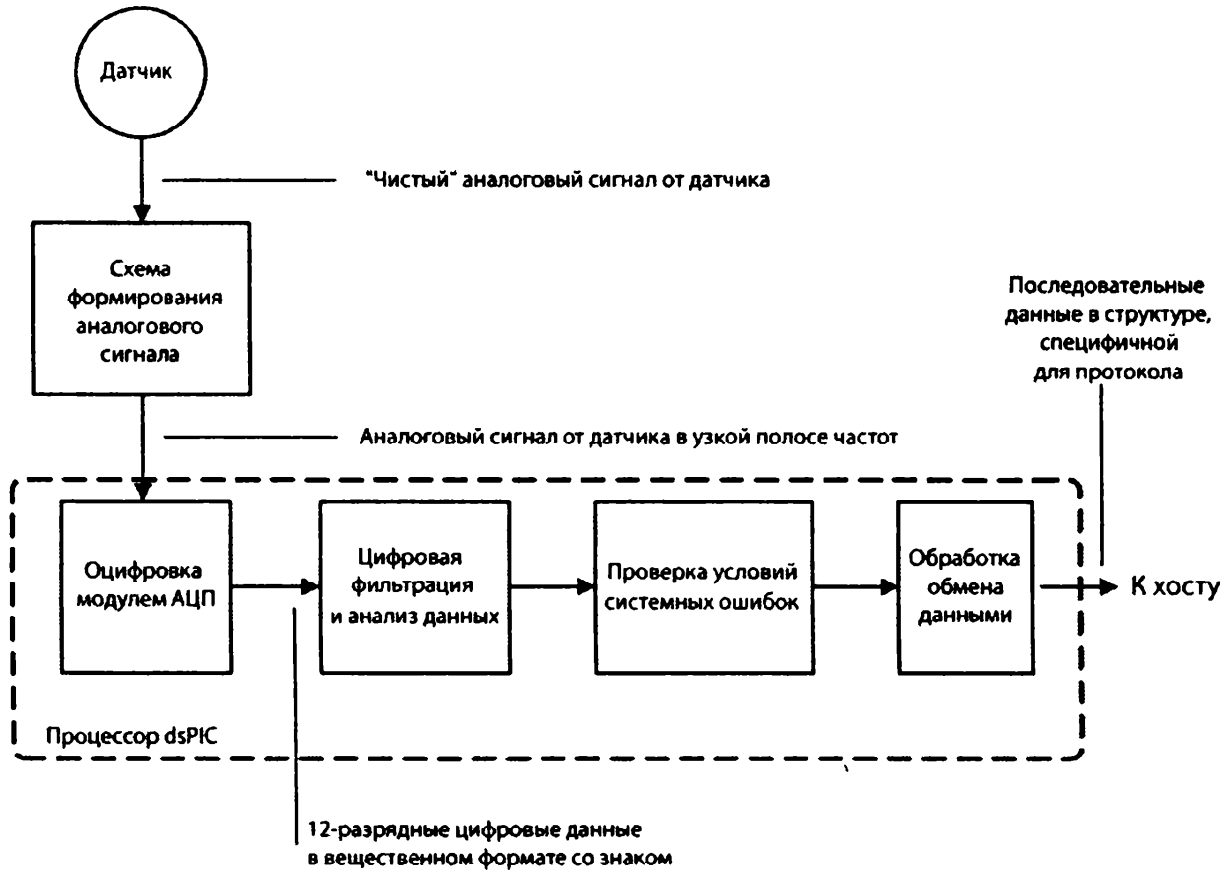


***Рис. 5.2.** Обобщенная схема потока данных через приложение*

"Сырые" аналоговые сигналы от датчика вначале проходят через специализированные схемы формирования аналоговых сигналов, предназначенные для приведения напряжения сигнала к необходимому диапазону, а также для ограничения полосы пропускания частотного наполнения сигнала таким образом, чтобы он соответствовал критерию дискретизации Найквиста. Затем модуль АЦП оцифровывает сформированные аналоговые сигналы и сохраняет дискретные данные в программном буфере, общем для программных конечных автоматов, выполняющих анализ данных. После обработки оцифрованных данных компоненты анализа данных передают полученные ими результаты во

второй общий буфер, который служит входом как для конечных автоматов управления обработкой данных, так и для конечных автоматов формирования отчетности. Модули управления обработкой данных обновляют состояния управляющих выходных сигналов в соответствии со специфичными для прикладной задачи алгоритмами, а модули формирования отчетности передают проанализированные данные компонентам внешних систем для их последующего использования.

Все это выполняется в строгой последовательности:

1. Данные поступают в систему через заданные интервалы времени.
2. Данные подвергаются обработке управляющими алгоритмами и интерфейсами формирования отчетности.
3. Извлекаются очередные дискретные значения.

Нельзя сказать, что управляющие алгоритмы и интерфейсы формирования отчетности должны работать с частотой дискретизации входных данных (зачастую это условие не выполняется), однако между входными и выходными частотами существует прямое, обычно постоянное соотношение. Например, приложение может квантовать входные сигналы 1000 раз в секунду, но выдавать проанализированные данные с частотой всего лишь в 250 выборок в секунду. При этом алгоритмы управления и интерфейсы формирования отчетности могут работать с еще меньшей частотой. Самое главное, чтобы обновление на каждом этапе обработки происходило с выходной частотой предшествующего этапа. При этом каждый компонент всегда должен быть готов к обработке очередной выборки при ее поступлении.

Существует один аспект потока данных, которому присущ асинхронный характер: сообщения об ошибочных состояниях. Такие состояния могут отражать проблемы со входным сигналом, получаемым от датчика (например, значение, лежащее вне ожидаемого диапазона), неполадку в базовом аппаратном обеспечении, ошибку в алгоритме или же просто допустимое, однако нежелательное состояние, определенное на микропрограммном уровне (например, отсоединение кабеля). Вне зависимости от причин возникновения аномальных рабочих условий, структура информирует о них другие элементы системы, зависящей от данных датчика, чтобы те могли предпринять необходимые корректирующие действия. Асинхронную природу обработки ошибок можно увидеть по строкам "обработчика ошибок", который начинается в основных программных модулях и заканчивается в модуле формирования отчетности.

Обратите внимание на то, что коммуникационный канал между модулем формирования отчетности и внешними компонентами системы может быть реализован с помощью различных средств последовательного обмена данными: UART (для стандартных каналов RS-232 или RS-485), CAN, SPI или I^2C (для очень коротких расстояний, наподобие внутриплатных соединений). В то время как фактическая структура отчетов зависит от потребностей приложения и ограничений коммуникационного канала, высокоуровневая концепция системы формирования отчетности и аномальных условий одинакова для всех режимов обмена данными.

Системный поток задач

Итак, рассмотрев поток данных через приложение, мы переходим к исследованию различных задач, решение которых реализует этот поток, а также правильной последовательности их выполнения. Реализация подразумевает следующие основные задачи:

- инициализация программной среды (в нашем случае — среда C реального времени);
- инициализация аппаратных средств системы и программных конечных автоматов таким образом, чтобы они находились в гарантировано безопасном начальном состоянии;
- обслуживание различных прерываний по мере их возникновения и согласованное предоставление информации о них остальной части системы;
- реализация цикла обработки событий, который постоянно отслеживает и соответствующим образом обрабатываем конкретные системные события.

Хотя специфические элементы этой схемы могут изменяться от приложения к приложению (особенно, инициализация системы и обслуживание отдельных источников прерываний), обобщенная структура дает прочное основание для создания приложения. Использование такой структуры повышает эффективность работы (поскольку она уже протестирована) и позволяет уделить больше внимания самому приложению (которое, собственно, и представляет ценность), а не скрытому за ним "водопроводу".

Один из главных вопросов при реализации любого приложения — это структура программного кода. В частности, необходимо обеспечить простоту изменения кода под потребности конкретного приложения, сохранив модульность, которая изолирует остальную часть системы от влияния модификаций. Применение модульного принципа повышает

стабильность системного программного обеспечения, поскольку изменения в одной части программы не влияют на другие части, а в случае возникновения проблем код, приведший к ошибке, может быть относительно легко протестирован.

В программах, рассматриваемых в этой книге, применяется группирование программ, выполняющих сходные задачи (например, фильтрацию или анализ данных), и данных, с которыми они работают, в одном исходном файле `.c` вместе с соответствующим заголовочным файлом, который содержит определения функциональных прототипов, необходимых для использования функций. Подобный подход позволяет разработчику относиться к системе как к набору функциональных объектов, взаимодействующих посредством вызовов функций и общих глобальных данных. Однако во всех других отношениях это — "черные ящики"[33] в том смысле, что, например, код, выполняющий анализ данных, никогда не знает и не беспокоится о том, как происходит фильтрация. Подпрограммы анализа просто работают с отфильтрованными данными.

Модульная структура состоит из следующих десяти основных модулей:

- `Main.c` / `SystemEventDef.h` — точка входа приложения `main()`, которая вызывает подпрограммы системной инициализации и запускает цикл обработки событий;
- `SystemCfg.c` / `SystemCfgDef.h` — высокоуровневая подпрограмма инициализации системы, вызывающая низкоуровневые подпрограммы инициализации АЦП, фильтрации, анализа данных и коммуникационных модулей, а также подпрограммы для выполнения различных системных задач, наподобие чтения-записи данных системной конфигурации и переустановки сторожевого таймера;
- `ADCIF.c` / `ADCIFDef.h` — подпрограммы инициализации интерфейса аналого-цифрового преобразования, обработки данных и обслуживания прерываний, реализующие дискретизацию данных от датчика;
- `Filter.c` / `FilterDef.h` — подпрограммы инициализации фильтров и высокоуровневой фильтрации данных;

---

[33] Термин "черный ящик" относится к системам, для которых известно поведение выходных сигналов при заданном наборе входных сигналов, однако ничего не известно (и не обязательно знать) о внутреннем устройстве этой системы. В качестве примера такой системы можно привести систему зажигания автомобиля. Всем известно, что при повороте ключа зажигания запускается двигатель, однако для большинства людей внутренняя реализация этого процесса остается тайной за семью печатями (и их это, в общем-то, и не интересует).

- `Analysis.c` / `AnalysisDef.h` — подпрограммы анализа данных, которые обрабатывают отфильтрованные данные от датчиков с целью извлечения необходимой информации и формирования отчета, передаваемого по каналам связи;
- `CommIF.c` / `CommIFDef.h` — высокоуровневые интерфейсные подпрограммы, которые инициализируют специфичные для приложения каналы связи и управляют потоком данных через эти каналы;
- `Timer.c` / `TimerDef.h` — подпрограммы инициализации таймера и обслуживания прерываний, которые выполняют критичные по времени задачи обработки данных, планируют процесс дискретизации и фиксируют затраты системного времени;
- `Protocol.c` / `Protocol.h` — высокоуровневые подпрограммы обслуживания протоколов, занимающие промежуточное положение между приложением и коммуникационными подпрограммами из файла `CommIF.c`;
- `Sensor.c` / `SensorDef.h` — подпрограммы для хранения, извлечения и применения информации о конфигурации датчиков к дискретным данным;
- `dsPICDEMIF.c` / `dsPICDEMIF.h` — интерфейсные подпрограммы для обеспечения простого доступа к аппаратным ресурсам макетной платы dsPICDEM 1.1.

При реализации конкретного приложения для большинства из перечисленных модулей потребуется определенная корректировка, чтобы они могли работать со специфичной аппаратной платформой и выполнять характерные задачи приложения. Однако при использовании представленной выше кодовой структуры такая корректировка может быть выполнена поэтапно, что позволяет тщательно оттестировать все изменения. Так, например, при переходе на новую аппаратную платформу первым обычно тестируется программный код инициализации системы. При этом разработчик может изменить только соответствующие подпрограммы, оставив остальную часть приложения без изменений до тех пор, пока код инициализации не будет тщательно протестирован. Как только эта часть будет отлажена, можно переходить к интерфейсу АЦП и т.д., пока все приложение не заработает правильно.

Если между версиями приложения будет заменена аппаратная платформа, то достаточно будет изменить только код в модулях `SystemCfg.c`, `ADCIF.c` и `dsPICDEMIF.c`. Если же необходимо извлечь из данных значения других параметров, то достаточно внести коррективы только в модуль `Analysis.c`. Объектная модульность, когда программный код сгруппирован с соответствующими данными, позволяет

разработчику повторно использовать большие программные фрагменты, что снижает время и стоимость разработки и повышает надежность системы.

Инициализация программной среды

Любая программа, написанная на языке С, должна конфигурировать рабочую среду С сразу же после запуска, до выполнения кода приложения, который начинается с функции `main()`. В процессе инициализации задаются важные рабочие параметры, наподобие размещения стека и кучи[34], и формируются необходимые начальные значения каждой переменной в различных областях данных, генерируемых компилятором, и только после этого запускается само приложение вызовом функции `main()`. Список возможных областей данных с указанием соответствующего типа данных представлен в табл. 5.1.

Таблица 5.1. Области данных и кода, генерируемые компилятором С30

Область	Описание
`.text`	Фактически эта область содержит не "данные" в общепринятом смысле этого слова, а исполняемый коды
`.data`	Одна из двух областей данных, являющихся объектом инициализации. Содержит информацию, необходимую для установки значений глобальных переменных, инициализированных путем явного задания начальных значений. Область `.data` отвечает за инициализацию данных с атрибутом `far` и используется по умолчанию для всех инициализируемых данных, если приложение использует большую модель памяти
`.ndata`	Вторая из двух областей данных, зарезервированная для инициализации данных с атрибутом `near`. Используется по умолчанию при инициализации данных, если приложение использует малую модель памяти
`.const`	Содержит данные, имеющие постоянное значение, например, текстовые строки или числа со спецификатором `const`. Обычно эта область размещена в памяти программы, и доступ к ней организован с помощью окна PSV

[34] *Стек* и *куча* — две области памяти, которые формируются во время выполнения программы (точнее, в момент ее запуска). Стек — это блок памяти, используемый для хранения временных переменных (т.е. локальных переменных функций), параметров, передаваемых в функции, а также программного адреса, по которому необходимо вернуться после выхода из функции. В ходе выполнения приложения размеры стека по мере необходимости могут варьироваться.

Таблица 5.1. Окончание

Область	Описание
.dconst	Аналогична области .const и создается компилятором при определенных условиях, когда приложение использует большую модель памяти
.nconst	Также подобна области .const и создается компилятором при определенных условиях, когда приложение использует малую модель памяти
.bss	Первая из трех областей, содержащих неинициализируемые данные (глобальные переменные). Содержит переменные со спецификатором far. Используется по умолчанию при формировании неинициализируемых данных для приложений, которые создавались с применением большой модели памяти. При инициализации рабочей среды С времени выполнения данные в этой области обнуляются
.nbss	Вторая из трех областей, содержащих неинициализируемые данные (глобальные переменные). Содержит переменные со спецификатором near. Используется по умолчанию при формировании неинициализируемых данных для приложений, которые создавались с применением малой модели памяти. При инициализации рабочей среды С времени выполнения данные в этой области обнуляются
.pbss	Последняя из трех областей, содержащих неинициализируемые данные. Предназначена для переменных, значение которых не подвергается влиянию сброса устройства (например, сброса ЦКС dsPIC). В отличие от данных, содержащихся в областях .bss и .nbss, данные в области .pbss во время инициализации не обнуляются и не принимают каких-либо других значений. Область .pbss размещается в ближней памяти данных.

Обычно код инициализации среды С времени выполнения (или *код запуска*) "прозрачен" для прикладного программиста, поскольку внедрен в программный модуль crt0.o библиотеки libpic30.a компилятора С30. Если по какой-то причине разработчик предпочтет отказаться от инициализации с помощью инициализируемых областей данных (.data, .ndata, .const, .dconst и .nconst), то он может подключить файл crt1.o вместо crt0.o. Единственное различие между этими двумя файлами заключается в том, что crt0.o пропускает этап инициализации данных. Кроме того, если разработчику требуется выполнить какую-либо дополнительную инициализацию, которая просто не может

дожидаться запуска приложения, то он может отредактировать ассемблерный файл любого из двух модулей запуска (`crt0.s` и `crt1.s`, расположенные в подкаталоге `src\pic30` каталога компилятора C30) и включить необходимые добавления, отредактировав связи в новом модуле.

Важно отметить, что образцы исходных файлов `crt0.s` и `crt1.s`, прилагаемые к компилятору, написаны специально для dsPIC30F2010, на что указывают первые две строки после вступительных комментариев:

```
...
.equ _30F2010,  1
.include "p30f2010.inc"
...
```

Для обеспечения переносимости кода между различными устройствами семейства dsPIC, компилятор Microchip C30 использует отдельные включаемые файлы, определяющие адреса регистров и распределение разрядов, специфичные для каждого конкретного процессора. Для проверки того, что программист включил правильный файл для требуемого процессора, каждый включаемый файл в начале определения содержит специальные ассемблерные команды. В том случае, если они не найдут требуемый идентификатор устройства, будет выдано сообщение об ошибке. В представленном выше образце кода первый оператор определяет идентификатор устройства, а второй — включает заголовочный файл для соответствующего процессора.

В том случае, если приложение рассчитано на устройство, отличное от dsPIC30F2010, программист должен указать ассемблеру правильное устройство. Это можно сделать одним из трех способов:

- с помощью ключа командной строки при вызове ассемблера (если программист использует ассемблер в режиме командной строки), например:

  ```
  C:\> AS -p30F6014A
  ```

- разместив директиву `.equ`, за которой следует оператор включения файла, например:

  ```
  .equ _30F6014A, 1
  .include "p30f6014a.inc"
  ```

- указав процессор в диалоговом окне **Select Device** (Выбор устройства) (рис. 5.3), которое открывается из среды MPLAB по команде **Configure ► Select Device** (простейший способ).

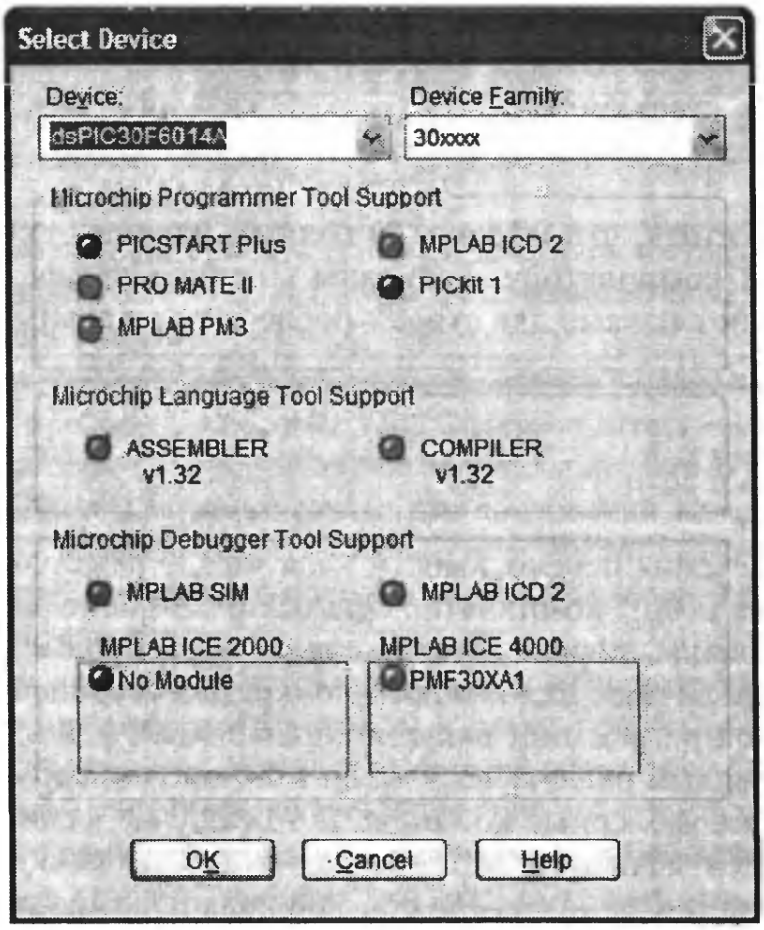


Рис. 5.3.** Диалоговое окно **Select Device

В заключение инициализации рабочей среды С необходимо запустить пользовательское приложение, вызвав функцию `main()`. Именно об этой функции мы дальше и поговорим.

Инициализация системных аппаратных средств и программных конечных автоматов

Код запуска намеренно выполняет минимум операций по инициализации, требуемых для подготовки программной среды и ее запуска. При этом отсутствует какое-либо конфигурирование, ориентированное на приложение. Это позволяет среде С быть максимально гибкой и нейтральной по отношению к платформе, однако теперь ответственность за конфигурирование всех портов ввода-вывода ЦКС dsPIC и встроенных периферийных устройств возлагается на программиста.

Существует начальный, обычно короткий период времени, когда сигналы ввода-вывода находятся в состоянии, принимаемом по умолчанию после сброса. Для одних приложений это приемлемо, для других — нет. В зависимости от назначения конкретного сигнала, могут потребо-

ваться специальные аппаратные средства, гарантирующие, что система будет работать безопасно также и в период между сигналом сброса и программной инициализацией. Если, например, один из сигналов используется для включения мощного лазера, то, конечно же, нежелательно, чтобы он самопроизвольно сработал при запуске системы, поскольку это может привести к тяжелым последствиям. Следовательно, очень важно, чтобы в приложении в первую очередь было инициализировано безопасное состояние портов ввода-вывода.

Конечно же, конфигурирование портов ввода-вывода в большинстве систем — это всего лишь малая часть всей процедуры инициализации. Наилучший подход заключается в том, чтобы начать инициализацию с установки портов ввода-вывода, а затем двигаться дальше, конфигурируя системные ресурсы (и не только ЦКС dsPIC) в том порядке, в котором будет возникать необходимость в этих ресурсах. В зависимости от конкретной рабочей среды, это может даже означать, что приложению потребуется инициализировать какие-то внешние компоненты системы до настройки встроенных модулей. Самое главное — выполнить конфигурирование системы максимально быстро и в том порядке, в котором будет возникать потребность в соответствующих ресурсах. Разработчик должен также гарантировать, что система не начнет отслеживать и обрабатывать системные события до того, как будут успешно инициализированы все системные ресурсы, необходимые для обработки этих событий. Пренебрежение этим очень важным правилом может привести к неустойчивой работе системы и даже к катастрофическому сбою.

Некоторые разработчики предпочитают инициализировать аппаратные средства всех функциональных блоков системы еще до инициализации программных конечных автоматов, которые работают с этими аппаратными компонентами; другие же считают, что лучше инициализировать программный конечный автомат сразу же после конфигурирования связанного с ним аппаратного обеспечения. Выбор одного из подходов должен основываться на природе самого приложения. Например, может оказаться критично важным максимально быстро перевести все аппаратные модули в определенное состояние. По возможности, при инициализации следует применять модульный подход, согласно которому аппаратные и программные средства конкретного функционального элемента системы инициализируются одновременно. Это улучшает модульность программного кода и обеспечивает четко определенный порядок активизации компонентов системы.

В нашем случае используется модульный подход к инициализации, что с одной стороны упрощает кодирование, а с другой — четко иллюстрирует обсуждаемые идеи.

Конфигурирование прерываний dsPIC

Конфигурирование прерываний в нашем случае — процесс, состоящий из четырех этапов.

1. Определение источников прерываний, подлежащих обслуживанию.
2. Задание уровня приоритета для каждого прерывания.
3. Разрешение отдельных прерываний с тем, чтобы они могли быть распознаны контроллером прерываний.
4. Общее разрешение прерываний с тем, чтобы ЦКС dsPIC обрабатывал любой из источников прерываний, который активен и разрешен.

Хотя ни один из перечисленных этапов сам по себе не представляет ничего сложного, вместе взятые они должны быть пройдены с абсолютной точностью, иначе система окажется ненадежной. Хуже всего, когда один или больше этапов выполнены *почти* правильно, поскольку это может привести к промежуточным системным сбоям, которые крайне сложно выявить, смоделировать и устранить.

В наших приложениях мы будем иметь дело с тремя группами источников прерываний: системные таймеры, АЦП и каналы связи. Из перечисленных источников прерываний наиболее критичны — системные таймеры и АЦП, поскольку они организуют поток данных через систему. Каналы связи, хотя и важны, могут обслуживаться с более низким приоритетом, поскольку их функциональность не требует настолько узкого временного окна, как в случае с двумя другими группами источников прерываний. Если нам придется временно буферизировать входной символ, то это — не большая проблема, а вот если дискретизация станет апериодической, то это сразу же отразится на производительности системы.

Ниже перечислены обслуживаемые источники прерываний в порядке их приоритетности:

- Timer 2/3 — 32-разрядный режим таймера на переменной частоте (таймер дискретизации для АЦП);
- Timer 1 — 16-разрядный режим таймера с интервалом прерываний 10 мс (общий системный таймер);
- ADC Sample Ready — сообщает о готовности полной выборки данных к фильтрации;

- UART 1 Rx Data Ready — прием данных по UART 1 ожидает готовности канала связи;
- UART 1 Tx Holding Register Empty — в канале связи активен передатчик.

5.3. Реализация структурных модулей

Итак, от обсуждения кода запуска переходим к реализации программной структуры, и начнем со входной точки приложения: функции `main()`. Как было упомянуто в предыдущем разделе 5.2, функция `main()` вначале инициализирует системные компоненты, а затем запускает цикл обработки событий. Этот цикл непрерывно отслеживает события, которые генерируются подпрограммами обслуживания прерываний или возникают в результате обработки данных, выполняемой в рамках самого цикла контроля событий. Эти системные события, в действительности, представляют собой обычные флаги в виде разрядов глобальной 16-разрядной переменной `g_vui16SysEvent`[35], а сами события определены в заголовочном файле `SystemEventDef.h`. Установка флага соответствует возникновению события, которое необходимо обработать, а сброс (обнуление) — завершению обработки события.

В листинге 5.1 представлена функция `main()`, вызывающая процедуру инициализации системы `SystemInit()`.

Листинг 5.1. Пример функции `main()`

```
Int16 main (void)
{
// Локальные переменные
Int16
   ui8AnalysisCount, // Счетчик прореживания для
                     // планирования анализа данных
   ui8RxData,        // Принятые коммуникационные данные
   ui8Status;        // Состояние выполнения функции

// Инициализация системных аппаратных средств
// и связанных с ними программных конечных автоматов
```

[35] Переменная `g_vui16SysEvent` иллюстрирует соглашение об именовании переменных, которое соблюдается во всех генерируемых программных кодах. В общем случае имени переменной предшествует аббревиатура ее типа. Так, например, `vui16` означает "volatile unsigned 16-bit value" ("переменное беззнаковое 16-разрядное значение"). В именах глобальных переменных используется префикс `g_` (от слова "global"), который указывает на то, что переменная глобальная, а не локальная.

Листинг 5.1. Продолжение

```
SystemInit();

// Обработка системных событий при их возникновении
while (FOREVER)
{
// Существуют ли какие-нибудь ожидающие системные события?
// Начинаем с проверки наличия дискретных данных, готовых
// к фильтрации

  if (g_vuil6SysEvent)
  {
    // Да. Дискретные данные готовы к фильтрации?
    if (g_vuil6SysEvent & EVT_FILTER)
    {
       // Да, дискретные данные готовы, поэтому сбрасываем
       // флаг события и фильтруем выборку

       g_vuil6SysEvent &= ~EVT_FILTER;
       ui8Status        = FilterData();
       g_vuil6SysEvent |= EVT_ANALYZE; // Да, сигнализируем
                                       // о событии "Анализ данных"
    }

    // Присутствуют ли отфильтрованные данные для анализа?
    if (g_vuil6SysEvent & EVT_ ANALYZE)
    {
       // Да, поэтому сбрасываем флаг события
       // и выполняем анализ

       g_vuil6SysEvent &= ~EVT_ANALYZE;
       ui8Status        = AnalyzeData();

       // Показал ли анализ наличие условий, о которых
       // необходимо сформировать сообщение?

       ui8AnalysisCount++;       // Прибавляем 1 к счетчику
                                 // прореживаний данных

       if (ui8AnalysisCount >= ANALYSIS_TIME)
       {
         ui8AnalysisCount = 0; // Сброс счетчика прореживаний
         g_vuil6SysEvent |= EVT_REPT_RESULTS; // Флаг о
                           // наличии результатов для отчета
       }
    }
```

Листинг 5.1. Продолжение

```
// Существуют ли результаты анализа, о которых
// необходимо сообщить хосту?

if(g_vuil6SysEvent & EVT_REPT_RESULTS)
{
  // Да, поэтому сбрасываем флаг события и сообщаем
  // о результатах через коммуникационный канал

  FormatResultsMsg(g_ui8ResultsMsg,
                   &g_uil6ResultsMsgLength);

  if (g_ui16ResultsMsgLength <= CommGetTxFreeCount())
  {
    // В очереди передачи есть свободное место, поэтому
    // добавляем сообщение о результатах и сбрасываем
    // флаг события, чтобы указать, что оно обработано

    CommPutBuff(g_ui8ResultsMsg, g_uil6ResultsMsgLength);
    g_vuil6SysEvent &= ~EVT_REPT_RESULTS;
  }
}

// Был ли сигнал таймера об истечении 100 мс?
if (g_vuil6SysEvent & EVT_TIMER)
{
  // Да, сбрасываем флаг события и выполняем требуемую
  // обработку сигнала таймера

  g_vuil6SysEvent &= ~EVT_TIMER;

  // Здесь размещаем программный код для любой
  // дополнительной обработки, если она нужна
}

// Проверяем, получили ли мы какие-либо
// данные от хоста

if (CommGetRxPendingCount() > 0)
{
  // Да, получили, поэтому считываем следующий
  // символ и обрабатываем его

  ui8Status = CommGetRxChar(&ui8RxData);
  if (ui8Status == ST_OK)
    CommProcRxChar(ui8RxData, &g_ui8CommParseState);
}
```

Листинг 5.1. Окончание

```
    //  Сбрасываем сторожевой таймер, чтобы он не сработал

    ResetWatchdogTimer();
  }
  // Мы не должны НИКОГДА оказаться в этом месте программы,
  // если система работает нормально. Если управление
  // передается сюда, то программа завершится, и код запуска
  // выполнит сброс устройств

  return ST_SYSTEM_FAIL;
}
```

В отношении инициализации системы следует отметить две вещи:

- она происходит перед всем остальным, за исключением размещения локальных переменных;
- детали инициализации оформлены в виде отдельной функции, вместо того, чтобы быть включенными в состав функции `main ()`.

Первое гарантирует, что аппаратные средства в результате инициализации будут приведены в безопасное исходное состояние максимально быстро, в то время как второе является примером хорошего стиля разработки программного модуля, относительно нейтрального по отношению к платформе (по крайней мере, на высоком уровне).

Подробнее программный код инициализации системы мы рассмотрим немного позже, а сейчас вернемся к функции `main()`. После инициализации системы запускается цикл обработки событий, который постоянно отслеживает системные события и обрабатывает их, а также контролирует и обрабатывает любые данные, принятые по каналу связи и сбрасывает сторожевой таймер. Цикл продолжается вплоть до сброса процессора.

Реализованный цикл за время одного прохода может контролировать и обрабатывать множество событий. Если исходить из наихудшего сценария, когда на протяжении одного прохода цикла придется обработать все возможные события, а также проверить и обработать все принятые по каналам связи данные, то возможно понадобиться изменить структуру цикла таким образом, чтобы за время одного прохода обрабатывалось только одно событие. Это легко сделать, просто изменив последовательности операторов `if()` на `if()...else if()`. Единственный критерий, которого необходимо придерживаться, заключается в том, что сторожевой таймер должен сбрасываться до его срабатывания, иначе произойдет сброс всей системы. Обычно мы рассчитываем на такой сброс только в том случае, когда программа "зависает" в оши-

бочном состоянии, а не потому, что затратили немного больше времени на проход основного цикла.

Для лучшего понимания инициализации системы рассмотрим подробнее функцию `SystemInit()`. Конфигурационный код использует функции из библиотеки обслуживания периферийных устройств ЦКС dsPIC30F[36], которая поставляется совместно с компилятором C30. Ее также можно загрузить с Web-сайта Microchip. Документация на эту библиотеку находится в документе "16-Bit Language Tools Libraries" ("16-разрядные библиотеки языковых средств")[37], размещенном в подкаталоге `MPLAB C30\Docs` каталога установки компилятора. Этот документ также может быть загружен с Web-сайта Microchip.

Еще одной особенностью всех наших систем является их способность выполнять усложненную цифровую фильтрацию с помощью функций из библиотеки ЦОС от Microchip. При этом в большинстве случаев требуется использовать фильтры с конечной импульсной характеристикой (КИХ-фильтры). Для реализации такой фильтрации приложение должно вначале создать и инициализировать структуру данных для каждого фильтра. Эта структура называется `FIRStruct`, а ее программное определение находится в поставляемом совместно с компилятором заголовочном файле `dsp.h`. Она содержит всю информацию, необходимую для обеспечения линии задержки с отводами, а также для применения коэффициентов фильтрации.

Создание упомянутой структуры заключается в размещении вектора фильтрующих коэффициентов (с его инициализацией) и вектора линии задержки с отводами. Затем необходимо объявить КИХ-структуру как глобальную переменную или же — как локальную переменную внутри функции, область действия которой совпадает с периодом существования фильтра. Как правило, автор предпочитает первый подход, поскольку даже в случае изменения выполняемых ветвей кода приложения фильтр всегда будет находиться в области действия переменной. Если же фильтр будет объявлен как локальная переменная, а выполняемые ветви кода приложения в будущей версии программы изменятся, то структура фильтра может временно оказаться за пределами области действия переменной. В результате возникает вероятность затирания структуры в памяти другими данными и появления ошибочных резуль-

---

[36] Номер библиотеки обслуживания периферийных устройств ЦКС dsPICC30F по каталогу Microchip — SW300021.

[37] Номер руководства "16-Bit Language Tools Libraries" на Web-сайте Microchip DS51456C.

татов, когда потребуется опять воспользоваться структурой данных фильтра.

Одно из главных преимуществ программы dsPIC Filter Design заключается в том, что она одним нажатием клавиши генерирует ассемблерный код, который управляет созданием структур фильтров и связанных с ними линий задержки с отводами, а также фильтрующих коэффициентов. Например, для создания фильтрующего кода для нашего приложения, использующего термопару, необходимо выполнить следующие действия в окне **Filter Design**.

1. В главном меню выбрать пункт **Design ► FIR Window Design** (Разработка ► Окно КИХ-разработки) (рис. 5.4).

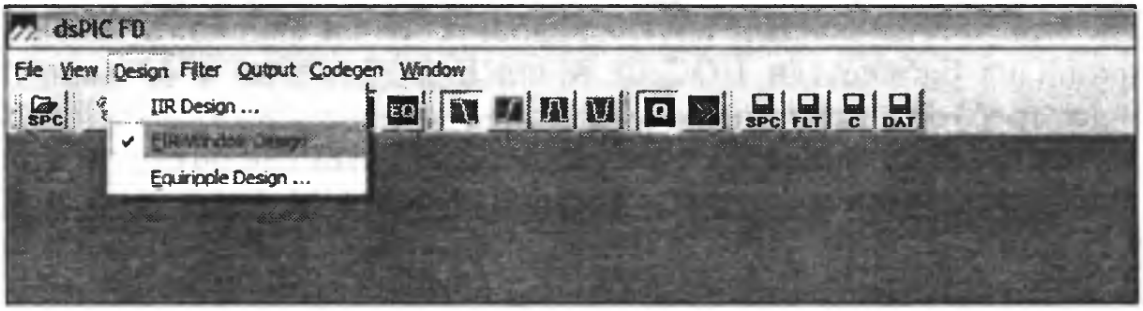


Рис. 5.4. Выбор пункта меню FIR Window Design

2. Программа отобразит первое окно разработки фильтра (рис. 5.5). Выберите переключатель **Lowpass** (ФНЧ) и нажмите кнопку **Next** (Далее).

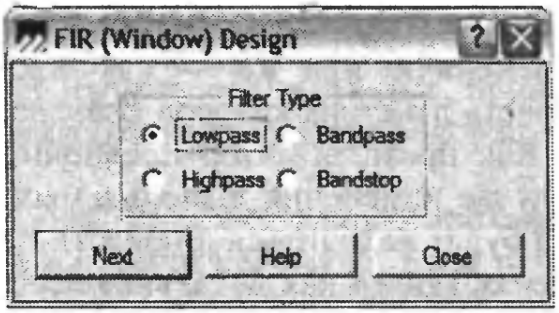


Рис. 5.5. Перове окно КИХ-разработки

3. Введите параметры фильтра согласно рис. 5.6 и нажмите кнопку **Next**, чтобы перейти к третьему окну разработки фильтра.
4. Выберите тип фильтра, который хотите использовать. Обратите внимание на количество отводов, необходимое для каждого типа фильтра для соблюдения требований фильтрации, заданных в предыдущем окне. На рис. 5.7 показано, что для нашего приложения выбран фильтр Гаусса, требующий 51 отвод. Обратите также вни-

мание на то, что в случае необходимости можно точно указать требуемое количество отводов вручную.

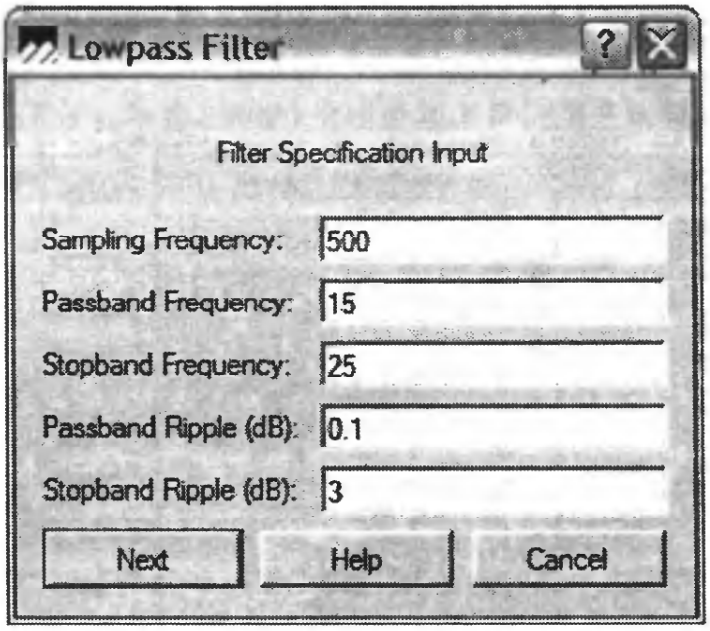


Рис. 5.6. Второе окно КИХ-разработки

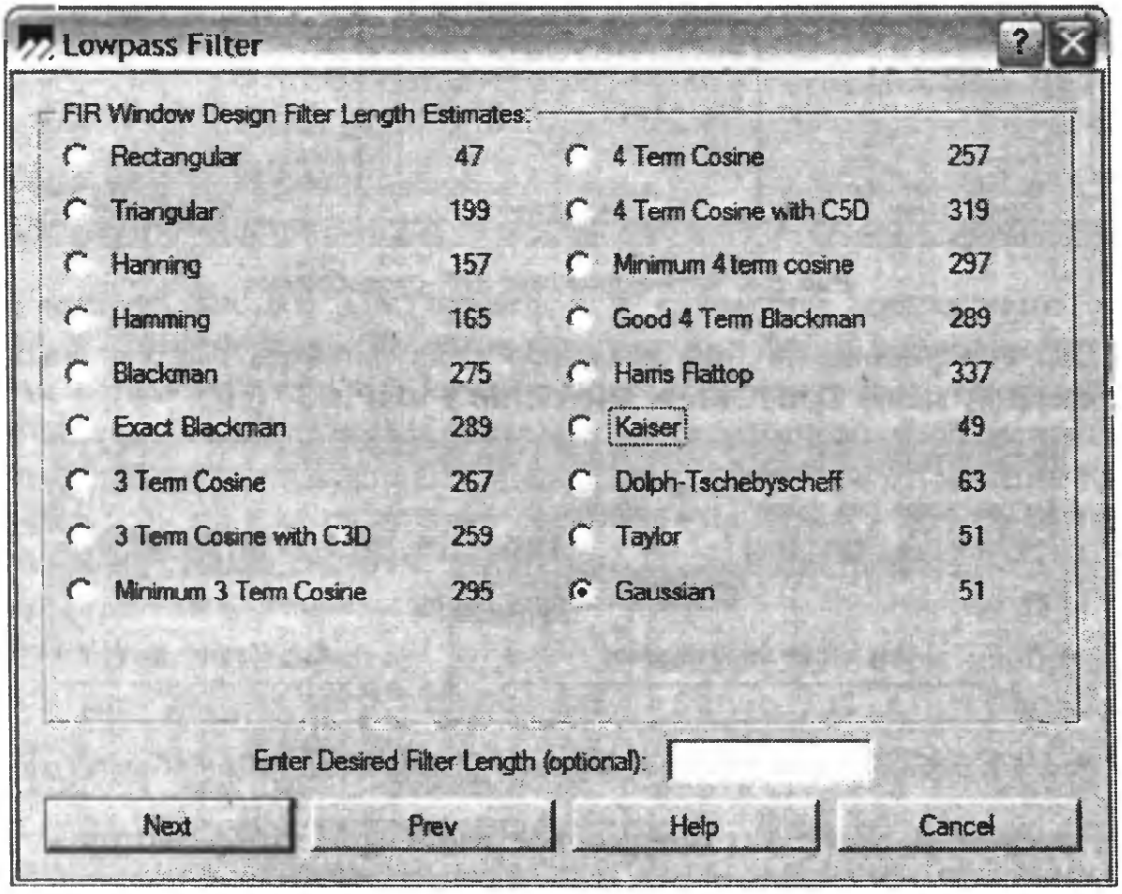


Рис. 5.7. Третье окно КИХ- разработки

5. После нажатия кнопки **Next** на экране появятся результирующие кривые характеристик фильтра (рис. 5.8). На данном этапе фильтр уже создан, однако еще не сгенерирован никакой программный код.

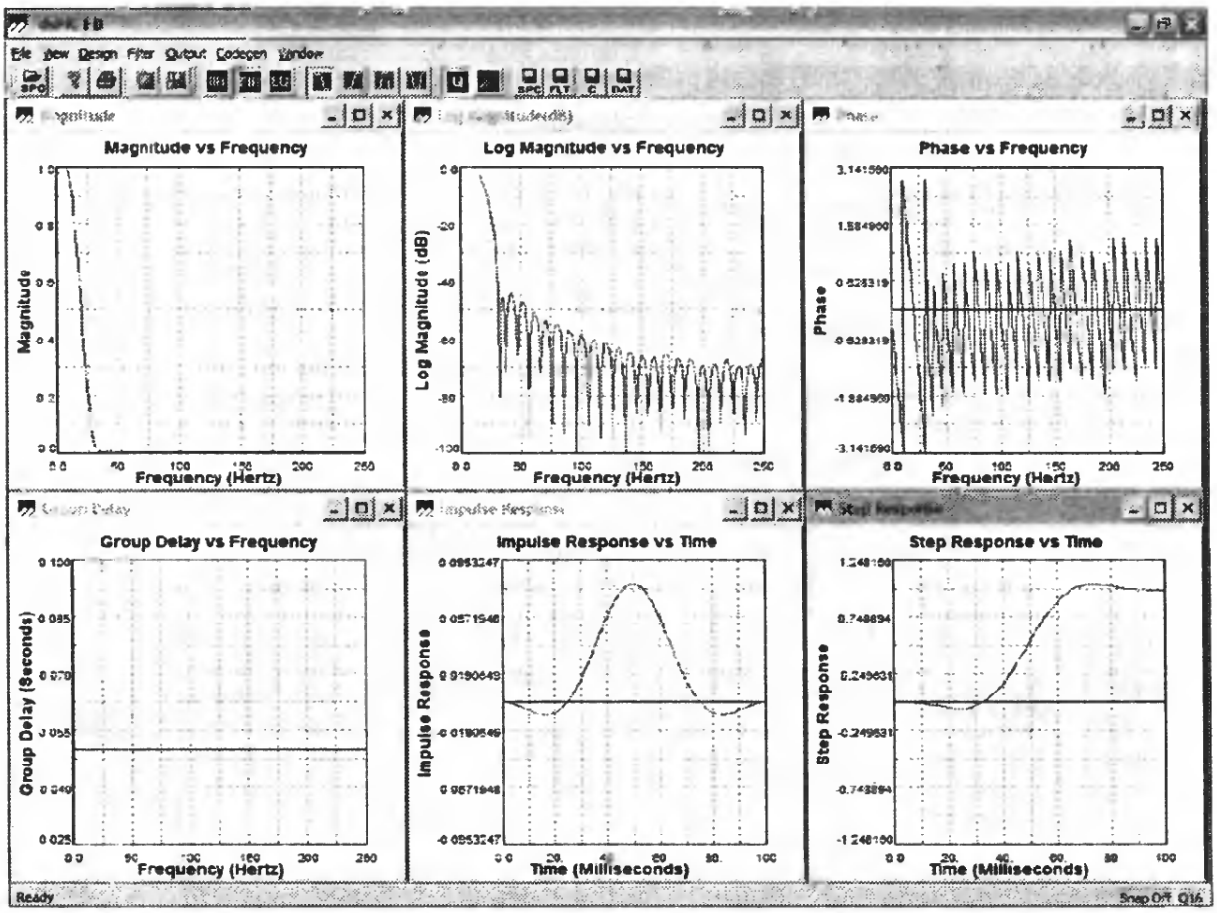


***Рис. 5.8.** Четвертое окно КИХ-разработки*

6. Для генерирования программного кода фильтра следует выбрать команду меню **CodeGen ► Microchip ► dsPIC30** (рис. 5.9).

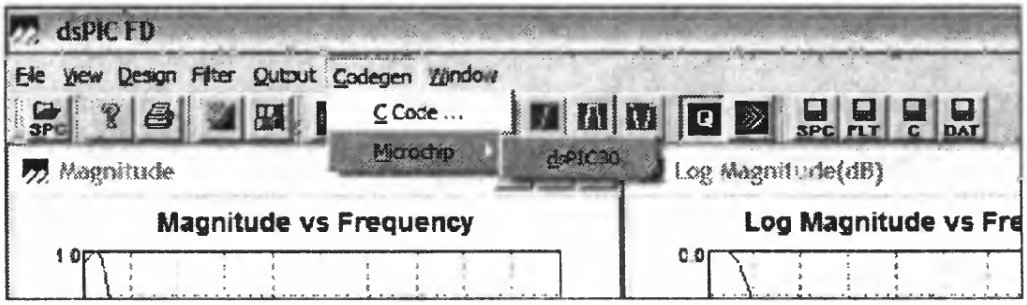


***Рис. 5.9.** Выбор команды для генерирования программного кода КИХ-фильтра*

7. Программа отобразит диалоговое окно (рис. 5.10), которое дает пользователю возможность выбрать необходимые опции генерирования кода. В нашем случае мы используем установку по умолча-

нию: переключатели **Use General Subroutine** (Использовать общую подпрограмму) и **XData Space** (Пространство XData), — а также установим флажок **C Header File and Sample Calling Sequence (.h)** (Заголовочный файл С и последовательность выборки), указывающий программному обеспечению на необходимость создания соответствующего заголовочного файла С.

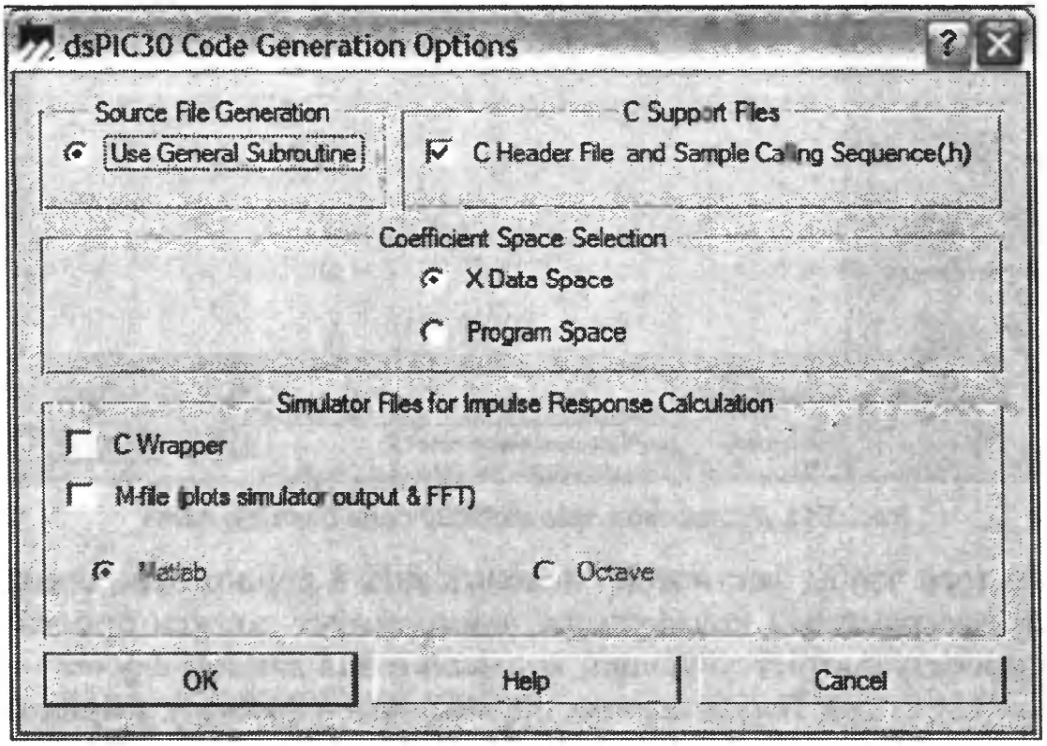


***Рис. 5.10.** Диалоговое окно опций генерирования программного кода*

8. Нажатие кнопки **OK** приведет к открытию диалогового окна **dsPIC30 code base file name** (Базовое имя файла программного кода dsPIC30) (рис. 5.11), которое позволяет задать имя сгенерированного файла. Поскольку генератор программного кода использует имя файла в качестве базового для создаваемой структуры КИХ-фильтра, это имя не должно содержать пробелов и знаков пунктуации. Введите в качестве имени файла `SensorFilter` (Фильтр датчика) и нажмите кнопку **OK**.
9. Программа сгенерирует несколько файлов. При этом наибольший интерес для нас представляют файлы `SensorFilter.h` (заголовочный файл С, определяющий структуру нашего фильтра для использования другими С-файлами) и `SensorFilter.s` (содержит ассемблерный код, описывающий структуру фильтра, а также связанные с ним коэффициенты и буферы линий задержки с отводами).

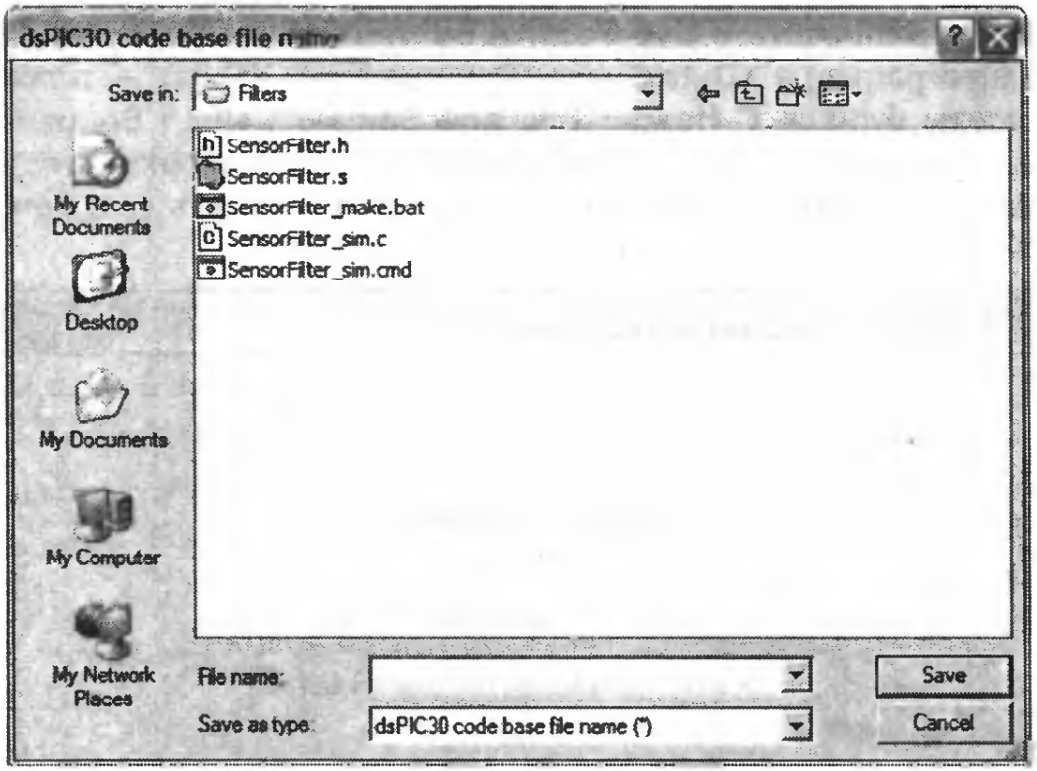


Рис. 5.11. Диалоговое окно dsPIC30 code base file name

Для того чтобы фактически использовать в приложении сгенерированный программный код фильтра, программист должен просто добавить к проекту соответствующий код фильтра (в данном случае — файл `SensorFilter.s`). Перед первым обращением к фильтру в приложении он должен быть инициализирован с помощью вызова функции `FIRDelayInit()`, которая инициализирует линию задержки с отводами в надежное состояние. Обратите внимание на то, что в случае использования кода, сгенерированного программой Filter Design, для инициализации соответствующей структуры нет необходимости вызывать функцию `FIRStructInit()` — эту задачу выполняет сгенерированный код. Если не выполнить инициализацию линию задержки с отводами, то это может вызвать проблемы с характеристикой фильтра вплоть до того момента, пока новые данные не смогут пройти линию задержки.

Как только фильтр инициализирован, его очень просто применить к новым данным. Как только блок новых данных поступит на фильтр, приложение вызовет функцию `FIR()`, передав ей следующие указатели:

- на новые данные для фильтра;
- на целевой буфер данных, который будет содержать результирующие отфильтрованные значения;
- на задействованную структуру фильтра.

Поскольку функция FIR() управляет обновлением соответствующей линии задержки с отводами, с приложения снимается ответственность за решение этой задачи.

И в заключение — еще одно важное замечание в отношении использования функций из библиотеки ЦОС. Если выполнение библиотечной функции прерывается, то прерывающая подпрограмма должна обеспечить восстановление содержимого регистра состояния, а также значений в командах DO и REPEAT в том состоянии, в котором они находились до прерывания. В противном случае функция ЦОС может Вернуть неверные результаты. Функции библиотеки ЦОС используют некоторые общие аппаратные ресурсы и в случае, если подпрограмма обслуживания прерывания изменит конфигурацию этих ресурсов, могут возникнуть проблемы с той частью функции ЦОС, которая выполняется после завершения обработки прерывания. Для того чтобы упростить задачу разработчика, в описании каждой библиотечной функции ЦОС представлен перечень используемых ею ресурсов.

5.4. Резюме

В данной главе был представлен обзор программной структуры, которая будет использоваться в последующих трех главах. В каждой из этих глав будет показано реальное применение ранее рассмотренных принципов. Хотя до сих пор упомянутая структура подвергалась лишь схематичному обзору, уже в следующей главе этот "костяк" обрастет "мясом", поскольку мы приступаем к разработке датчика температуры.

6

Реальный пример: датчик температуры

Не важно, какая температура в комнате, — она всегда будет комнатной температурой.

— Стивен Райт

В каждой из последующих трех глав (с 6 по 8) разрабатывается завершенная сенсорная система, измеряющая какой-нибудь типичный параметр и передающая результаты измерений в центральную систему. Данная глава посвящена измерению температуры, в главе 7 разрабатывается система контроля давления и нагрузки, а в главе 8 — датчик потока. Разумеется, каждый пример сосредоточен на конкретном типе датчиков, однако излагаемые концепции применимы к широкому диапазону чувствительных элементов. Зачастую для учета уникальных требований конкретного датчика, используемого для измерений интересующего параметра, достаточно внести лишь минимальные коррективы. Читателю предоставляется полная свобода в развитии излагаемых здесь идей (например, добавление управляющего алгоритма, который обрабатывает результаты измерений параметров для достижения требуемого эффекта). Итак, после этого краткого введения переходим к вопросу температуры — наверное, наиболее часто измеряемого физического параметра в мире.[38]

Важность температуры вытекает из ее влияния на большое количество ситуаций, связанных как с окружающей средой, так и с технологическими процессами. Нам (или, точнее сказать, — тому оборудованию, которое мы используем) требуются точные замеры температуры при обогреве или охлаждении домов, при управлении двигателями автомобилей, при приготовлении пищи, при промышленной обработке материалов, при контроле жизненно важных симптомов пациентов и т.д. —

---

[38] Статья "Temperature Technology Claimed to Eliminate Ambient Variations", журнал "InTech", декабрь 1998.

этот перечень можно продолжать бесконечно. Напомним, что в главе 1, "Что такое интеллектуальные датчики, и зачем они нужны?", был Рассмотрен пример такого устройства — обычный ртутный термометр, в котором абсолютную температуру определяют по уровню жидкой ртути. К сожалению, хрупкость датчиков этого типа (обычно столбик ртути заключен внутри стеклянной трубки), а также токсичность его чувствительного элемента (ртуть очень ядовита) ограничивает область применения ртутных термометров узким кругом хорошо контролируемых обстоятельств.

Сенсорная система, к разработке которой мы приступаем, использует чувствительный элемент, гораздо более подходящий для широкого круга сред и применений. Речь идет о термопаре. Однако прежде, чем углубится в подробности применения термопар, взглянем на разнообразие существующих чувствительных элементов, применяемых для измерения температуры.

6.1. Типы датчиков температуры

За много лет ученые разработали множество специализированных чувствительных элементов, реагирующих на абсолютное значение температуры или на ее изменение варьированием некоторого физического свойства. В этой книге нас интересуют прежде всего чувствительные элементы, которые можно контролировать с помощью электроники. Причина же очень проста: соблюдение этого условия позволяет нам легко преобразовывать контролируемое физическое свойство в замеры, которыми можно манипулировать с помощью ЦКС dsPIC. Этим не означает, что все другие подходы (например, визуальное наблюдение состояния ртутного термометра) неверны — просто мы хотим сказать, что все неэлектрические методики контроля температуры неприменимы для наших целей по причине их несовместимости с задействованными платформами мониторинга. Как известно, две доски можно скрепить с помощью гвоздей или шурупов, однако, если у вас под рукой есть только молоток, вы, скорее всего, предпочтете именно гвозди.

В настоящее время существует пять популярных типов чувствительных элементов, выходные сигналы которых можно отслеживать электрическим способом. Развитие технологий, безусловно, расширит этот список, однако в настоящее время большинство датчиков температуры с электрическим контролем относится к одному из пяти типов:

- термопары;
- термопреобразователи сопротивления (RTD — Resistance Temperature Detector);

- терморезисторы,
- кремниевые датчики;
- инфракрасные датчики.

Хотя мы обсудим каждый из этих типов датчиков в следующих подразделах, приложение, которое мы разрабатываем в данной главе, использует термопару. Выбор пал именно на термопары, поскольку они широко распространены, хорошо изучены, точны (при правильной эксплуатации) и относительно недороги (одна из причин их популярности).

Термопары

Термопары — это двухпроводные чувствительные элементы, использующие *термоэлектрический эффект* для измерения температуры спая двух проводов. Термоэлектрический эффект, открытый ученым Томасом Зеебеком (Thomas Seebeck) в 1821 году[39], заключается в возникновении электрического напряжения на спайке двух разных металлов, значение которого зависит от температуры области перехода между металлами. Хотя это напряжение очень незначительное (порядка нескольких микровольт на каждый градус температуры), можно создать системы, которые будут точными в широким диапазоне температур, если применить соответствующие методики обработки аналоговых и цифровых сигналов.

Очень малые напряжения, производимые термопарами (порядка милливольт) требуют наличия в системах надежного заземления и экранирования во избежание внесения в замеренное напряжение неприемлемых уровней помех. Кроме того, поскольку проводники печатной платы выполнены из металла, отличного от металлов термопары, сама схема, задействованная для измерения первичного термоэлектрического напряжения, формирует дополнительный переход Зеебека, выходной сигнал которого изменяется в зависимости от температуры! В заключение следует отметить, что термопары в своем диапазоне измерений могут быть в высшей мере нелинейными. Так, зависимость выходных сигналов различных термопар от температуры показана на рис. 6.1.

Учитывая все эти серьезные ограничения, почему вообще используют термопары для измерения температуры? Термопары популярны по трем причинам: они относительно недороги и работают в широком диа-

---

[39] Термоэлектрический эффект описан во многих источниках, и один из них — книга "Electromagnetic & Electromechanical Machines" ("Электромагнитные и электромеханические машины"), 2nd Edition, by Leander W. Matsch. Copyright 1977 by Thomas Y. Crowell Company, Inc.

пазоне температур, — а их недостатки могут быть компенсированы (как будет показано в подразделе "Специфика" раздела 6.2, "Ключевые аспекты измерения температуры"). Даже несмотря на то, что термопары требуют дополнительных схемных и программных решений, они позволяют создавать надежные датчики, стоимость и рабочие характеристики которых соответствуют требованиям многих прикладных задач.

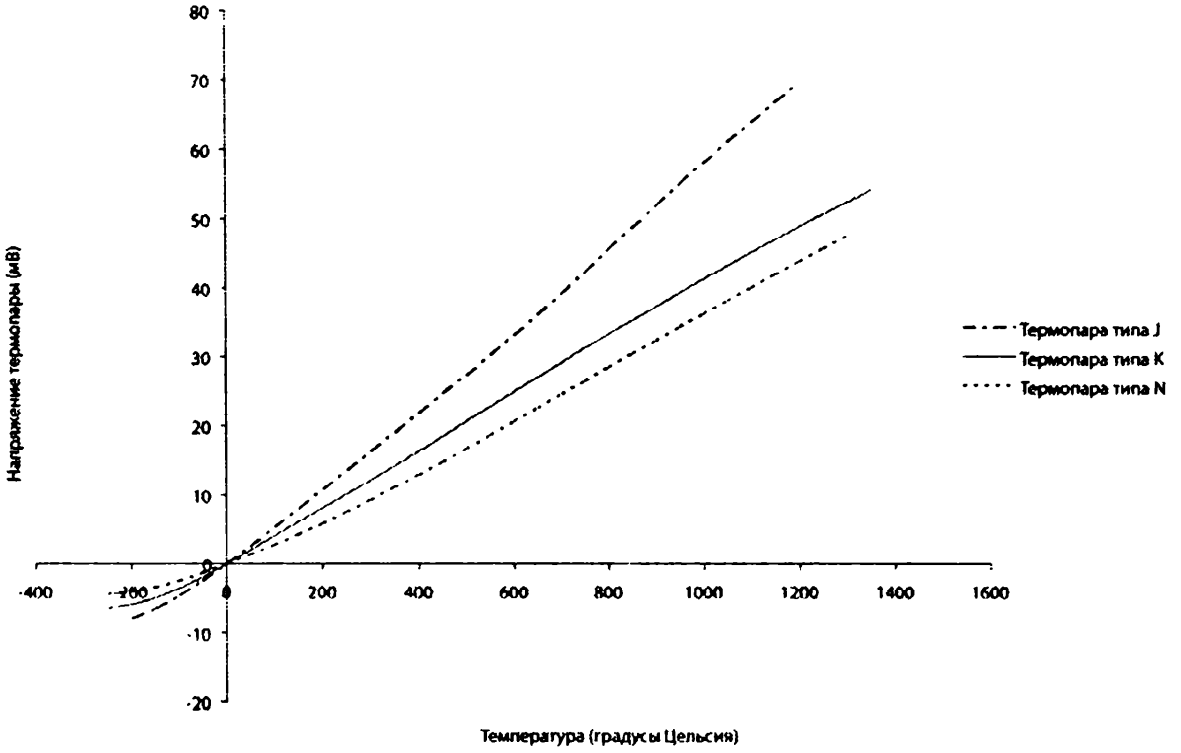


Рис. 6.1. Характеристики различных термопар

Термопреобразователи сопротивления

Еще одним очень популярным типом датчиков температуры являются *термопреобразователи сопротивления* (RDT). Обычно они состоят из тонкого платинового провода, намотанного на непроводящий цилиндр (*сердечник*). Вся конструкция покрыта непроводящим материалом. Сопротивление RDT-элемента изменяется линейно в зависимости от температуры. Пропуская через это устройство ток известной силы, мы можем получить выходное напряжение, соответствующее температуре термопреобразователя. При этом справедливо следующее уравнение:

$$R_{RTD} = R_0 + \alpha T_{RTD}$$

где R_{RTD} — сопротивление RTD-элемента в Омах, измеренное при температуре T_{RTD}; R_0 — сопротивление RTD-элемента в Омах, измеренное

при температуре 0°C; α — температурный коэффициент термопреобразования; T_{RTD} — температура RTD-элемента в °C.

Температурный коэффициент зависит от чистоты и состава платинового провода, используемого в термопреобразователе сопротивления. Наиболее распространены два стандартных RTD-элемента, один из которых имеет значение температурного коэффициента 0,00392, а другой — чуть ниже: 0,00385. Поскольку термопреобразователи с меньшим значением α первоначально широко использовались в Европе, их называют RTD с "европейской характеристикой", а устройства с более высоким значением температурного коэффициента — с "американской характеристикой".

Учитывая тот факт, что термопреобразователи сопротивления гораздо более линейны, чем термопары, возникает вопрос: "Чем предпочтительнее термопара?". На этот вопрос можно ответить двояко: с экономической точки зрения, и с точки зрения системной эффективности. Термопреобразователи обычно дороже своих термоэлектрических конкурентов, поэтому себестоимость устройства, использующего RTD-элемент, может оказаться выше себестоимости аналогичного устройства на термопаре. Еще более серьезные ограничения на использование термопреобразователей сопротивления возникают в области системной эффективности.

Мы обычно хотим измерять температуру системы, не нарушая работу самой этой системы. Поскольку для работы RTD-элемента через него необходимо пропускать ток (иначе не возникнет напряжение на устройстве), присутствует некоторый ненулевой самонагрев чувствительного элемента, обусловленный протеканием тока. Этот эффект особенно заметен в системах с малым потоком материала через датчик, из-за чего возникает дополнительное тепло, которое может вызвать тепловую погрешность измерений, переменную во времени. Тепловая погрешность действительна в том смысле, что окружающий датчик материал на самом деле становится теплее, однако это не отражает точную температуру материала или среды даже на небольшом расстоянии от датчика. Термопары, которые, по сути, являются устройствами с нулевым протеканием тока, эффекту самонагревания не подвержены.

Другое ограничение системной эффективности связано с диапазоном температур, в котором могут работать RTD-элементы. В общем случае этот диапазон гораздо уже, чем у термопар, хотя отдельные типы термопар также могут иметь ограничения для температурного диапазона. Термопреобразователи сопротивления работают с температурами от –250°C до +850°C, в то время как термопары — в диапазоне от –270°C

до +2300°С. Таким образом, для приложений, работающих с температурами выше 850°С, термопара может оказаться единственным жизнеспособным вариантом чувствительного элемента.

Терморезисторы

Подобно RTD-элементам, терморезисторы изменяют свое сопротивление в зависимости от температуры. Однако, в отличие от RTD-элементов, сопротивление которых с повышением температуры увеличивается, сопротивление терморезисторов при нагреве уменьшается. *Отрицательный температурный коэффициент* — не единственное расхождение между этими двумя устройствами, поскольку, в отличие от RTD-элементов, терморезисторы в высшей мере нелинейны. К счастью, терморезисторы можно использовать в виде согласованных пар, когда нелинейность одного из них компенсирует нелинейность другого, в результате чего на выходе формируется приемлемый линейный сигнал.

Поскольку для работы терморезисторов через них необходимо пропускать ток, они разделяют с RTD-элементами проблему самонагревания. При этом их рабочий диапазон значительно уже и обычно составляет от –40°С до +150°С. Тем не менее, внутри этого диапазона правильно сконфигурированные терморезисторы могут выдавать очень точные результаты.[40]

Кремниевые датчики

Датчики температуры часто включают в состав электронных систем, функционирующих в жестких внешних условиях. Система может использовать замеры температуры для активизации климатического контроля или же для защитного самоотключения, если вокруг станет слишком жарко или слишком холодно. Или же она может использовать данные о температуре для выполнения других задач, специфических для приложения. Какой бы ни была причина, возрастающая интеграция компонентов систем привела к созданию первых датчиков температуры на основе кремния с их последующим внедрением в микросхемы.

Обычно кремниевые датчики температуры довольно точны (на уровне RTD-элементов и терморезисторов), однако их рабочий диапазон еще уже, чем у терморезисторов (от –40°С до +125°С). Кроме того, им присущи те же проблемы с самонагреванием, что и любым другим

---

[40] Справочная информация об RTD-элементах и терморезисторах взята с Web-сайтов `http://www.omega.com/techref/measureguide.html` и `http://rdfcorp.com/anotes/pa-r/pa-r_01.shtml`.

чувствительным элементам, которые для своей работы требуют электроэнергии. Устройства на базе кремниевых датчиков обычно стоят дороже некремниевых аналогов, однако, как правило, не требуют дополнительных издержек на прокладку проводов, поскольку зачастую совмещены со схемами, которые они контролируют. Даже если такой датчик размещен на некотором расстоянии от процессора, его можно соединить с основной системой обычным медным проводом, в отличие от дорогостоящих соединений в случае с термопарой или RTD-элементом.

Инфракрасные датчики

Некоторые задачи измерения температуры не позволяют использовать датчики, которые находятся в физическом контакте с контролируемым материалом. В качестве примера можно привести очень горячие субстанции (с температурой выше 2 300°C — предела для термопар) или материалы, к которым нежелательно прикасаться (например, тонкая пленка или свежевыкрашенная поверхность). В подобных случаях одним из возможных решений является измерение инфракрасного излучения с последующим вычислением соответствующей температуры на основании данных об излучаемой объектом электромагнитной энергии.

Хотя, теоретически в измерении температуры с помощью инфракрасных датчиков нет ничего сложного, на практике могут возникнуть трудности. Основная трудность заключается в том, что различные материалы характеризуются различной *излучательной способностью*[41], т.е. они испускают инфракрасное излучение с различной эффективностью. Для того чтобы использовать инфракрасный датчик правильно, необходимо знать характеристики излучательной способности контролируемого материала, иначе измерения температуры окажутся неточными. В качестве реального примера этого явления можно привести отслеживание температуры расплавленной пластмассы в металлическом цилиндре, подающем пластмассу в форму. В этом случае инфракрасный датчик будет воспринимать, скорее, температуру не расплава, а металлического цилиндра.

Вторым обстоятельством, которое, в зависимости от обстоятельств, может быть преимуществом, а может — и недостатком, является замер температуры для определенной области, а не отдельной точки. Хотя

---

[41] *Излучательная способность* обычно обозначается как процент между 0% и 100% или же как дробь между 0 и 1. В обоих случаях меньшим значениям соответствует менее активное инфракрасное излучение. Обсуждение значений относительной излучательной способности можно найти на Web-сайте `http://www.omega.com/techref/measureguide.html`.

в какой то мере это относится ко всем датчикам температуры (в конце концов, еще никто не изобрел датчик, выполняющий измерения в области нулевого размера!), рассмотренные ранее типы датчиков обычно соотносят с точечными измерениями, т.е. они достаточно малы для того, чтобы их размерами можно было пренебречь. Но иначе обстоит дело с инфракрасными датчиками, которые исследуют некоторую область, спроецированную на объект наблюдения. По сути, они сообщают о средней температуре этой области. При использовании такого датчика очень важно, чтобы оператор полностью осознавал связь контролируемой области с объектом, иначе датчик будет выдавать неточные данные, отображающие усредненную температуру окружающей среды.

Третья потенциальная проблема инфракрасных датчиков заключается в том, что они могут воспринимать инфракрасное излучение, отраженное от других источников, а не только испускаемое объектом наблюдения. Подобно всем электромагнитным волнам, инфракрасное излучение отражается от любой поверхности, с которой входит в контакт, поэтому часть замеренной инфракрасной энергии может запросто принадлежать другим источникам. Если по сравнению с излучаемой энергией объекта отраженная энергия ничтожно мала, то ею можно пренебречь, в противном же случае она серьезно отразится на качестве измерений температуры.

Впрочем, одной из основных проблем инфракрасных датчиков является их стоимость. Нет ничего необычного в том, чтобы инфракрасный датчик стоил тысячи долларов, что на несколько порядков выше, чем стоимость любого другого из ранее рассмотренных датчиков. При такой цене, а также учитывая необходимость дополнительного обучения операторов плюс дополнительные операционные сложности (в частности, необходимо располагать подробной информацией о характеристиках инфракрасного лучеиспускания контролируемого вещества), не удивительно, что в настоящее время инфракрасные датчики находят очень ограниченное применение.

6.2. Ключевые аспекты измерения температуры

В любой системе измерения температуры существует ряд фундаментальных сфер, которые необходимо учитывать, чтобы получить осмысленные и точные результаты. В зависимости от конкретной ситуации, неточности в таких сферах могут иметь последствия от несущественных (если термостат кондиционера не вовремя выключает кондиционер, то его можно просто отрегулировать) до удручающих (попробуйте испечь кекс, когда вы думаете, что температура в духовке равна

200°C, а на самом деле в ней — 120°C[42]) и просто катастрофических (вообразите, что будет при перегреве линии по производству водорода).

В остальной части этой главы будут рассмотрены вопросы, имеющие отношение к датчикам на основе термопары, однако они вполне применимы к любым системам измерения температуры, вне зависимости от типа используемых в них датчиков. Для того чтобы гарантировать корректную работу системы, необходимо рассмотреть следующие важные факторы:

- типы доступных датчиков температуры;
- требуемый диапазон измерений;
- необходимая разрешающая способность и точность (это — не одно и то же);
- характеристики сигнала от термопары;
- источники шума в измеренном сигнале.

Диапазон измерений

В любой измерительной системе в первую очередь необходимо определить диапазон обрабатываемых значений, поскольку он зачастую определяет тип датчика. Хотя рабочий диапазон температур для термопар в общем случае составляет от –270°C до +1 760°C, отдельные типы термопар могут перекрывать только его часть, как показано в табл. 6.1 (взята из реестра термоэлектрических напряжений и коэффициентов, изданного Национальным институтом стандартов и технологии[43]).

Таблица 6.1. Рабочие диапазоны термопар

Тип	Диапазон (°C)	Тип	Диапазон (°C)
B	0 – 1 820	N	–270 – 1 300
E	–270 – 1 000	R	–50 – 1 760
J	–210 – 1 200	S	–50 – 1 760
K	–270 – 1 370	T	–270 – 400

Диапазоны измерений, указанные в табл. 6.1, — это, в действительности, лишь отправная точка оценки термопар, поскольку они определяют предельные условия, при которых может использоваться термопара того или иного типа. Фактический же рабочий диапазон термопары уменьшается в зависимости от диаметра провода термопары (чем тонь-

---

[42] Если вы, как и автор, далеки от кулинарии, то представьте себе теплую липкую массу.

[43] Эти данные можно найти на Web-сайте Национального института стандартов и технологий `http://srdata.nist.gov/its90/download/download.html`.

ше провод, тем более ограничен диапазон температур), а также допустимой температуры защитной оболочки провода (если она есть). Например, верхний температурный предел для термопар типа J в табл. 6.1 определен как 1 200°С, однако согласно спецификации серийной защищенной термопары типа J с оголенным проводом[44] диаметром 3,25 мм, компания-изготовитель оценивает верхний предел температурного диапазона всего лишь в 760°С. Если же в термопаре используется провод диаметром 0,13 мм то максимальная рабочая температура термопары понижается до 315 °С, т.е. потеря ожидаемого диапазона температур составляет почти 75 %.

Еще один фактор реальной жизни, влияющий на выбор типа термопары... Возможно, у разработчика вообще не будет альтернатив. Зачастую тип термопары заранее определен стандартом, изменить который просто невозможно. Например, в Северной Америке, в процессе литья под давлением в большинстве случаев используют термопары типа J, в то время как европейские и азиатские заказчики в той же области применения обычно используют термопары типа K. Разумный проектировщик, как правило, по возможности старается сделать так, чтобы система поддерживала более одного типа термопар. Это сохраняет время и силы как конечного пользователя, так и изготовителя системы.

Разрешающая способность измерений

Требуемый диапазон измерений не только определяет тип термопары, но и влияет на *разрешающую способность*, которую мы ожидаем от получить. Под разрешающей способностью понимают точность измерений, которая обычно выражается в градусах. Очень распространено требование обеспечить измерение температуры с точностью до половины градуса по Цельсию (или, возможно, — до одного градуса по Фаренгейту), однако мы сможем его выполнить только в том случае, если используемый АЦП сможет оцифровать аналоговый сигнал от датчика с достаточным разрешением. Для этого должны быть соблюдены следующие два условия:

- необходимо обеспечить преобразование диапазона выходного аналогового сигнала от термопары в диапазон аналогового напряжения, которое может быть оцифровано АЦП;
- АЦП должен обеспечить оцифровку преобразованного диапазона сигнала с требуемой разрешающей способностью.

---

[44] Данные взяты из таблиц компании Omega Engineering, Inc. на Web-сайте `http://www.omega.com/temperature/z/tcref.html`.

Как мы ни расхваливали преимущества программной обработки сигнала от датчика, первое условие может быть удовлетворено только с помощью электроники. Для этого обычно между выходом датчика и входом АЦП dsPIC размещают схему усиления и сдвига уровней сигнала. Если не выполнить правильное сопоставление выходного сигнала от датчика с диапазоном напряжений, совместимым с АЦП, то оцифрованный сигнал может быть отсечен[45] и даже вывести из строя ЦКС dsPIC, если значение входного сигнала окажется за пределами допустимого входного абсолютного напряжения, указанного в спецификации на микросхему. Добавление каскада усиления также позволяет нам буферизировать выходной сигнал датчика, который обычно обладает очень ограниченной способностью к управлению, от относительно низкоомного входа АЦП ЦСК dsPIC. Благодаря добавлению буфера, мы предохраняем АЦП dsPIC от привнесения нежелательных искажений ввиду чрезмерной нагрузки на выходной сигнал датчика.[46]

Как же, в таком случае, разработчику выполнить это преобразование? Процедура проста и состоит всего лишь из трех простейших расчетов. Тем не менее, уже перед этими вычислениями мы должны знать диапазон напряжения выходного сигнала датчика (по крайней мере, для измеряемых значений интересующего параметра). Кроме того, необходимо знать допустимый диапазон входных напряжений АЦП. Вооружившись этими знаниями, мы затем должны выполнить следующие три расчета.

1. Выразить выходной сигнал датчика и входное напряжение АЦП в терминах их размаха и смещения:

---

[45] *Отсечка* происходит тогда, когда теоретическое значение выходного аналогового сигнала превышает фактический уровень выходного сигнала в результате неспособности схемы обеспечить требуемое выходное напряжение. Примером этого может быть схема усилителя с шинами "земли" и питания +5 В при коэффициенте усиления 10. Любой входной сигнал между 0 В и 0,5 В будет усилен правильно, однако для входных сигналов, превышающих 0,5 В, на выходе будет получено напряжение только 5 В, поскольку схема не может выдать максимальное напряжение, превышающее напряжение питания.

[46] Легко определить, что схема АЦП чрезмерно нагружает выход датчика, можно по тому, что выходной сигнал датчика нормально выглядит на осциллографе, однако значительно изменяется после подачи на вход АЦП. Такие изменения могут проявляться в виде уменьшения размаха сигнала, когда изменяется воздействие на датчик, или же приближения уровня сигнала от датчика к одному из двух напряжений питания АЦП.

$SPAN_{SENSOR} = VS_{MAX} - VS_{MIN}$

$OFFSET_{SENSOR} = VS_{MIN}$

$SPAN_{ADC} = VA_{MAX} - VA_{MIN}$

$OFFSET_{ADC} = VA_{MIN}$

где $SPAN_{SENSOR}$ — размах выходного напряжения датчика; VS_{MAX} — максимальное выходное напряжение датчика; VS_{MIN} — минимальное выходное напряжение датчика; $OFFSET_{SENSOR}$ — смещение выходного напряжения датчика; $SPAN_{ADC}$ — размах входного напряжения АЦП; VA_{MAX} — максимальное входное напряжение АЦП; VA_{MIN} — минимальное входное напряжение АЦП; $OFFSET_{ADC}$ — смещение входного напряжения АЦП.

2. Рассчитать коэффициент усиления, необходимый для сопоставления размаха выходного напряжения датчика с размахом входного напряжения АЦП:

   $GAIN = SPAN_{ADC} / SPAN_{SENSOR}$

   где GAIN — требуемый коэффициент усиления. Если этот коэффициент больше 1, то выходной сигнал датчика усиливается в традиционном смысле этого слова (т.е. становится больше); если же коэффициент усиления меньше 1, то выходной сигнал датчика ослабляется.

3. Рассчитать смещение уровня, требуемое для сопоставления смещения выходного напряжения датчика с требуемым минимальным входным напряжением АЦП:

   $SHIFT = OFFSET_{ADC} - OFFSET_{SENSOR}$

   где SHIFT — напряжение, которое должно быть добавлено к выходному напряжению датчика *после* усиления.

Удовлетворив первое из наших двух условий (сопоставление диапазона выходных напряжений датчика с диапазоном входных напряжений АЦП), мы должны предпринять меры для выполнения второго условия (проверка способности АЦП оцифровывать преобразованный диапазон напряжений с достаточной точностью). Это — тоже достаточно простой процесс, состоящий из трех этапов.

1. Рассчитать диапазон измеряемых значений параметра, который соответствует диапазону выходных напряжений датчика, полученного в ходе предыдущих вычислений:

$PARM_{RANGE} = PARM_{MAX} - PARM_{MIN}$

где $PARM_{RANGE}$ — диапазон значений замеряемого параметра, соответствующий диапазону выходных напряжений датчика (выражен в единицах измерения, соответствующих природе замеряемого параметра, а не в терминах самого напряжения); $PARM_{MAX}$ — значение параметра, которое соответствует максимальному выходному напряжению датчика (обратите внимание на то, что это — *не* максимальное выходное напряжение, а значение параметра, соответствующее этому напряжению); $PARM_{MIN}$ — значение параметра, которое соответствует минимальному выходному напряжению датчика (как и в случае с $PARM_{MAX}$, это — значение параметра, а не значение выходного напряжения датчика).

2. Рассчитать разрешающую способность АЦП в отсчетах, т.е. определить общее число уровней, на которые может быть разбит диапазон входных напряжений АЦП при заданной разрядности АЦП:

   $RES_{ADC} = 2^{NumBits}$

   где: RES_{ADC} — разрешение АЦП, выраженное в числе отсчетов (или уровней), на которое АЦП может разбить входной диапазон напряжений; NumBits — число разрядов, использованных АЦП для оцифровки (10 или 12, в зависимости от типа ЦКС dsPIC).

3. Рассчитать разрешающую способность в терминах измеряемого параметра, выраженное в единицы отсчета значений параметра на единицу (или уровень) отсчета АЦП:

   $PARM_{RES} = PARM_{RANCE} / RES_{ADC}$

   где: $PARM_{RES}$ — разрешение параметра, выраженное в виде количества единиц параметра на единицу отсчета (или уровень) АЦП.

Покажем все это на простом примере. Предположим, мы хотим использовать термопару J-типа для измерения температуры в диапазоне от 0–750°C. При этом мы выбрали термопару, конструкция которой рассчитана на весь требуемый диапазон температур, поэтому мы можем сосредоточиться только на обработке сигналов. Как видно из таблицы термоэлектрических напряжений ITS-90, термопара типа J выдает напряжение от 0,0 мВ при 0 °C до 42,281 мВ при 750 °C. Для простоты последующего масштабирования, примем, что используется диапазон входного напряжения 0–50 мВ (50 мВ соответствует приблизительно 870°C). Примем также, что мы используем АЦП dsPIC со входным диапазоном напряжений 5 В. Значение коэффициента усиления и сдвиг

уровня, требуемые для нашего усилительного каскада, определяем следующим образом.

1. Вычисляем размах и смещение выходного напряжения датчика, а также входное напряжение АЦП:

   $SPAN_{SENSOR} = 0{,}05\ В - 0\ В = 0{,}05\ В$

   $OFFSET_{SENSOR} = 0\ В$

   $SPAN_{ADC} = 5\ В - 0\ В = 5\ В$

   $OFFSET_{ADC} = 0\ В$

2. Вычислим коэффициент усиления для сопоставления выходного напряжения датчика с размахом входного напряжения АЦП:

   $GAIN = 5\ В / 0{,}05\ В = 100.$

3. Вычисляем смещение уровня, необходимое для сопоставления минимального выходного напряжения датчика с соответствующим минимальным входным напряжением АЦП:

   $SHIFT = 0\ В - 0\ В = 0\ В$

Поскольку нижний предел как для диапазона входных аналоговых напряжений, так и для диапазона выходных аналоговых напряжений составляет 0 В, то в аналоговом усилительном каскаде не требуется никакого смещения напряжения. Однако обычно усилительный каскад требует наличия компонента смещения уровня, а также сопоставления нижнего предела входного напряжения с требуемым нижним пределом диапазона выходных напряжений.

Сопоставив диапазон входных напряжений, мы готовы проверить соответствие второму условию. Если наш ЦКС dsPIC использует 12-разрядный АЦП, то каждый разряд выхода АЦП будет представлять 1/4096 часть диапазона входных напряжений АЦП ($1/2^{12} = 1/4096$). В нашем примере это означает, что мы можем оцифровать усиленный сигнал от датчика температуры с разрешающей способностью

$$T_{RES} = T_{RANGE} / RES_{ADC}$$

$$T_{RES} = (870°C - 0\ °C) / (4096)$$

$$T_{RES} = 0{,}2124\ °C$$

где T_{RES} — температурная разрешающая способность системы в °C; T_{RANGE} — температурный входной диапазон системы в °C; RES_{ADC} — разрешающая способность АЦП в отсчетах.

Несложные вычисления показывают, что в случае использования ЦКС dsPIC с 10-разрядным АЦП (разрешающая способность $2^{10} = 1024$ отсчетов) температурная разрешающая способность для этого примера снизилась бы до

$$T_{RES} = (870°C - 0°C) / (1024)$$

$$T_{RES} = 0{,}8496\ °C$$

Если бы нам понадобилось поддерживать температурную разрешающую способность 1°F (0,56 °C), то мы смогли бы сделать это с помощью 12-разрядного АЦП, а 10-разрядный АЦП не удовлетворял бы системным требованиям.

Точность измерений

Пользователи (а иногда — и проектировщики) часто путают разрешающую способность системы с ее *точностью*. В то время как разрешение соответствует степени детализации для вычислений, производимых с измерениями, точность определяет фактическую корректность этих измерений. Обе концепции важны, однако одно не обязательно подразумевает другое. Например, наша система теоретически может обеспечить разрешающую способность 0,5°C, но это не принесет особой практической пользой, если из-за проблем в системе измерения не будут выполняются уже при 10°C.

Где же могут возникать такие грубые ошибки? В первую очередь следует обратить внимание на сам чувствительный элемент, поскольку неточный датчик может вносит погрешность случайного характера (предпочтительнее, чтобы погрешность была систематической, поскольку ее можно компенсировать). Конечно же, различные типы термопар имеют различную точность в своем рабочем диапазоне. При этом наиболее точными являются термопары типа J (±0,1°C). Термопары типов E, R, и T обеспечивают точность до ±0,5°C; термопары типа K — до ±0,7°C; а термопары типа S — ±1,0°C.

Проблемы также могут возникать в цепи прохождения сигнала, например: отсутствие или недостаточная компенсация холодного спая (обсуждается ниже в подразделе "Компенсация холодного спая"); отсутствие надлежащего ослабления синфазных помех или внесение в схему несинфазных помех. Обычно мы можем компенсировать погрешности, вносимые самим датчиком или другими элементами схемы (при условии, что эти проблемы не носят временно- или температурно-зависимого характера), выполнив надлежащую калибровку и линеаризацию. Итак, разрешающая способность не эквивалентна точности, и потому

для получения корректно работающей, надежной системы необходимо обеспечить оба этих параметра.

Специфика

Теперь мы переходим от общего к частному: от аспектов проектирования, относящиеся ко всем сенсорным системам, к тем, которые являются специфичными именно для систем с термопарой в качестве чувствительного элемента.

Характеристики сигнала

Отправной точкой проектирования любой сенсорной системы является определение характеристик сигнала, получаемого от датчика в рабочем диапазоне. К таким характеристикам относят уровни напряжений, нагрузочную способность выхода датчика, а также ожидаемый частотный спектр сигнала.

Уровни сигналов

Даже беглого обзора таблицы NIST достаточно, чтобы увидеть, что термопары производят очень низкое выходное напряжение: порядка нескольких милливольт. Использование термоэлектрического сигнала усложняет тот факт, что это, по сути, — сигнал нулевого тока, а значит его нагрузка должна иметь очень высокое полное сопротивление. К сожалению, входной импеданс АЦП dsPIC не соответствует этому критерию, поскольку составляет около 20 кОм. Кроме того, в зависимости от диапазона измеряемой температуры, выходное напряжение термопары может быть отрицательным, что нарушает требования ко входным сигналам АЦП dsPIC. Даже если термопара будет формировать положительное напряжение для всех температур рабочего диапазона, то в ситуации, когда ее выводы будут реверсированы (что на практике случается гораздо чаще, чем многие предполагают), входное напряжение схемы оцифровки сигнала опять-таки может стать отрицательным. Совершенно очевидно, что в системе с термопарой между выходом датчика и входом АЦП необходимо разместить высокоомный усилитель для буферизации, усиления и смещения уровня сигнала.

Правильно выбрав коэффициент усиления и смещение усилителя, можно разработать систему, которая будет работать правильно как в случае корректного подключения термопары, так и в том случае, когда ее выводы реверсированы. На первый взгляд эта возможность может показаться несущественной. В конце концов, разве пользователь не может просто поменять местами выводы термопары и устранить пробле-

му? Иногда такое вполне допустимо, но бывают случаи, когда это невозможно. В одном из проектов, связанном с литьевым производством, в котором принимал участие автор книги, стоимость замены и модернизации формы (в которой были установлены термопары) составила 50 тыс. долларов — сумма, которая *ежедневно* увеличивалась на 20 тыс. из-за простоя оборудования. Полная такая система обычно использует более сотни датчиков различных типов, и законы Мэрфи[47] предсказывают, что выводы хотя бы одной из термопар обязательно будут перепутаны (что и происходит в действительности). В такой обстановке способность работать при минимальных проблемах с монтажом позволяет сохранить заказчикам массу денег и нервов.

Частотный спектр сигнала

Если мы предполагаем фильтровать выходные сигналы датчика для снижения электрической зашумленности, то прежде должны определить их частотный спектр, т.е. часто*т*ы, которые можем безопасно подавлять без чрезмерного ухудшения самого сигнала датчика. Для того чтобы определить спектральное содержимое температурного сигнала, необходимо кое-что знать о тепловых характеристиках контролируемой системы, а именно: как быстро ее температура может изменяться во времени. В некоторых системах температура может изменяться очень быстро, в то время как в других она может колебаться относительно медленно. Хотя у проектировщика обычно заранее нет четкой спецификации температурного поведения системы, он, как правило, достаточно четко представляет, какую максимальную скорость изменения температуры следует ожидать. Например, в приложении, связанном с нагревом, на основании максимальной мощности нагревательного элемента и характеристик теплового рассеяния нагреваемой среды можно довольно точно предположить максимальную скорость изменения температуры. Включая духовку, чтобы разогреть пиццу, мы не ожидаем, что она мгновенно нагреется до 230°С. Скорее всего, на это уйдет около трех минут. Используя этот, в общем-то, грубый подход, мы можем оценить, что максимальная температурная скорость[48] системы составляет около

---

[47] Общая суть законов Мэрфи заключается в следующем: если что либо может сломаться, то оно обязательно сломается; среди двух потенциально опасных элементов выйдет из строя именно тот, который более критичен.

[48] *Температурная скорость* определяет изменение температуры с течением времени по аналогии с обычной скоростью, которой соответствует изменение расстояния с течением времени.

$$\Delta T_{MAX} = 230°C / 180 \text{ секунд} = 1{,}28°C/c$$

Продолжим рассмотрение примера с 12-разрядным АЦП и термопарой типа J из подраздела "Разрешающая способность измерений". В этом случае максимальная скорость изменения температуры соответствует отсчетам АЦП:

$$\Delta COUNT_{MAX} = \Delta T_{MAX} / T_{RES}$$

$$\Delta COUNT_{MAX} = (1{,}28°C/c) / (0{,}2124 \text{ C /отсчет}) \approx 6 \text{ отсчетов/с.}$$

Предполагая, что мы хотим обеспечить отслеживание изменений температуры с точностью до одного отсчета АЦП, выборка должна происходить по крайней мере шесть раз в секунду. Однако, применив критерий Найквиста (рассматривается в главе 2, "Интуитивная цифровая обработка сигналов"), теоретическая минимальная частота дискретизации должна превышать полученное нами значение хотя бы в два раза (т.е. составлять 12 выборок/с), а практическая частота дискретизации — примерно в пять раз (т.е. составлять 30 выборок/с).

Этот подход имеет ряд недостатков, и мы начнем с наиболее очевидного из них. Вероятны периоды времени в течении цикла нагрева, когда характеристика изменения температуры будет отличаться от ожидаемой линейной. Кроме того, предполагается, что к сигналу датчика применены аналоговые фильтры для подавления широкополосного шума. Если фильтрация отсутствует или полоса пропускания фильтра шире, чем требуется частотой дискретизации, то необходимо соответствующим образом повысить частоту дискретизации. В добавок ко всему, термопары сами по себе характеризуются определенной скоростью реакции. Это означает, что для точного отражения колебаний температуры в виде изменения термоэлектрического выходного напряжения термопаре потребуется некоторое время. Тем не менее, при отсутствии какой-либо дополнительной информации описанную выше методику можно считать хорошей отправной точкой.

Компенсация холодного спая

Гораздо более сложный аспект использования термопар — так называемая *компенсация холодного спая*. Ранее уже был рассмотрен термоэлектрический эффект, создающий в области контакта двух разнородных металлов напряжение, зависящее от температуры, — именно то свойство, которое делает термопары полезными. К сожалению, область действия термоэлектрического эффекта не ограничена контактом двух проводов в термопаре (так называемый "горячий спай"), но распростра-

няется и на соединение выводов термопары с медными проводниками печатной платы (так называемый "холодный спай"). Возникшее в результате нежелательное напряжение необходимо устранить, что реализуют с помощью методики, известной как *компенсация холодного спая*.

Ее суть заключается в измерении температуры на холодном спае (т.е. в месте соединения выводов термопары с печатной платой) и ее сложении с температурой, рассчитанной для горячего спая (т.е. для термопары датчика). Таким образом, нежелательное напряжение, обусловленное термоэлектрическим эффектом, компенсируется.[49] Эквивалентный подход заключается в том, чтобы просто добавить соответствующее напряжение на холодном спае к напряжению на горячем спае.

Для того чтобы сделать это правильно, в конструкции должны быть соблюдены следующие условия:

- места соединения проводников в холодном спае для обоих выводов термопары имеют одинаковую температуру (так называемый *изотермический барьер*);
- устройство, используемое для измерения температуры изотермического барьера и формирования соответствующего напряжения, размещено максимально близко к месту соединения.

Невыполнение этих двух требований приведет к внесению в измерения неисправимой ошибки, которая, к тому же, будет изменяться в зависимости от температуры.

Линеаризация

После компенсации холодного спая наиболее сложной задачей при обработке термоэлектрических сигналов является сопоставление нелинейного термоэлектрического напряжения с линейной шкалой температур. Для этого обычно оценивают полиномиальные выражения высокого порядка с использованием рекурсивной методики и коэффициентов, устанавливаемых в зависимости от типа термопары. Однако эта операция требует интенсивных вычислений, и потому в общем случае она мало подходит для реализации встроенными микропроцессорами. Хотя эффективность математических расчетов в ЦКС dsPIC весьма высока, обычно применяется другая методика: *кусочно-линейная линеаризация*, которая дает превосходные результаты и менее сложна в реализации.

---

[49] Статья "Circuit Provides Cold Junction Compensation" ("Схема обеспечения компенсации холодного спая"), Марка Мэддокса (Mark Maddox) и Джона Уинна (John Wynne). Журнал "EDN", ноябрь 2002 г., `http://www.edn.com/article/CA260064.html`.

Основная идея кусочно-линейной линеаризации заключается в разбиении нелинейной кривой на ряд линейных сегментов (рис. 6.2).



***Рис. 6.2.** Пример кусочно-линейной линеаризации кривой*

Внутри каждого сегмента, мы аппроксимируем значения кривой значениями отрезка между начальной и конечной точкой сегмента. Для того чтобы рассчитать оцениваемое значение, мы просто определяем соответствующий сегмент и вычисляем значение на основании угла наклона прямой и точки ее пересечения с осью Y. Таким образом, задача вычисления полинома высокого порядка сводится к операциям поиска и линейных вычислений.

Эффективность этой методики определяют два фактора:

- число сегментов, используемых для аппроксимации кривой;
- характер аппроксимируемой кривой.

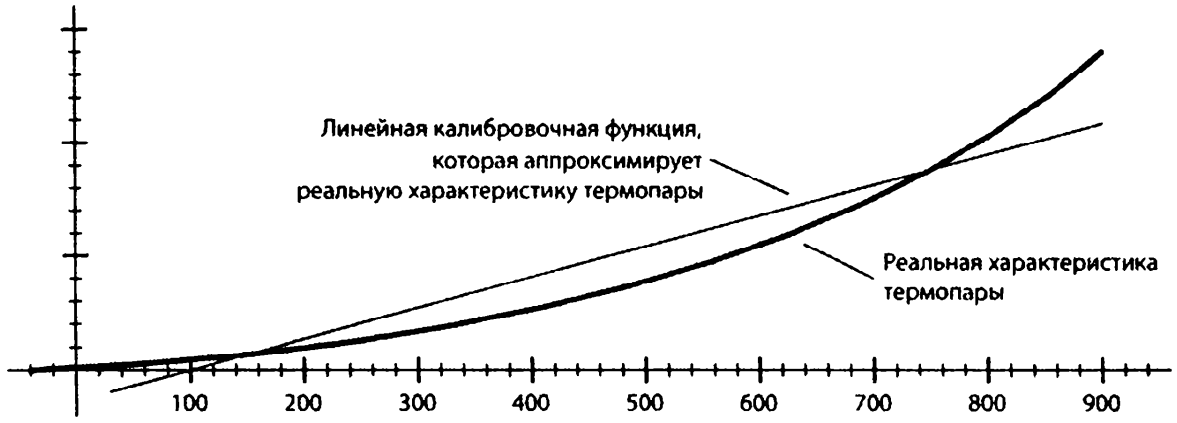
Очевидно, что чем больше задействовано сегментов, тем точнее будет аппроксимация, однако это подразумевает и больше времени, необходимого для определения сегментов. Таким образом, перед проектировщиком возникает еще одна задача: найти разумный баланс между точностью и быстродействием.

Калибровка

Линеаризация — важный этап преобразования температурного сигнала напряжения в значение, которое может использоваться микропро-

граммами dsPIC. Однако, он — не единственный, требуемый для формирования приемлемого цифрового значения. Линеаризованный сигнал должен также быть *откалиброван* по набору известных температурных замеров, чтобы гарантировать точность вычисленного значения сигнала. Наиболее популярной для термопар является калибровка *методом двух точек*, когда берутся два замера для различных температур (обычно в точках, близких к одной из границ рабочего диапазона), после чего эти точки используются для построения линейной калибровочной кривой.

Предположим, нам необходимо измерять температуры в диапазоне от 100°F (примерно 38°C) и 900°F (примерно 482°C). В таком случае в качестве калибровочных точек можно было бы выбрать температуры 150°F (примерно 66°C) и 750°F (примерно 399°C), поскольку они близки к границам рабочего диапазона и гарантируют, что полученная прямая не будет расположена слишком далеко от фактической характеристики термопары (рис. 6.3).



Обратите внимание на то, что изгиб реальной характеристики термопары преувеличен с тем, чтобы лучше проиллюстрировать отклонение калибровочной прямой от фактической кривой. При показанном выборе точек калибровки погрешность между ними меньше, однако она возрастает за пределами ограниченного ими участка.

***Рис. 6.3.** Калибровочная кривая, использующая точки 150°F и 750°F*

Хотя в качестве конечных точек можно было бы использовать и температуры 100°F и 900°F, это дало бы прямую со значительной максимальной погрешностью (рис. 6.4).

Для выполнения калибровки необходимо измерить линеаризованное напряжение в нижней точке калибровки, затем проделать измерение в верхней точке и вычислить наклон калибровочной прямой:

$$G_{CAL} = (T_{UCP} - T_{LCP}) / (V_{UCP} - V_{LCP})$$

$$G_{CAL} = (750°F - 150°F) / (V_{UCP} - V_{LCP})$$

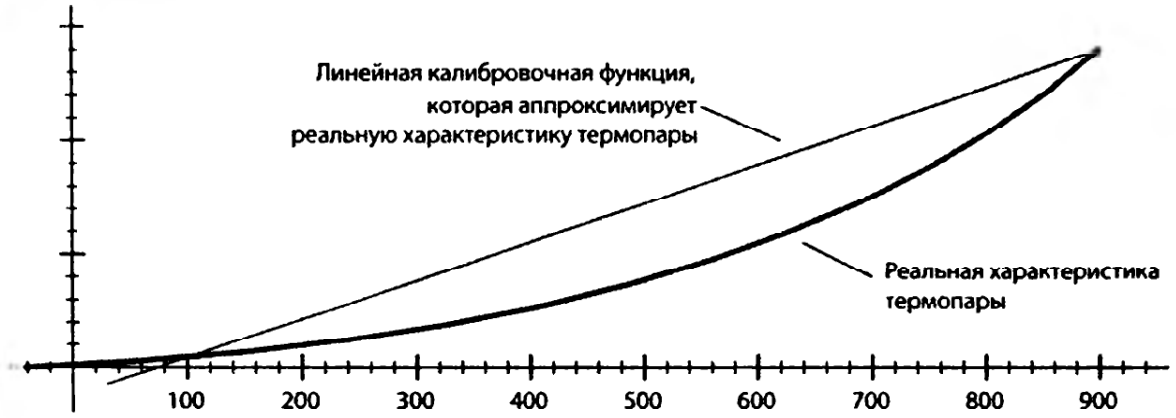
$$G_{CAL} = 600°F / (V_{UCP} - V_{LCP})$$

В то время как смещение

$$OFF_{CAL} = T_{LCP}$$

В этих уравнениях:

- G_{CAL} — наклон откалиброванной эталонной прямой;
- OFF_{CAL} — смещение откалиброванной эталонной прямой;
- T_{UCP} — температура верхней точки калибровки (750°F);
- T_{LCP} — температура нижней точки калибровки (150°F);
- V_{UCP} — значение напряжения в верхней точке калибровки;
- V_{LCP} — значение напряжения в нижней точке калибровки.



Обратите внимание на то, что изгиб реальной характеристики термопары преувеличен с тем, чтобы лучше проиллюстрировать отклонение калибровочной прямой от фактической кривой. При показанном выборе точек калибровки погрешность между ними значительно больше, однако она уменьшается у верхнего предела шкалы измерений (около 900°F).

***Рис. 6.4.** Калибровочная кривая, использующая точки 100°F и 900°F*

На практике микропрограмма датчика определит значение напряжения, линеаризует его, а затем — преобразует в соответствующее значение температуры по уравнению:

$$T_{CAL} = G_{CAL} \times (V_{READING} - V_{LCP}) + OFF_{CAL}$$

где T_{CAL} — откалиброванное значение температуры; G_{CAL} — наклон откалиброванной эталонной характеристики; OFF_{CAL} — смещение откалиброванной эталонной характеристики; $V_{READING}$ — линеаризованное значение напряжения; V_{LCP} — значение напряжения в нижней точке калибровки.

Самой собой напрашивает вопрос: "Если мы пытаемся откалибровать датчик температуры, то как мы можем быть "уверены" в значениях температуры в двух точках калибровки?". Дело в том, что мы должны использовать уже откалиброванный источник показаний, в роли которого обычно выступает калибратор, выдающий точное напряжение, соот-

ветствующее определенной температуре. Эти устройства представлены на рынке различными поставщиками и относительно недороги, хотя более точные и простые в обращении модели могут стоить от нескольких сотен до нескольких тысяч долларов.

Источники электрического шума

Источники электрического шума, неблагоприятно воздействующие на термоэлектрическую систему, — это объективная реальность. Учитывая тот факт, что термопарам присущи низкие уровни сигнала, шум может привести к катастрофическим последствиям. Впрочем, хорошая новость заключается в том, что для эффективной фильтрации значительной части нежелательного сигнала мы можем использовать как шумоподавляющие характеристики дифференциального сигнала термопары, так и частотные спектры температурного сигнала и шума.

Питание от сети переменного тока

Распространенным источником электрического шума является излучение от сети переменного тока — особенно в промышленных и других средах, в которых вблизи от термопар проходят кабели с большими токами. К счастью, спектральный состав шума от сети электропередачи имеет крайне узкую полосу частот, сосредоточенную в районе 50 или 60 Гц (в зависимости от географического месторасположения), с гармониками, кратными этим частотам. Такие четкие узкополосные шумовые сигналы обрабатывать значительно проще, чем широкополосный шум (конечно, если только полоса частот шума не захватывает слишком большую часть полосы самого сигнала от датчика).

Поскольку сигнал от термопары очень слаб и практически не возбуждает токов, он крайне восприимчив к электрическому шуму от сети электропитания. К счастью, дифференциальная природа сигналов от термопар помогает отсеять синфазный шум, а частотный спектр помех от сети электропитания зачастую лежит вне полосы частот температурного сигнала. Даже в тех случаях, когда частоты сетевых помех находятся в полосе частот температурных сигналов, их узкополосный характер обычно позволяет эффективно удалить их с помощью заграждающего фильтра.

На рис. 6.5 показан пример внеполосного частотного спектра сигнала сетевых помех вместе со спектром температурного сигнала. Применив фильтр нижних частот, показанный на рис. 6.6, можно легко удалить лишний спектральный состав, что даст отфильтрованный сигнал, показанный на рис. 6.7.

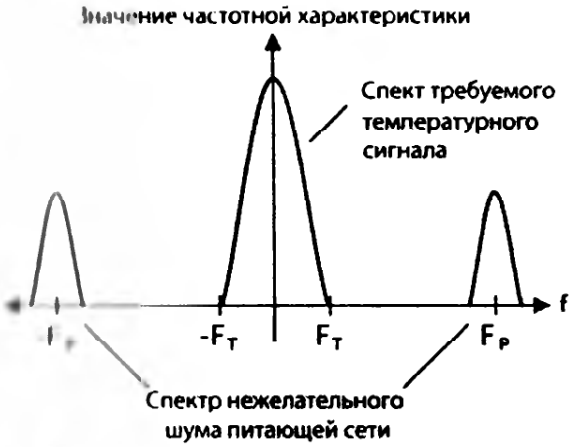


Рис. 6.5. Температурный сигнал от термопары вместе со внеполосным сетевым шумом

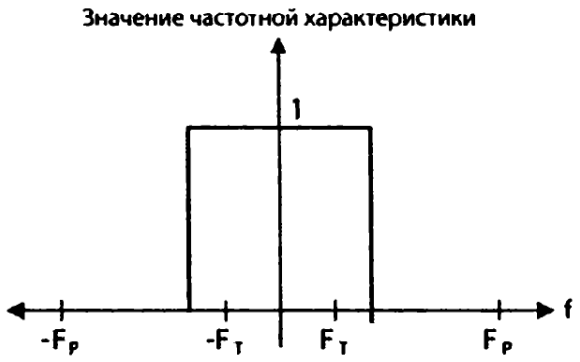


Рис. 6.6. Фильтр нижних частот для устранения внеполосного сетевого шума

Более сложный случай внутриполосных сигналов сетевого шума показан на рис. 6.8. Здесь спектр шума лежит внутри частотного спектра температурных сигналов. В таком случае для устранения нежелательных частот необходимо применить узкополосный режекторный фильтр высшего порядка (рис. 6.9). В результате мы получим улучшенный, хотя и не идеальный отфильтрованный температурный сигнал (рис. 6.10). Впрочем, этот сигнал все равно — более точный, чем его неотфильтрованная версия.

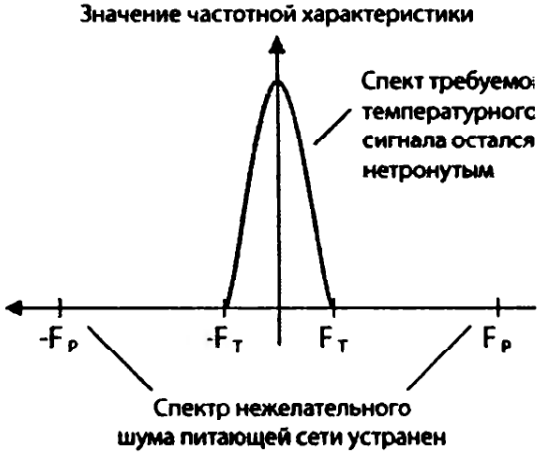


Рис. 6.7. Отфильтрованный сигнал от датчика

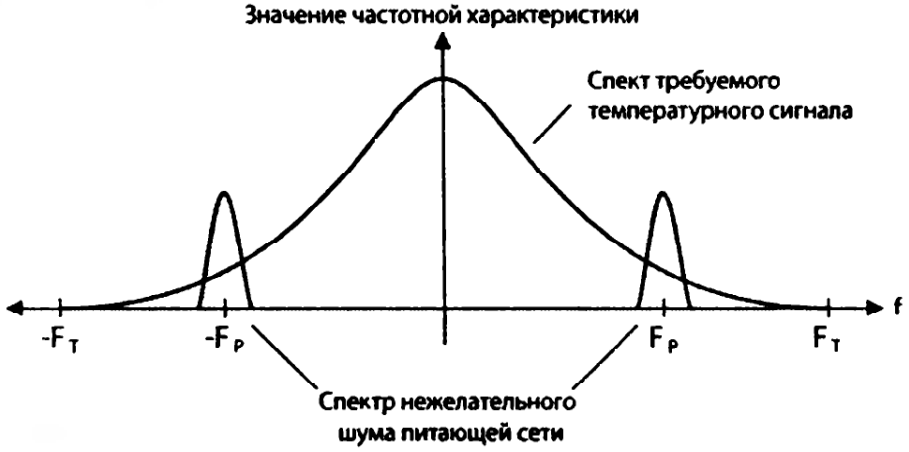
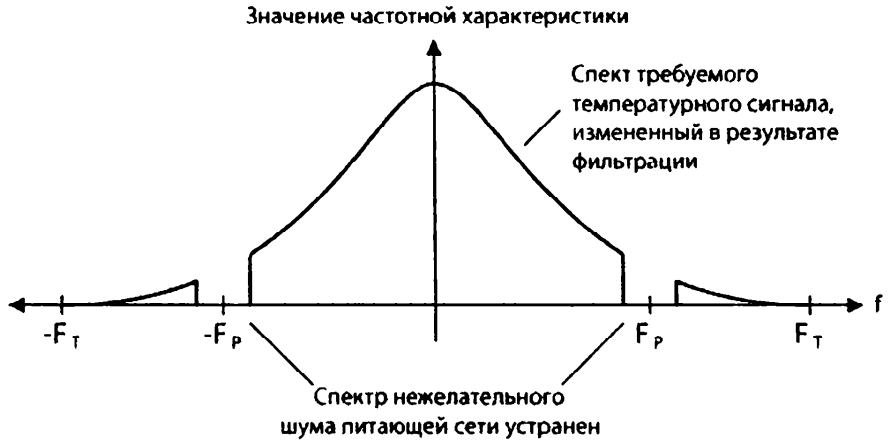


Рис. 6.8. Температурный сигнал термопары вместе с внутриполосным сетевым шумом



Рис. 6.9. Узкополосный режекторный фильтр, используемый для устранения внутриполосного сетевого шума



Хотя спектр полезного температурного сигнала затронут фильтрацией, большая часть исходного сигнала осталась без изменений, что позволяет получить приемлемые результаты.

Рис. 6.10. Результирующий отфильтрованный сигнал от датчика

Наконец, рассмотрим случай, когда спектр сетевого шума значительно накладывается на сигнал температуры (рис. 6.11).

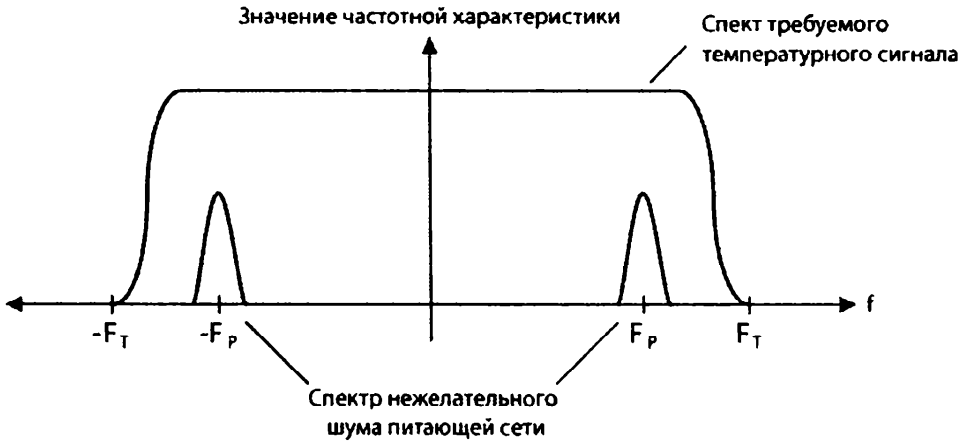


Рис. 6.11. Температурный сигнал, смешанный с сетевым шумом

В этом случае приемлемого подхода к фильтрации для компенсации сетевого шума не существует, и проектировщику приходится довольствоваться разве что попыткой установить соотношение сетевого шума на термопаре с самим сигналом сети, а затем вычесть результат из сигнала от датчика. Хотя это технически возможно, гораздо лучшим подходом для данной ситуации будет экранирование термопары и кабелей электропитания, а также прокладка кабелей, позволяющая минимизировать создаваемое совместное излучение. На практике крайне маловероятно, что системы на базе термопары вообще когда либо столкнутся с такими экстремальными условиями, поскольку существует совсем немного материалов (включая те, которые используются для создания термопар), которые могут реагировать термически на частоту 50 или 60 Гц.

Сбойные ситуации

Существуют две серьезные сбойные ситуации, которые можно довольно легко обнаружить путем разумного использования присоединенных к термопаре высокоомных резисторов смещения и контроля характеристик измеряемого напряжения. Пренебрежение сбойными ситуациями может привести к тому, что датчик будет выполнять чрезвычайно неточные замеры, что в свою очередь в ряде случаев может привести к катастрофическому отказу системы.

Разомкнутая термопара

Первая сбойная ситуация, которую мы можем контролировать, — это наличие разомкнутой термопары, т.е. термопары, в которой нарушен спай двух проводников. Это довольно часто случается в системах, подверженных значительным механическим или тепловым нагрузкам (особенно если эти нагрузки носят повторяющийся характер). Обнаружить такую ситуацию довольно легко при условии подсоединения ко входным выводам термопары двух согласованных высокоомных резисторов (по одному на каждый вывод) (рис. 6.12).

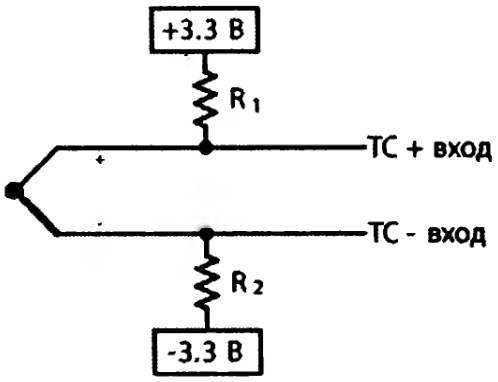


***Рис. 6.12.** Контрольные резисторы, подключенные ко входам термопары*

В случае нарушения спая термопары на ее входах проявятся крайние значения входного напряжения — условие, которое легко обнаружить с помощью ЦКС dsPIC.

Номинал резистора смещения следует выбирать так, чтобы не ослабить сам сигнал термопары в нормальных условиях. Обычно вполне приемлем номинал в районе 100 кОм, однако очень важно максимально согласовать два резистора, чтобы сохранить дифференциальную природу сигнала.

Реверсированная термопара

Поскольку выводы термопары поляризованы, существует вероятность их реверсирования. Хотя это и не повредит входной схеме, поскольку сигнал имеет очень низкий уровень, эта сбойная ситуация может привести к считыванию ошибочных результатов. Если выводы термопары реверсированы, то с увеличением температуры уровень напряжения будет уменьшаться. Однако обычно неизвестно, растет или понижается температура фактически, поэтому единственный способ удостовериться в том, что термопара реверсирована — это обнаружить напряжение, которое ниже ожидаемого нижнего предела диапазона сигнала. Например, термопары типа J дают отрицательное напряжение, когда подвержены воздействию температур ниже 0°C, поэтому при рабочем диапазоне температур 0–500°C (32–932°F) любое отрицательное напряжение укажет на то, что термопара реверсирована.

Поскольку нам необходимо обеспечить различение разомкнутой термопары, а также ситуации, когда ее выходной сигнал упал ниже некоторого минимального значения, важно обеспечить небольшой запас на обоих концах диапазона оцифровки. Таким образом мы сможем гарантировать, что измеряем весь интересующий нас диапазон температур, сохраняя способность правильно идентифицировать сбойные состояния термопары.

6.3. Разработка приложения

Итак, получив всю базовую информацию, мы готовы к разработке реального приложения для интеллектуального датчика температуры (по крайней мере, настолько же реальное, насколько можно применить демонстрационную плату в качестве аппаратной платформы). Хотя рассмотренное ниже приложение носит обобщенный характер, его можно легко расширить для решения более сложных задач (например, не только измерения, но и регулирования температуры).

Как и в случае любого другого приложения, на первом этапе разработки необходимо определить требуемые функциональные возможности системы, что мы и сделаем в подразделе “Спецификация системы”. Следующим этапом будет проектирование схемы формирования сигна-

на для конкретного тип датчиков. Далее необходимо определить требования цифровой фильтрации и разработать алгоритмы анализа данных. Этим этапам соответствуют подразделы "Формирование сигналов от датчика", "Анализ цифровой фильтрации" и "Алгоритмы анализа данных". Наконец, датчик с помощью предопределенного протокола, которому посвящен подраздел "Протокол обмена данными", должен передавать результаты измерений остальным компонентам системы.

Спецификация системы

Перед нами стоит задача разработать обобщенную аппаратно-программную платформу системы измерения температуры, поэтому спецификация не будет слишком детализированной. Зачем погрязать в подробностях, которые имеют отношение только к малому числу ситуаций? Впрочем, спецификации сенсорных систем зачастую — очень подробные, и инженеры должны тщательно их изучить перед выполнением серьезного проекта, чтобы убедиться в достижимости запланированных характеристик системы.

Рассматриваемый здесь датчик температуры должен соответствовать следующим функциональным требованиям:

- наличие одного канала термопары и одного канала компенсации холодного спая;
- опрос каждого канала должен выполняться 500 раз в секунду, предполагая, что частотный спектр сигнала температуры составляет максимум 15 Гц (дискретизация примерно с 33-кратным запасом по частоте);
- поддержка термопар типа K и J;
- обеспечение необходимой компенсации холодного спая для обоих типов термопар;
- обнаружение состояний разомкнутой и реверсированной термопары;
- фильтрация дискретных данных для устранения сетевого шума;
- предоставление пользователю возможности выполнять следующие действия через последовательный порт RS-232 на скорости 38,4 Кбит/с (восемь бит данных, один стоп-бит, отсутствие контроля на четность и управления потоком данных):
  а) выполнять калибровку каждого канала;
  б) задавать нижний и верхний безопасные пределы для каждого канала;
- ежесекундно извещать последовательный порт об измеренной температуре для всех каналов в текстовом виде через (если система об-

наруживает состояние сбоя конкретной термопары, то вместо данных о температуре должно быть передано сообщение об ошибке);
- выдавать извещение о выходе температуры за допустимые пределы путем индикации "----" с включением светодиода LED1 на демонстрационной плате, если измеряемая температура меньше нижнего предела, и путем индикации "++++" с включением светодиода LED2, если температура превысила верхний предел.

С целью уменьшения количества реализуемых схем это приложение отслеживает только два сигнальных канала, однако, с точки зрения эффективности, ничто не мешает увеличить число каналов до 16, поддерживаемых микросхемой dsPIC30F6014A. Хотя требуемая частота дискретизации может показаться несколько завышенной с точки зрения абсолютных величин, она позволит легко устранить сетевой шум без значительной задержки прохождения сигналов через систему. Как и в случае с количеством поддерживаемых каналов, частоту дискретизации можно при необходимости увеличить. Кроме того, мы используем крайне низкую частоту выдачи сообщений об измеренных температурах только потому, что эти данные предназначены для человека. Если бы они предназначались для считывания другими электронными компонентами системы, то следовало бы применить гораздо более скоростной бинарный протокол.

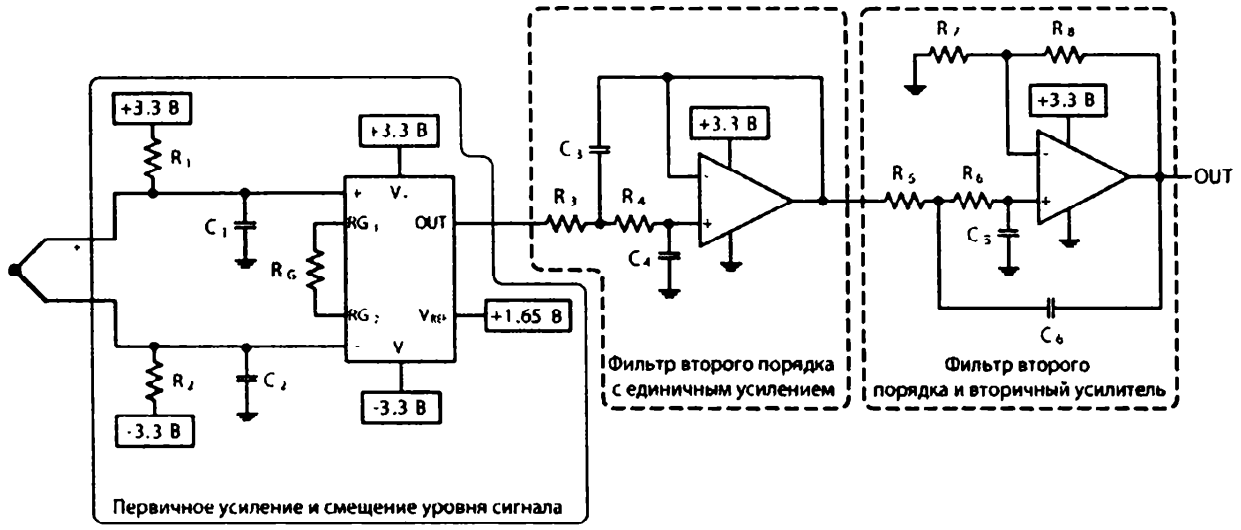


Программный код реализации этого протокола находится на прилагаемом к книге компакт-диске в файлах `BinaryProtocol.c` и `BinaryProtocolDef.h`.

Формирование сигналов от датчика

Как было отмечено ранее в этой главе, интерфейс датчика на основе термопары нуждается в:
- буферизации и усилении выходного сигнала термопары;
- компенсации холодного спая;
- обнаружении сбойных ситуаций для датчика.

Кроме того, в систему необходимо включить аналоговый фильтр защиты от наложения спектров, эффективно устраняющий частоты выше 250 Гц (поскольку частота дискретизации 500 Гц должна по крайней мере вдвое превосходить самую высокую частотную гармонику сигнала температуры). Мы предположили, что максимальная полоса частот для сигнала температуры составляет 15 Гц (т.е. частота дискретизации в 33 раза превышает теоретически необходимую), поэтому частоту отсечки фильтра также необходимо сократить до 15 Гц. Все это (для одного канала) реализует схема, показанная на рис. 6.13.



ПРИМЕЧАНИЯ:

1. R_1 и R_2 обеспечивают как входной путь тока в синфазном режиме, так и способ удостовериться в фактическом соединении термопары со схемой. Номинал резисторов должен быть идентичен и находиться в диапазоне 10–100 кОм во избежание нагрузки входного сигнала слишком низким входным сопротивлением, и в то же время, для обеспечения достаточно низкого сопротивления для получения эффективного входного пути тока.
2. Усиление для измерительного усилителя управляется резистором R_G согласно уравнению:
   Усиление = 1 + (50 кОм / R_G)
3. C_1 и C_2 устраняют высокочастотный входной шум. Конденсаторы установлены в обеих входных цепях, чтобы поддерживать сбалансированный дифференциальный входной импеданс.
4. Фильтры имеют частотную характеристику фильтра Баттерворта. В оконечный фильтрующий каскад включен усилитель для компенсации неспособности измерительного усилителя формировать выходные напряжения, близкие по значению к напряжениям шины питания. Усиление оконечного каскада определяется уравнением:
   Усиление = 1 + (R_8 / R_7)

Рис. 6.13. *Схема интерфейса термопары (для одного канала)*

Исследуем каждую часть этой схемы, чтобы лучше понять суть ее компонентов. Обратите внимание на то, что она сконфигурирована для работы при напряжении питания 3,3 В. Если схема будет выполнена на макетной плате dsPICDEM 1.1, в которой используется напряжение питания 5 В, то опорные напряжения 3,3 В необходимо заменить на 5 В; напряжения –3,3 В — на –5 В, напряжение 1,65 В — на 2,5 В.

Дифференциальный усилитель

Измерительный усилитель INA326 обеспечивает буферизацию и усиление дифференциального сигнала от термопары. Измерительные усилители прекрасно подходят для этой области применения, поскольку имеют очень высокое входное сопротивление и дифференциальные по природе. Обратите внимание на резисторы смещения, подключенные к каждой из линий передачи сигнала от термопары. Они обеспечивают обнаружение размыкания термопары и формируют цепь смещения постоянного тока, необходимую для правильной работы измерительного усилителя.

Фильтр защиты от наложения спектров

Главная задача фильтра защиты от наложения спектров — предотвратить зашумление дискретного сигнала частотами, превосходящими половинную частоту дискретизации. На первый взгляд может показаться, что для решения этой задачи лучше использовать аналоговый фильтр высокого порядка, чтобы обеспечить самую сильную фильтрацию, однако во многих случаях этот подход — не оптимальный. В предыдущей главе мы упоминали о недостатках, присущих аналоговым фильтрам (дрейф параметров, стоимость, габариты), причем усложнение фильтра только усугубляет эти проблемы. Поэтому лучше задействовать пару каскадных аналоговых фильтров Баттерворта второго порядка, а все остальное сделать с помощью фильтров ЦОС. Структура такого фильтра оптимально плоская в полосе пропускания и имеет довольно резкий спад в 20 дБ/дек.разряд/вывод[50] (рис. 6.14).

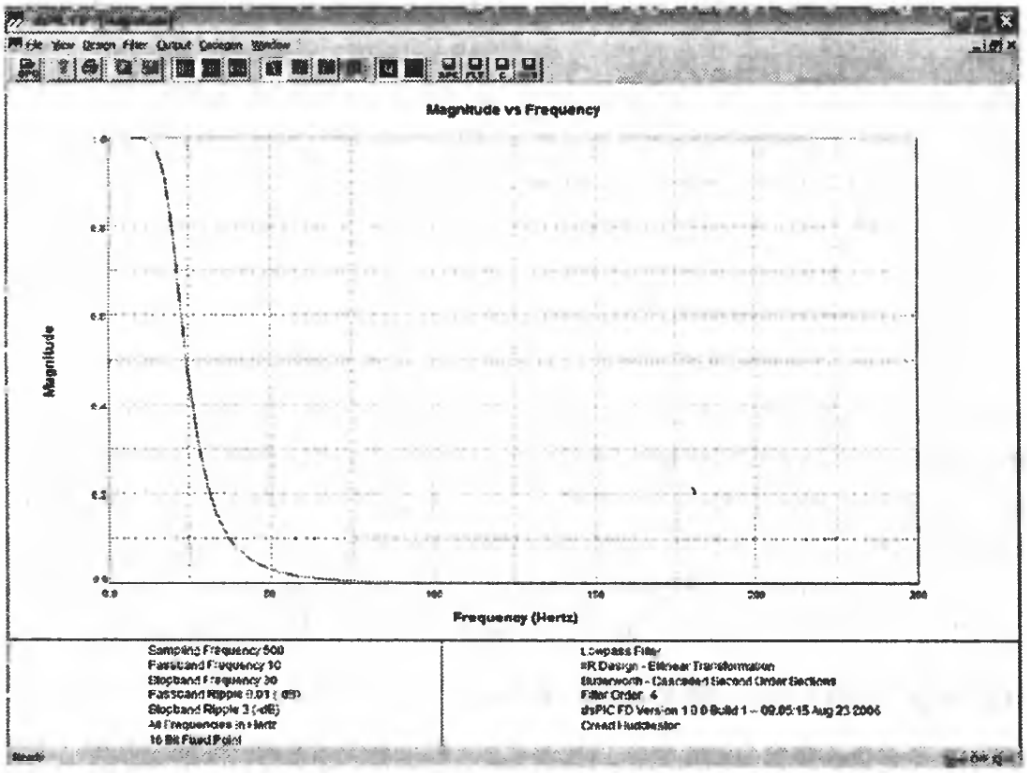


Рис. 6.14. Частотная характеристика фильтра Баттерворта

[50] Краткое, но информативное обсуждение использования структуры Баттерворта в качестве фильтра защиты от наложения спектров представлено на Web-сайте `www.daytronic.com/reference/aliasing.htm`.

Анализ цифровой фильтрации

В данном примере сигнал от сети рассматривается как внеполосный электрический шум, что позволяет для уменьшения шума в сигнале температуры использовать цифровой фильтр нижних частот с крутой характеристикой. Если бы наш температурный сигнал был более широкополосным, то пришлось бы также использовать узкополосный режекторный фильтр, как будет показано в следующей главе.

Существует ряд программных пакетов, предназначенных для разработки цифровых фильтров, однако в этой книге будет использован пакет Microchip Digital Filter Design System 1.0.0. Как и в случае с другими программными средствами, рассмотренными в этой книге, пакет Filter Design можно получить с помощью Web-сайта компании Microchip[51]. Одно из преимуществ этого программного пакета для проектирования фильтров заключается в том, что с его помощью можно быстро и легко получить графическое представление обратной связи для характеристик фильтра. Например, рис. 6.15 и рис. 6.16 показано влияние более свободных (рис. 6.15) и более жестких (рис. 6.16) требований к пульсации в полосе заграждения фильтра. Как видим, более жесткая спецификация пульсации также добавляет пульсацию в полосу пропускания, а также немного расширяет ее и создает более резкий переход между полосой пропускания и полосой заграждения.

Хотя фильтр с более свободными требованиями к пульсации в полосе заграждения, на первый взгляд, кажется более предпочтительным, поскольку его полоса пропускания более плоская, он пропускает только чуть больше 78% входного сигнала при частоте 6 Гц (см. рис. 6.15). В отличие от него, фильтр с более жесткими требованиями к пульсации в полосе заграждения хотя и имеет большее пульсации как в полосе пропускания, так и в полосе заграждения, пропускает больше сигнала при частоте 6 Гц (82%) и имеет более крутой переход между полосой пропускания и полосой заграждения (см. рис. 6.16). Таким образом, проектирование фильтров — это путь компромиссов, когда улучшение характеристик в одной области происходит за счет их ухудшения в другой.

---

[51] Номер полной версии программы Digital Filter Design по каталогу Microchip — SW300001. Сокращенная версия этого пакета стоит значительно меньше, однако отличается от полной тем, что не поддерживает большое количество отводов для КИХ- и БИХ-фильтров, а также не поддерживает MATLAB. Во многих случаях "облегченной" версии Digital Filter Design вполне достаточно. Ее номер по каталогу Microchip — SW300001-LT.

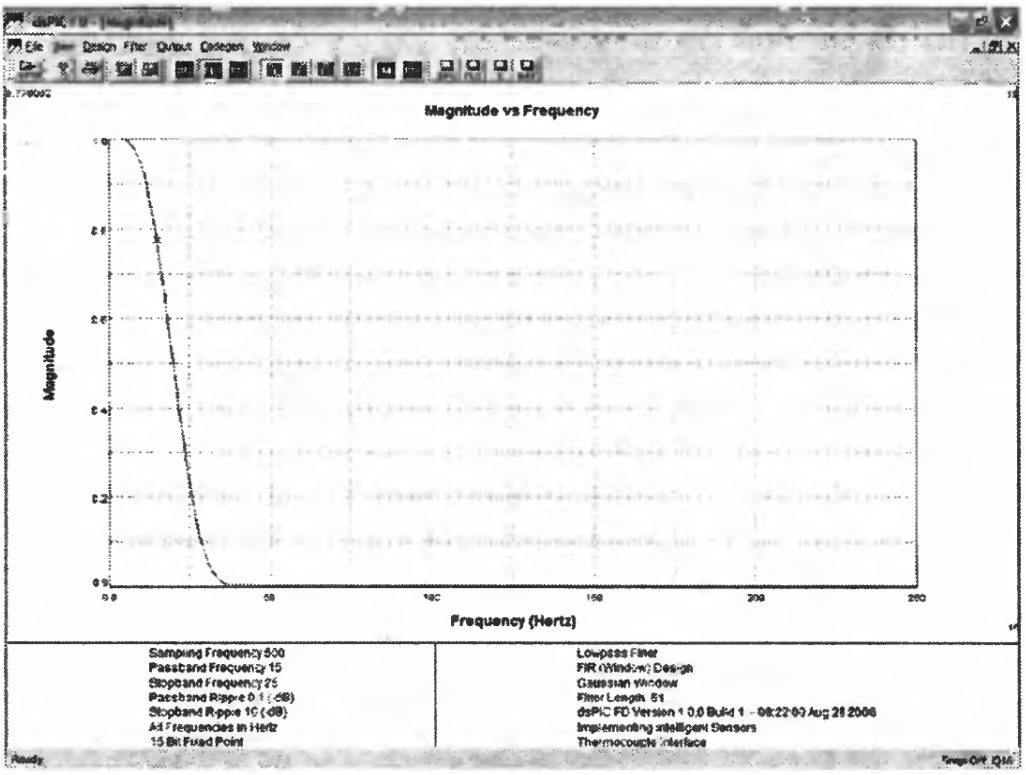


Рис. 6.15. *Характеристика фильтра для более свободных требований к пульсации в полосе заграждения фильтра*

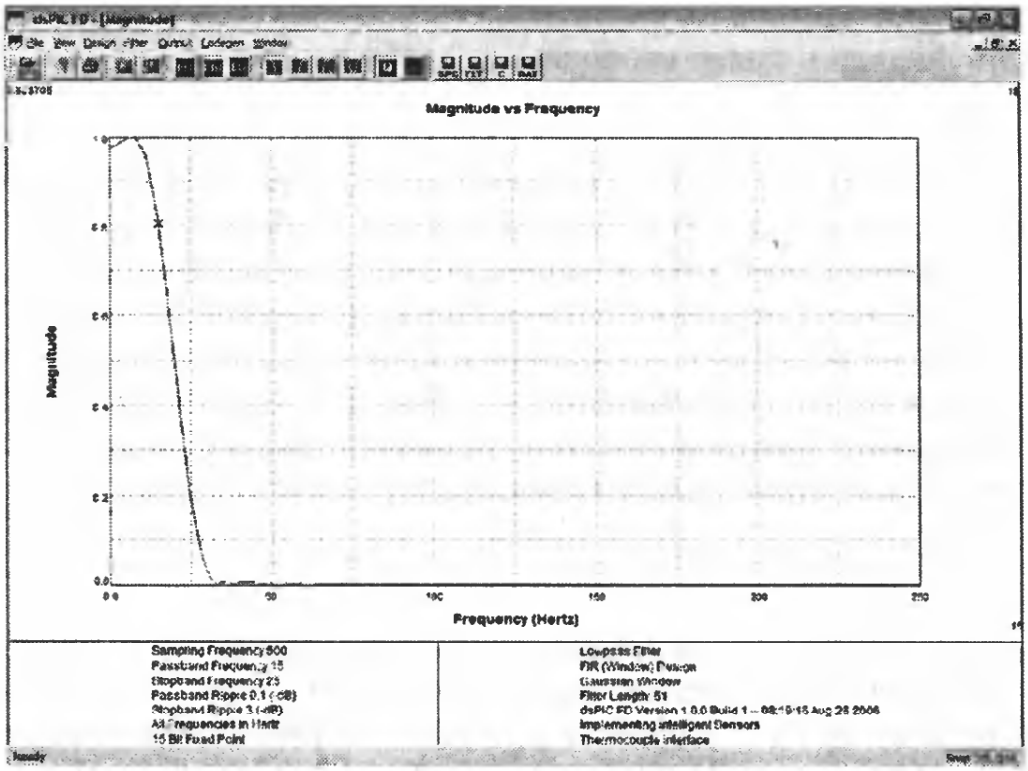


Рис. 6.16. *Характеристика фильтра для более жестких требований к пульсации в полосе заграждения фильтра*

В рамках разработки нашего приложения мы выберем более свободные требования к фильтрации, поскольку они соответствуют нашим потребностям и не оказывают неблагоприятного воздействия на сигнал. Принцип фильтрации выглядит так: "Как можно меньше" (но все же, достаточно для достижения требуемого результат). Этому принципу необходимо следовать всегда, поскольку он помогает избежать чрезмерного искажения исходного сигнала. В этом смысле он близок принципу, которым руководствуются врачи: "Прежде всего — не навреди!"[52].

Необходимо отметить еще один аспект цифрового фильтра. В обоих случаях фильтр имеет 51 отвод. Конечно, можно было бы повысить эффективность фильтра, добавив дополнительные отводы, однако это привело бы к увеличению задержки сигнала при его прохождении через фильтр. Для данного примера был преднамеренно выбран относительно короткий фильтр, чтобы снизить время реакции системы.

Алгоритмы анализа данных

Анализ данных в этом приложении довольно прост. В основном, требуется проверять каждую отфильтрованную выборку данных о температуре на предмет превышения установленных пользователем верхнего и нижнего безопасных пределов, и в случае обнаружения такого превышения выдавать световой сигнал. Блок-схема алгоритма для такого довольно ограниченного анализа данных показана на рис. 6.17.

Хотя представленный здесь анализ весьма прост, его можно легко расширить, включив более сложные процедуры или добавив, к примеру, алгоритм контроля за выводом данных. Это — именно тот тип структуры, которого всегда должны придерживаться разработчики: она проста, ясна и легко расширяема.

Протокол обмена данными

Разработчики приложений иногда по ошибке приравнивают интерфейс обмена данными, работающий в номинальных условиях, к тому, который работает во всех условиях, включая ошибки в физической передающей среде. Хороший проектировщик принимает во внимание все возможные ситуации и разрабатывает способ амортизации отказов в системе, когда она входит в нежелательные состояния. Такая аморти-

---

[52] Хотя выражение "прежде всего — не навреди" приписывают Гиппократу, в его клятве оно отсутствует. Один из возможных источников — это труд "Эпидемии", в котором Гиппократ говорит, что врачи должны "... при лечении болезни преследовать две главные цели: сделать лучше и не навредить" (http://ancienthistory.about.com/od/greekmedicine/f/HippocraticOath.htm).

зация отказов чрезвычайно важна. Даже если система не может откорректировать ошибку, о ней должен быть уведомлен хост, после чего должны быть предприняты все необходимые меры для обеспечения безопасного состояния системы. И для достижения этой задачи огромную роль играет реализация основанного на состояниях интерфейса обмена данными, который автоматически повторно инициализируется при каждом поступлении непредвиденного входного сигнала.

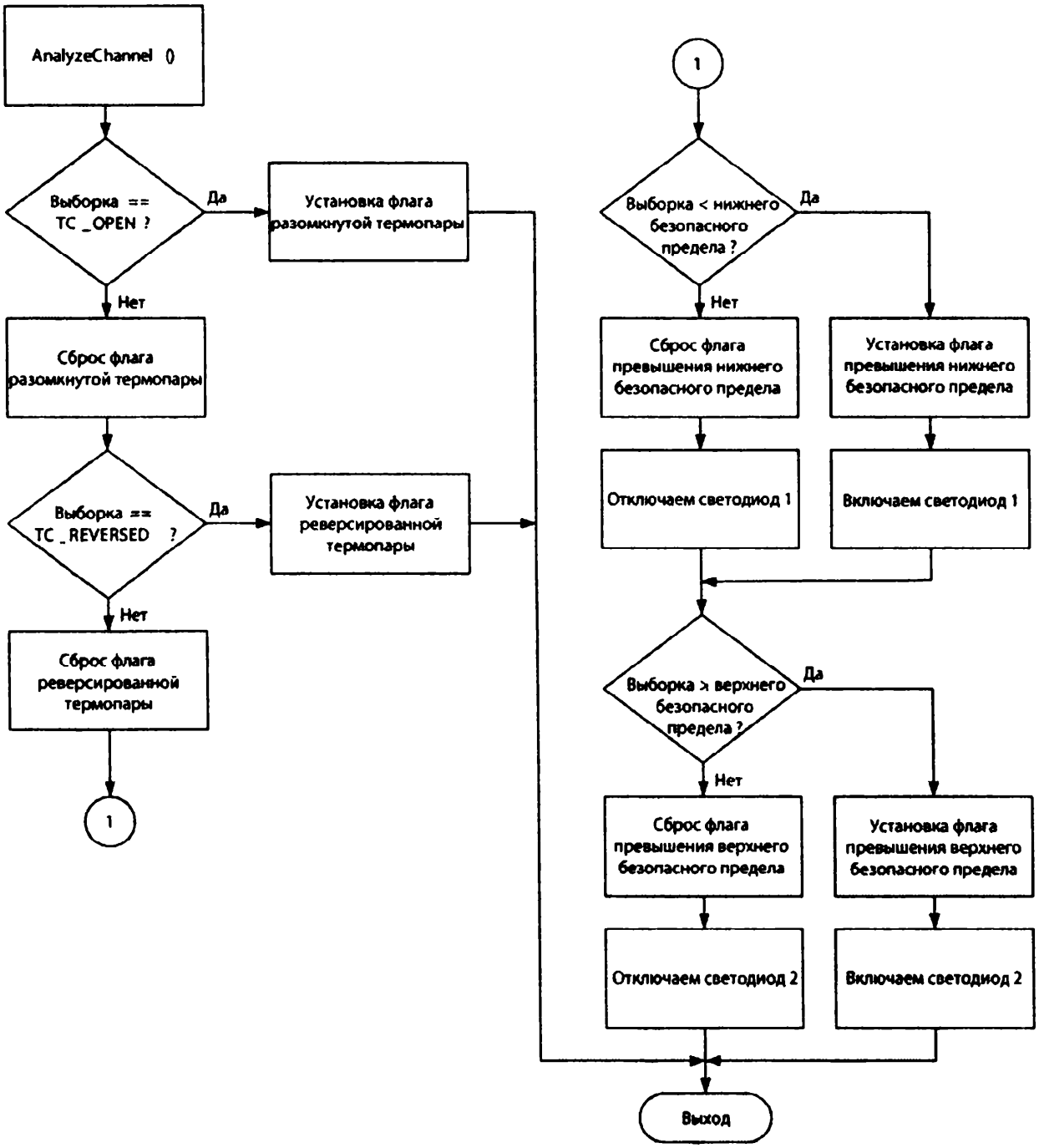


Рис. 6.17. Блок-схема алгоритма анализа данных для датчика на основе термопары

В нашем первом приложении мы воспользуемся простым, ориентированным на человека-оператора протоколом, работающим через стандартный последовательный порт RS-232. Этот подход позволяет не только фактически взаимодействовать пользователю с системой, но и слегка соприкоснуться с миром обработчиков коммуникационных состояний. Из двух названных аспектов, последний — более важный, поскольку на удивление много систем поставляется со слабой и ненадежной реализацией интерфейса обмена данными. В мире совсем немного вещей, способных так разочаровывать пользователей, как необъяснимые отказы системы (при этом часто возникает впечатление, что система "зависла"). Если же такие отказы возникают по причине плохой реализации информационного обмена, то они просто непростительны.

А теперь выясним, что же подразумевается под обработкой коммуникационных состояний, и как ее применить в процессе разработки систем. В обработке, основанной на состояниях, действия приложения зависят как от текущего состояния обработчика (значение, хранимое в глобальной переменной), так и от обрабатываемых данных. При этом значение переменной состояния может изменяться или оставаться неизменным. Поясним эту концепцию на примере.

На рис. 6.18 показана диаграмма состояний для простого обмена данными.

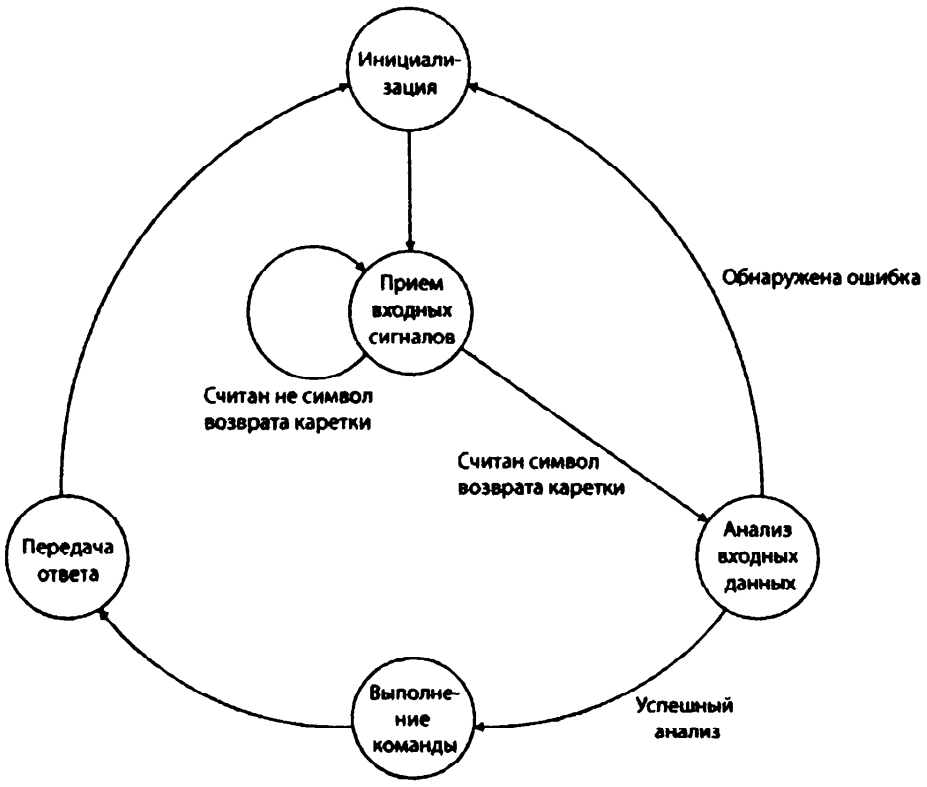


Рис. 6.18. *Простой конечный автомат для управления обменом данными*

Каждой окружности в этой диаграмме соответствует отдельное состояние обработки данных, а линиям, соединяющим отдельные состояния, — *условия перехода*, т.е. условия, требуемые для выполнения перехода из одного состояния в другое. Если с некоторой линией перехода не сопоставлено ни одно условие, то это означает, что по завершении всех операций в начальном состоянии конечный автомат сразу же перейдет в конечное состояние. Например, линия перехода без условия ведет из состояния "Инициализация" к состоянию "Прием входных сигналов". Это означает, что конечный автомат перейдет в состояние "Прием входных сигналов", как только будут завершены все операции в состоянии "Инициализация".

Как только конечный автомат перейдет в состояние "Прием входных сигналов", он будет оставаться в нем до момента приема ASCII-символа возврата каретки. Этому на схеме соответствуют две линии перехода с условиями "Считан не символ возврата каретки" и "Считан символ возврата каретки". Первая из них просто приводит к возврату в состояние "Прием входных сигналов", а вторая ведет к состоянию "Анализ входных данных". Всякий раз, когда конечный автомат принимает новые данные, он обрабатывает их в состоянии "Прием входных данных", а затем или остается в этом состоянии, или переходит к состоянию "Анализ входных данных".

Обычно обработка, выполняемая в состоянии "Прием входных данных", заключается в добавлении новых данных в буфер (при этом предполагается, что в буфере есть свободное место). Впрочем, в некоторых приложениях все может оказаться гораздо сложнее. Но в любом случае, прием символа возврата каретки сообщает конечному автомату о необходимости выполнить анализ полученных данных с целью идентификации ожидаемого пользователем действия.

Если обработка данных в состоянии "Анализ входных данных" привела к успешному извлечению команды пользователя из входного буфера, то конечный автомат переходит в состояние "Выполнение команды", которое выполняет задачу, определенную командой пользователя. С другой стороны, если в процессе анализа в пользовательской команде обнаружена ошибка, то конечный автомат возвратится в состояние "Инициализация", т.е. будет повторно инициализирован в заведомо безопасное состояние в ожидании ввода данных от пользователя. При такой организации система изолирована от ошибок, которые могут присутствовать во вводимых пользователем входных данных. Пользователи могут (и вероятнее всего будут) совершать ошибки, однако эти ошибки не повлияют на надежность системы.

В состоянии "Выполнение команды" решаются задачи, связанные с конкретной командой и ее параметрами. Как только задача выполнена, конечный автомат переходит в следующее состояние "Передача ответа", в котором сообщается о результатах выполнения команды. И, наконец, после передачи хосту результата выполнения команды конечный автомат опять возвращается в состояние "Инициализация", чтобы повторно инициализировать переменные для подготовки к приему следующей команды.

Конечно же, конечный автомат можно реализовать аппаратно (например, на программируемой логической матрице), однако намного удобнее и дешевле сделать это программно (предполагая, конечно, что процессор может достаточно быстро проходить в цикле требуемые состояния). Такая реализация очень проста, однако для тех, кто никогда раньше не разрабатывал программный конечный автомат, хотелось бы дать несколько практических советов.

В случае программной реализации конечного автомата для отслеживания конкретного состояния (как, впрочем, и любой информации, которая должна сохраняться при циклических переходах между состояниями) в приложении следует использовать глобальные переменные. Сам конечный автомат (по крайней мере, на верхних уровнях) обычно реализуется в виде одной функции, которая повторно вызывается при поступлении новых входных данных. Если для создания приложения используется язык программирования C, то такая функция обычно содержит оператор `switch`, причем в качестве параметра этого оператора используется текущее состояние обработки данных. Например, программный код рассматриваемого нами примера может соответствовать листингу 6.1.

Листинг 6.1. Реализация простого конечного автомата обмена данными

```
            //  Определение состояний обмена данными
#define COMM_ST_INIT              0  // Инициализация конечного
                                     // автомата
#define COMM_ST_ACCEPT_INPUT      1  // Прием входных данных
#define COMM_ST_PARSE_INPUT       2  // Анализ введенной строки
#define COMM_ST_EXEC_CMD          3  // Выполнение команды
#define COMM_ST_TX_RESPONSE       4  // Передача результатов
                                     // выполнения команды

            //  Переменные состояния

Uint8 g_ui8CommState = COMM_ST_INIT;  //Глобальное состояние
                                      // процесса обмена данными
```

Листинг 6.1. Продолжение

```
*****************************************************************
ФУНКЦИЯ:     CommProcRxData(Uint8 ui8NewData)
ОПИСАНИЕ:    Эта функция реализует высокоуровневый конечный
             автомат для процесса обмена данными. Она
             вызывается из главного цикла всякий раз, когда
             процессор принимает новые данные. При этом
             функция циклически проходит соответствующие
             состояния, опираясь на текущее состояние
             обработки и новые входные данные
ПАРАМЕТРЫ:   ui8NewData - новые входные данные для обработки
РЕЗУЛЬТАТ:   Функция возвращает новое состояние обмена данными
ВЕРСИЯ: 0    v1.0
  Оригинальная версия
*****************************************************************

Uint8 CommProcRxData(Uint8 ui8NewData)
{
   // Локальные переменные
   Bool bProcessingState = TRUE;  //Флаг продолжения обработки
                                  // (устанавливается для
                                  // продолжения обработки
                                  // состояния, сбрасывается
                                  // для прекращения обработки
                                  // после текущего состояния)
// Идентификация текущего состояния обмена данными
// и соответствующая обработка новых данных
   while (bProcessingState)
   {
      switch (g_ui8CommState)
      {
         case COMM_ST_INIT:
            // Инициализация конечного автомата
            ...  // Операции, специфичные для состояния
            bProcessingState = FALSE; // Завершение обработки
                                      // для даного цикла
            // Переход в следующее состояние
            g_ui8CommState = COMM_ST_ACCEPT_INPUT;
            break;
         case COMM_ST_ACCEPT_INPUT:
            // Прием новых входных данных
            ... // Операции, специфичные для состояния
            // Переход в следующее состояние
            if (ui8NewData =_ '\r')
               // Выполнить анализ данных, если принята
               // целая строка
               g_ui8CommState = COMM_ST_PARSE_INPUT;
```

Листинг 6.1. Окончание

```
            else
               bProcessingState = FALSE; // Завершение цикла
            break;
         case COMM_ST_PARSE_INPUT:
            // Анализ всего буфера входных данных
            ... // Операции, специфичные для состояния,
                // возвращающие значение bError для
                // идентификации успешного завершения анализа

            // Переход в следующее состояние на основании
            // успешности анализа команды
            if (bError)
               // Обнаружена ошибка, сброс конечного автомата
               // и ожидание следующей команды
               g_ui8CommState = COMM_ST_INIT;
            else
               // Ошибок нет – выполнение команды
               g_ui8CommState = COMM_ST_EXEC_CMD;
            break;
         case COMM_ST_EXEC_CMD:
            // Выполнение проанализированной команды
            ... // Операции, специфичные для состояния
            // Переход в следующее состояние
            g_ui8CommState = COMM_ST_TX_RESPONSE;
            break;
         case COMM_ST_TX_RESPONSE:
            // Передача результатов выполнения команды
            ...  // Операции, специфичные для состояния
            // Переход в следующее состояние
            g_ui8CommState = COMM_ST_INIT;
            break;
         default:
            // Управление НИКОГДА не должно быть передано
            // сюда, поэтому – сброс конечного автомата
            g_ui8CommState = COMM_ST_INIT;
            break;
      }
   }
   return g_ui8CommState;
}
```

Хотелось бы обратить внимание на некоторые аспекты программного кода конечного автомата. Так, мы используем глобальную переменную `g_ui8CommState`, которая отражает состояние обработки данных. Значение `g_ui8CommState` инициализируется при объявлении переменной. Это гарантирует, что конечный автомат начнет работать с из-

вестного состояния (состояния инициализации). Функция обновляет значение `g_ui8CommState` всякий раз, когда конечный автомат должен изменить свое состояние на новое, и возвращает конечное состояние после завершения всей обработки данных в текущем состоянии. Однако, поскольку текущее состояние уже сохранено в глобальной переменной, то возвращаемое значение служит лишь для информационных целей.

Для обеспечения правильной работы конечного автомата модифицировать переменную `g_ui8CommState` должна только функция `CommProcRxData()`. Даже если другие функции отслеживают состояние по значению, возвращаемому функцией `CommProcRxData()`, они не должны изменять значение переменной `g_ui8CommState`, потому что это может привести к скрытым программным взаимодействиям, вызывающим проблемы при последующих доработках программы.

Диаграмма состояний, представленная на рис. 6.18, показывает, что для перехода в некоторые состояния требуется ввод данных или возникновение специфических условий.

6.4. Аппаратная реализация

Структурная схема аппаратной реализации, которая будет рассмотрена в следующих подразделах, представлена на рис. 6.19.

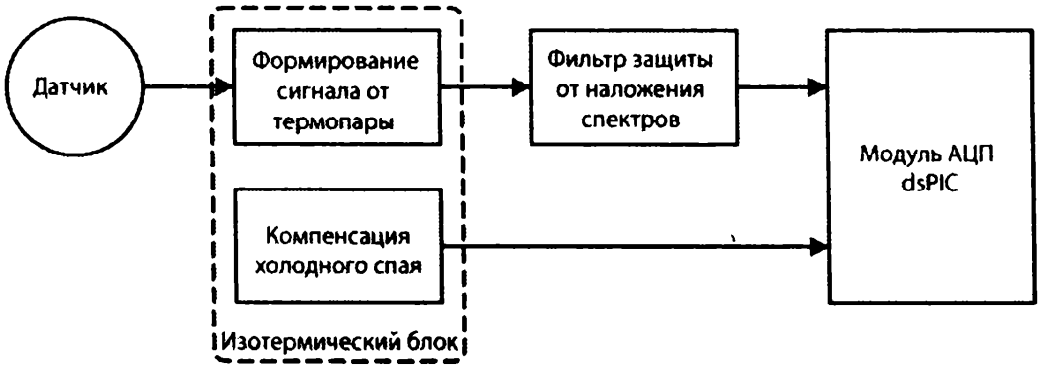


Рис. 6.19. Блок-схема аппаратных средств

Обращаем внимание на то, что это — только один из подходов к построению схем, и читатели, имеющие определенный опыт проектирования аналоговых устройств, могут модифицировать рассматриваемые схемы на свое усмотрение.

Аналоговый усилитель и фильтр защиты от наложения спектров

Принципиальная схема аналогового усилителя и фильтра защиты от наложения спектров показана на рис. 6.13. Усилитель реализован в виде

однокристального измерительного усилителя, выходной сигнал которого подается на фильтр нижних частот Баттерворта четвертого порядка, состоящий из нескольких пассивных компонентов и двух операционных усилителей с размахом выходного сигнала, равным напряжению питания. При реализации этой схемы необходимо учитывать три основных требования:

- минимизация потребляемой мощности для уменьшения самонагревания;
- ясная, компактная разводка печатной платы, сводящая к минимуму перекрестные помехи и электрический шум;
- хорошая развязка источника питания для уменьшения электрического шума.

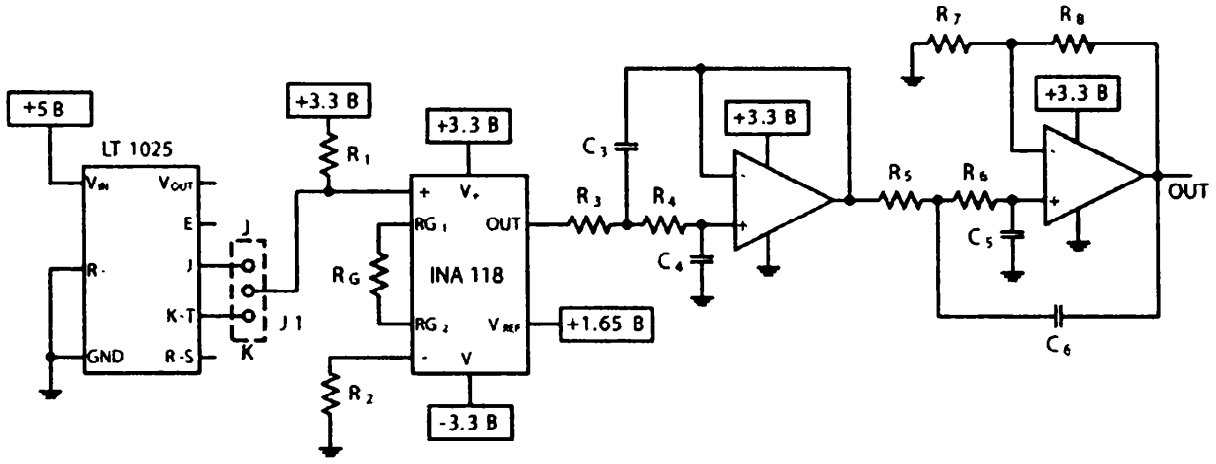
Малая мощность потребления достигается, прежде всего, за счет использования маломощных интегральных схем. Поскольку на выходе термопар получают сигнал низкого уровня, крайне важна правильная разводка печатной платы. Проводники, ведущие к термопаре, должны быть максимально короткими. При этом проводники, ведущие к одной термопаре, должны быть расположены как можно ближе друг к другу для максимальной защиты от синфазных помех (это гарантирует, что термический и электрический шум в одинаковой степени влияет на оба проводника). Также крайне важно использовать хороший источник питания с минимальными пульсациями и дрейфом, иначе в систему будет проникать шум от самого источника (в противоположность шумам на выводах термопары).

Компенсация холодного спая

Хотя компенсацию холодного спая можно реализовать несколькими различными способами, в рассматриваемой схеме для этого задействована микросхема LT1025 от компании Linear Technologies (рис. 6.20). Эта схема обеспечивает возможность переключения на использование термопар типа J или K путем перестановки перемычки J1.

Выходное напряжение LT1025 буферизуется, а затем измеряется контроллером dsPIC, который для удаления из сигнала термопары соответствующего напряжения холодного спая использует программный алгоритм. Для обеспечения максимальной точности можно для каждого входа термопары использовать отдельную микросхему LT1025, однако на практике считается приемлемым разделить одну LT1025 на две термопары, когда микросхема размещается между парами проводников каждой термопары (т.е. пара проводников одной термопары размещается с одной стороны, а пара проводников другой термопары — с другой

стороны LT1025). Одну микросхему LT1025 можно разделить и между большим числом термопар, однако, чем дальше схема компенсации холодного спая размещена от фактической концевой заделки проводников термопары, тем менее эффективно она будет выполнять компенсацию.



Примечания:

1. Номиналы резисторов и конденсаторов должны быть согласованы с номиналами резисторов и конденсаторов, используемых в каналах обработки сигналов от термопар.
2. В случае компенсации холодного спая для термопары типа J перемычка J1 должна соединить со входом измерительного усилителя вывод J, а в случае термопары типа K — вывод K.

Рис. 6.20. Схема компенсации холодного спая

Изоляция сигнала

Уровень требуемой изоляции сигнала зависит от среды, в которой будет работать датчик. Во многих ситуациях требования по изоляции сигнала относительно минимальны, поскольку датчик не будет подвергаться воздействию жесткой или крайне восприимчивой среды. В других же обстоятельствах (например, когда термопары находятся вблизи электрических нагревательных элементов или используются для измерения температуры у людей), уровень изоляции должен быть намного выше, чтобы защитить схему датчика или всю измерительную систему.

Как и можно было ожидать, существуют различные способы изолирования сигналов термопар, каждый их которых отличается по стоимости, сложности и силе деградации сигнала. Наименее дорогостоящая методика — просто задействовать между схемой датчика и отслеживаемым элементом сопротивления очень большого номинала. При таком подходе хорошим вариантом с точки зрения стоимости был бы простой буфер на операционном усилителе с высокоомными входами, который мог бы обеспечить входное сопротивление от нескольких до сотен МОм. Приняв во внимание как цену, так и характеристики, можно отме-

тить, что измерительный усилитель типа INA118 от компании Texas Instruments имеет входное сопротивление 10^{10} Ом и может выдерживать на своих входах синфазное напряжение вплоть до 40 В. Это чрезвычайно полезно, если датчик должен функционировать в среде, где существует вероятность смешения сигнала от термопары и синфазного напряжения.[53] Как уже было отмечено ранее, это может произойти, если термопара закорочена на линию питания (или другое ненулевое напряжение). Если входная цепь не сможет справиться с дополнительным напряжением, то датчик выйдет из строя.

Перенося ответственность за защиту от перенапряжения на саму микросхему, проектировщик экономит место на печатной плате и сокращается затраты на детали. Однако следует отметить, что защиты от синфазного напряжения 40 В для большинства промышленных сетей, имеющих диапазон переменного напряжения 110–480 В, недостаточно. Если система должна быть защищена от подобного напряжения, то необходимо применить дополнительные схемы защиты, понижающие максимальные входные напряжения до уровней, с которыми может справиться микросхема.

К верхнему срезу спектра цены/производительности относятся методики оптической изоляции сигнала термопары от остальной системы. Это может быть реализовано с помощью оптически изолированных аналоговых усилителей (очень дорогостоящие) или путем оцифровки неизолированного усиленного сигнала от термопары с помощью внешнего АЦП с последующим применением оптически изолированного цифрового интерфейса для передачи оцифрованных данных процессору (менее дорогостоящий подход). Хотя оптическая изоляция только цифрового интерфейса АЦП менее дорогостоящая, она может оказаться неприемлемой в ситуациях, когда датчик требует ограничения токов и/или напряжений, поступающих в контролируемую систему (например в медицинских задачах).

В нашем примере будет применен подход с изоляцией измерительного усилителя, поскольку он прост в реализации, относительно недорог и обеспечивает превосходные рабочие параметры. Схема для одного канала изоляции сигнала с использованием измерительного усилителя INA326 показана на рис. 6.13.

---

[53] Существует множество измерительных усилителей различных изготовителей, которые прекрасно подойдут в данном примере. Устройство Texas Instruments — это только один из вариантов, с которым хорошо знаком автор.

6.5. Микропрограммная реализация

Для упрощения микропрограммной разработки приложение для доступа к периферийным устройствам ЦКС dsPIC и реализации функций ЦОС, наподобие фильтрации, будет интенсивно использовать 16-разрядную библиотеку от компании Microchip.

Дискретизация сигнала

В данном приложении для дискретизации сигнала от термопары используется внутренний 12-разрядный АЦП ЦКС dsPIC. Как было сказано в главе 3, "Возможности цифрового контроллера сигналов dsPIC", для конфигурирования модуля АЦП, необходимо знать:

- выводы порта, которые будут задействованы в качестве аналоговых входов;
- частоту дискретизации;
- будет ли применена выборка с чередованием или же после преобразования всех сигналов в течение заданного периода дискретизации будет формироваться одиночный запрос на прерывание;
- требуется ли мультиплексирование буфера результатов;
- формат преобразованных данных.

Поскольку мы используем макетную плату dsPICDEM, то некоторые аппаратные решение уже были приняты за нас. Так, плата dsPICDEM использует встроенный стабилизатор на 5 В для V_{DD} и AV_{DD}, а для синхронизации dsPIC — кварц на 7,3728 МГц совместно с системой ФАПЧ с четырехкратным умножением тактовой частота (т.е. фактически частота составляет 29,4912 МГц). Кроме того, несколько аналоговых входов выделены специально под функции платы dsPICDEM (табл. 6.2).

Таблица 6.2. Назначение входных аналоговых сигналов

Сигнал	Целевой	Описание
AN0/RB0	Да	Данные для программирования (PGD) от ICD2
AN1/RB1	Да	Тактирование программирования (PGC) от ICD2
AN2/RB2	Нет	Доступно пользователю
AN3/RB3	Да	Цифровой потенциометр, сигнал которого прошел через фильтр нижних частот
AN4/RB4	Да	Аналоговый потенциометр RP2 — от 0 до AV_{DD}
AN5/RB5	Да	Аналоговый потенциометр RP3 — от 0 до AV_{DD}
AN6/RB6	Да	Аналоговый потенциометр RP 1 — от 0 до AV_{DD}
AN7/RB7	Нет	Доступно пользователю

Таблица 6.2. Окончание

Сигнал	Целевой	Описание
AN8/RB8	Да	Датчик температуры, сигнал которого прошел через фильтр нижних частот
AN9/RB9	Нет	Доступно пользователю
AN10/RB10	Нет	Доступно пользователю
AN11/RB11	Нет	Доступно пользователю
AN12/RB12	Нет	Доступно пользователю
AN13/RB13	Нет	Доступно пользователю
AN14/RB14	Нет	Доступно пользователю
AN15/RB15	Нет	Доступно пользователю

Обращаем внимание на то, что выводы AN0 и AN1 выделены для программных данных (PGD) и программной синхронизации (PGC) соответственно только тогда, когда установлена соответствующие перемычки в блоке перемычек J8. Если эти перемычки не установлены, то приложение может использовать выводы AN0/RB0 и AN1/RB1 как аналоговые входы или как цифровые выводы порта.

В нашем примере вывод AN11 используется в качестве входа компенсации холодного спая, а AN7 — в качестве входа термопары. Эти входы отмасштабированы с тем, чтобы использовать полный диапазон аналогового напряжения питания (от AV_{SS} до AV_{DD}).

Реализация цифрового фильтра

Благодаря библиотеке ЦОС от компании Microchip, а также программе dsPIC Filter Design™, простая цифровая фильтрация, используемая в нашем примере, заключается, главным образом, в формировании правильных коэффициентов в пакете dsPIC Filter Design с последующей реализацией структуры цифрового КИХ-фильтра с помощью функций, включенных в библиотеку ЦОС. Но прежде рассмотрим базовую структуру, которую использует упомянутая библиотека для обработки данных. Кроме того, мы должны понимать суть ограничений, налагаемых этой библиотекой на наше приложение: как общих, так и специфичных для подпрограмм фильтрации.

Библиотека ЦОС для хранения данных использует *дробные векторы*, которые представляют собой обычные массивы из двухбайтовых элементов с числовыми значениями в формате 1.15. Библиотека предъявляет к дробным векторам два требования:

- массив элементов должен быть непрерывным (т.е. связанные списки не допускаются);

- массив должен начинаться с четного адреса памяти, поскольку его элементы — 16-разрядные.

Эти небольшие ограничения являются прямым следствием использования аппаратных средств dsPIC и позволяют библиотеке задействовать для оптимизации математических операций специализированный процессор ЦОС. Также важен тот факт, что библиотечные функции, в основном, рассчитаны на работу с дробными векторами, размещенными в используемой по умолчанию памяти RAM (память X- и Y-данных).[54] Подпрограммы фильтрации налагают дополнительные ограничения на размещение определенных данных в памяти. Так, входные и выходные выборки могут размещаться в используемом по умолчанию пространстве RAM (память X- или Y-данных), однако коэффициенты фильтрации могут располагаться только в области X-данных или же в памяти программ, а значения временных задержек — *только* в области Y-данных.[55]

Самой собой, для обработки векторов применимы все обычные математические правила и общепринятые подходы программирования (например, целевые буферы векторов должны быть достаточно большими для хранения результатов математических операций). Читатель должен учесть, что для обеспечения быстродействия библиотеки ЦОС не выполняют никакой проверки корректности входных или выходных данных, а их функции не используют флаги насыщения и переполнения. Программист должен явным образом проверять флаги после каждого обращения к библиотеке.

В нашем примере будут использованы следующие параметры:

- частота дискретизации — 500 Гц;
- частота полосы пропускания — 15 Гц;
- частота полосы заграждения — 25 Гц;
- пульсация в полосе пропускания — 0,1 дБ;
- пульсация в полосе заграждения — 3 дБ;
- тип фильтра — Гаусса.

---

[54] Это примечание приведено в разделе "User Considerations" руководства пользователя 16-разрядной библиотеки Language Tools Library.

[55] Требования по использования пространства памяти подпрограммами ЦОС-фильтрации можно найти в разделе "Fractional Filter Operations" руководства пользователя 16-разрядной библиотеки Language Tools Library.

Ввод этих параметров в программе dsPIC Filter Design и генерирование программного кода для результирующего фильтра сформирует требуемую структуру данных фильтрации.



Структура фильтра находится на прилагаемом к книге компакт-диске в папке Коды в файле `SensorFilter.s`.

Реализация анализа данных

Анализ данных в нашем приложении крайне прост: преобразование отфильтрованных данных в соответствующую температуру с последующим сравнением этой температуры с верхним и нижним пределом. Хотя этот процесс — только базовый, выполняемые операции формируют элементы не только системы отслеживания температуры, но и любой другой системы температурного контроля. Никогда не следует недооценивать силу простоты...

Реализация обработки ошибок

Как и в случае с анализом данных, алгоритм обработки ошибок в нашем примере очень прост. Следует отметить, что в данном контексте под ошибками подразумеваются неожиданные условия, которые неблагоприятно влияют на работу системы, а не выход температуры за пределы допустимого диапазона, что контролируется функциями анализа данных. Разница тонка, однако важна. Хотя выходить за пределы диапазона температур нежелательно, с точки зрения системы это абсолютно допустимо, и возникновение подобного условия не нарушит ее нормальной работы. Состояние же ошибки, о котором сейчас идет речь, — это нечто совершенно другое. Если его не обнаружить, система будет выдавать неверную информацию и действовать на ее основании, что может привести к катастрофическим последствиям.

Мы будем отслеживать два основных состояния ошибки, обычно возникающих в системах измерения температуры: разомкнутая термопара и реверсированная термопара. Каждое из двух состояний имеет свои особенности и по-разному воздействует на работу системы. Например, разомкнутая термопара считается катастрофической неисправностью, которая может быть обнаружена приложением, но не может быть исправлена. В то же время, система может компенсировать реверсированную термопару. Состояние разомкнутой термопары относительно просто распознать, просто сопоставляя уровень "сырого" входного сигнала с некоторым пороговым значением. Поскольку мы подключили к входам термопары специальные резисторы, нарушение спая приведет к повышению входного сигнала почти до уровня напряжения питания,

которое явно находится за пределами допустимого диапазона любого сигнала от термопары.

Обнаружить состояние реверсированной термопары сложнее, поскольку существуют диапазоны температур, которые производят "допустимый" выходной сигнал термопары, когда ее выводы перепутаны местами, т.е. выходные значения находится в пределах уровней напряжения, ожидаемых во всем диапазоне температур. Единственный способ точно распознать реверсированную термопару — это контролировать напряжение на ее выходе при изменении температуры. В случае нормального, линеаризованного сигнала термопары выходное напряжение с ростом температуры повышается; если же выводы термопары реверсированы, то с ростом температуры ее выходное напряжение понижается.

Если в состав системы входит какой-нибудь источник тепла (например, нагревательный элемент), соединенный с термопарой, то система может включать на короткое время нагреватель и контролировать повышение напряжения на выходе соответствующей термопары. Если с ростом температуры напряжение повышается, то термопара смонтирована верно (по крайней мере, с точки зрения полярности выводов), если же выходное напряжение термопары уменьшается, то это означает, что термопара реверсирована. Очевидно, что при наличии в составе системы охлаждающего элемента (например, холодильника) может применяться тот же подход, но только теперь на выходе правильно смонтированной термопары следует ожидать понижения напряжения.

Трудности возникают, если системе недоступен ни тепловой, ни охлаждающий источник. В этом случае единственное решение заключается в ожидании момента, когда уровень выходного сигнала термопары превысит ожидаемые пределы напряжения в отрицательном (если система нагревается) или в положительном (если система охлаждается) направлении. При возникновении такого условия система может установить флаг ошибки и выдать пользователю соответствующее сообщение.

Реализация протокола обмена данными

Коммуникационный протокол, используемый во всех трех рассмотренных в этой книге примерах, обсуждается в главе 4, "Умение быть хорошим коммуникатором". Для передачи данных через стандартный интерфейс RS-232 в центральную систему мы используем модуль UART 1. Поскольку вся необходимая нам поддержка реализована в базовой командной структуре протокола, для данного примера никаких расширений в нее вносить не нужно. Для того чтобы проиллюстриро-

вать реализацию связи с помощью текстового протокола, рассчитанного на человека, или бинарного протокола, рассчитанного на машинное считывание, на прилагаемом к книге компакт-диске представлены соответствующие примеры программного кода. Проектировщик может применить любую методику, просто удалив комментарии в требуемой строке заголовочного файла `ProtocolDef.n`.



Примеры реализации протоколов находятся на прилагаемом к книге компакт-диске в папке Коды в файлах `BinaryProtocol.c` и `TextProtocol.c`.

Базовый подход к обработке данных заключается в проверке получения данных в главном цикле обработки с помощью функции `CommIsRxPending()`. Если доступны новые последовательные данные, то они считываются с помощью функции `CommGetRxData()`, а затем обрабатываются путем вызова функции `ProtocolProcRxData()` с только что полученным символом. Эта функция выполняет синтаксический анализ данных и в случае получения полного командного сообщения вызывает соответствующий обработчик команды с помощью функции `ProtocolProcMsg()`. После идентификации команды вызывается соответствующая функция обработки низкого уровня, а результат передается обратно хосту с помощью специфической для команды подпрограммы формирования ответа. Как только ответ был послан, функция `ProtocolProcRxData()` сбрасывает конечный автомат синтаксического анализа, чтобы система начала ожидать следующее сообщение.

6.6. Резюме

В этой главе мы кратко рассмотрели процесс проектирования системы отслеживания температуры. Теперь мы можем оцифровать аналоговый сигнал от датчика, отфильтровать его для удаления нежелательного шума, проанализировать отфильтрованные данные, чтобы извлечь полезную информацию, и передать эту информацию другим компонентам системы. По мере разработки более сложных систем мы продолжим использовать рассмотренную программную структуру в качестве основания для построения приложений. В следующей главы мы немного расширим эту структуру в ходе проектирования нашего второго проекта: системы контроля давления/нагрузки.

7

Реальный пример: датчики давления и нагрузки

Без давления не бывает алмазов.
— Мэри Кейс

Разработав в предыдущей главе датчик температуры, мы получили определенный практический опыт в области проектирования сенсорных систем. Опираясь на этот опыт, далее мы расширим базовую структуру датчика таким образом, чтобы она поддерживала измерение нагрузки и давления. Следует отметить, что датчики нагрузки, измеряющие вес, и датчики давления, измеряющие воздействие веса на определенную область, находят широкое применение в самых разнообразных сферах. Будь то подсчет количества крошечных электронных элементов, измерение кровяного давления у пациента или же управление давлением на сверло с алмазным наконечником — везде используются датчики нагрузки и давления, хотя мы этого можем и не осознавать.

7.1. Типы датчиков нагрузки и давления

Перед тем как углубиться в эту тему, важно понять, что датчики нагрузки и датчики давления — это, по сути, идентичные устройства. Они оба измеряют нагрузку (или вес), и единственное различие между ними заключается в том, что выходной сигнал датчика давления масштабируется таким образом, чтобы приложенная нагрузка делилась на площадь контролируемой области. Например, одна из популярных разновидностей датчика давления поставляется в армированном цилиндрическом корпусе для использования в средах с высоким давлением. Выходной сигнал, снимаемый с этого датчика, отражает силу, приложенную к плоской части цилиндра (предполагая, что датчик установлен правильно). Зная диаметр датчика, приложение может в дальнейшем отмасштабировать замер с целью вычисления соответствующего давления.

Рассмотрим следующий примерю. Допустим, мы используем цилиндрический датчик, подобный упомянутому выше, и диаметр его цилиндра составляет 0,318 см (1/8 дюйма). Если датчик указывает, что

к нему приложена сила в 45,36 кг (100 фунтов), то соответствующее значение давления можно получить из уравнения:

$$\begin{aligned}\text{Давление} &= \text{Нагрузка} / \text{Область} = \\ &= \text{Нагрузка} / (\pi r^2) = \\ &= 45{,}36 \text{ кг} / (3{,}14 * 0{,}318 \text{ см})^2) = \\ &= 45{,}43 \text{ кг/см}^2.\end{aligned}$$

Если эту же силу в 45,36 кг приложить к датчику, диаметр которого будет равен половине диаметра первого датчика (т.е 0,1588 см или $^1/_{16}$ дюйма), то соответствующее давление составит

$$\text{Давление} = 45{,}36 \text{ кг} /(3{,}14 * 0{,}1588 \text{ см})^2) = 90{,}97 \text{ кг/см}^2.$$

Существует два типа датчиков, которые в настоящее время наиболее часто используются для измерения нагрузки и/или давления в приложениях мониторинга и управления: тензометры (датчики деформации) и пьезоэлектрические датчики. Хотя представленный далее материал не является исчерпывающим учебным курсом по вышеупомянутым типам датчиков, он содержит достаточно информации об их принципах работы и организации интерфейса с ними.

Тензометры

Говоря в двух словах, тензометр — это устройство, изменяющее свое сопротивление в зависимости от приложенной к нему силы. Этот эффект был впервые замечен Кельвином в 1856 году, однако, учитывая очень незначительное изменение сопротивления даже в случае больших изменений в нагрузке, практическое применение ему было найдено только в 1938 году. Любопытно, что точно замерять эти небольшие колебания сопротивления позволяет схема, предложенная английским физиком Чарльзом Уитстоном (Charles Wheatstone) еще в далеком в 1843 году (рис. 7.1).

В показанной схеме Ex– и Ex+ — это точки подключения к внешнему источнику питания, а SG– и SG+ — выводы, на которых измеряется дифференциальное напряжение. Такая организация резисторов формирует выходное напряжение, которое радиометрически[56] связано со значениями

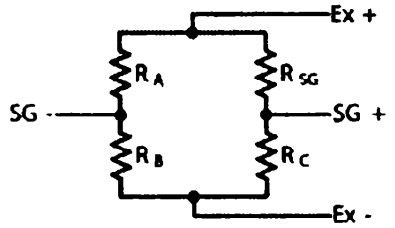


Рис. 7.1. Принципиальная схема моста Уитстона

[56] Рациометрическим называют сигнал, который отражает не абсолютное значение, а соотношение двух напряжений (прим. пер.).

сопротивлений резисторов в каждом плече моста. Если согласовать сопротивления в плече SG– (R_A и R_B) с соответствующими сопротивлениями в плече SG+ (R_{SG} и R_C), то на выходе получится нулевое напряжение. Возможно, в этой схеме нет ничего выдающегося, однако она позволяет легко отслеживать изменения в сопротивлении конкретного элемента, проявляющиеся в виде ненулевого выходного напряжения. Таким образом, установив в качестве одного из резисторов тензометр, можно точно измерять изменения в его сопротивлении, вызванные воздействием на него силы.

Тензодатчики используют тензометры, собранные по мостовой схеме, для измерения сил, действующих на конкретные механические оси. Термопары, рассмотренные в предыдущей главе, называют *пассивными чувствительными элементами*, поскольку изменения в контролируемом параметре (в данном случае — температуры) производят энергию выходного сигнала датчика. В отличие от термопар, тензометры — это *активные чувствительные элементы,* в которых изменения контролируемого параметра модулируют внешнее *напряжение возбуждения.* Это дает проектировщику дополнительную степень свободы, поскольку он может выбрать характеристики напряжения возбуждения, оптимизирующие параметрические измерения. С другой стороны, это требует дополнительных схем, обеспечивающих напряжение возбуждения. Кроме того, в систему выделяется дополнительное тепло, которое может ухудшить качество показаний.

Подобно термопарам, тензодатчки на тензометрах дают очень низкое выходное напряжение (порядка милливольт на вольт напряжения возбуждения). Это требует тщательного формирования сигнала, однако для расширения возможностей по измерению этих слабых сигналов мы можем использовать рациометрические характеристики выхода моста. Благодаря простоте применения и относительной дешевизне, тензодатчики на тензометрах, на сегодняшний день, — наиболее популярный тип тензодатчиков. Именно их мы и используем для примера, рассмотренного в данной главе.

Пьезоэлектрические датчики

На втором месте по популярности находятся пьезоэлектрические датчики нагрузки, которые используют свойство некоторых материалов формировать под действием силы крошечный электрический заряд.[57] Выходной сигнал таких датчиков чрезвычайно слаб (порядка нанокуло-

[57] Это свойство называется пьезоэлектрическим эффектом.

нов), поэтому для преобразования незначительных изменений заряда в напряжение, которое может быть измерено внешней системой, требуется специальная схема, называемая *усилителем заряда*.

Пьезоэлектрические датчики обычно используют в системах с динамической (т.е. регулярно изменяющейся) нагрузкой, поскольку они реагируют на быстрые изменения нагрузки лучше, чем тензодатчики на тензометрах. Впрочем, это не обязательно подразумевает более высокое качество замеров, если реакция контролируемой системы намного слабее реакции самого датчика. Кроме того, пьезоэлектрические датчики не отличаются точностью в ситуациях, когда нагрузка относительно постоянна, поскольку соответствующий выходной заряд склонен к затуханию со временем.

К сожалению, усилители заряда дорогостоящи и сложны в разработке, что сужает область применения пьезоэлектрических датчиков. Вместе с тем, в ситуациях, требующих максимальной частоты отклика, дополнительные расходы на использование пьезоэлектрических датчиков, могут быть оправданы требованиями задачи.

7.2. Ключевые аспекты измерения нагрузки

В общем случае, ключевые аспекты измерения нагрузки такие же как в любой сенсорной системе: диапазон измерений, разрешающая способность и точность. Хотя измерение нагрузки связано с определенными трудностями, и ей присущи уникальные характеристики сигнала, другие аспекты, наподобие линеаризации и калибровки, — общие для всех сенсорных систем.

Диапазон измерений

Вследствие того, что приложения контроля нагрузки могут работать в большом диапазоне значений (от микрограмм до килотонн), разработчик должен определить диапазон для конкретного случая. Здесь есть один важный аспект, который многие инженеры упускают из виду (по крайней мере, в первый раз): необходимость выяснить как нормальный рабочий диапазон, так и динамический диапазон для наихудшего случая. Например, система для взвешивания тюков, рассчитанная на работу в диапазоне 0 до 90 кг, может столкнуться со случаем, когда тюк случайно упадет на весы с небольшой высоты. В результате будет приложена мгновенная сила, превосходящая номинальную нагрузку в несколько раз. И учитывать необходимо не только механические факторы (система должна быть способной выдержать удар без физических по-

вреждений), но и обеспечить формирование сигнала, которое из-за мгновенного перенапряжения не повредило бы какую-нибудь схему.

В качестве примера разработаем систему контроля давления с номинальным рабочим диапазоном 0 – 9 000 кг/дюйм^2 и максимально допустимой нагрузкой до 13 600 кг/дюйм^2. Это был бы приемлемый диапазон для измерения пустотного давления (давление внутри расплава) в формовочной машине. Наша задача — обеспечить самую высокую точность в рабочем диапазоне, поэтому в качестве точек калибровки воспользуемся значениями 0 и 9 000.

Разрешающая способность измерений

Предполагая, что мы усиливаем входной сигнал, для которого минимальное и максимальное значения параметра соответствуют AV_{SS} и AV_{DD}, разрешающая способность измерений будет получена путем деления максимальной нагрузки на число доступных уровней АЦП (1 024 для 10-разрядного АЦП, 4 096 уровней для 12-разрядного АЦП и 65 536 уровней для 16-разрядного АЦП). В данном случае в нашем распоряжении встроенный 12-разрядный АЦП dsPIC30F6014A, что дает нам 4 096 уровней. Принимая во внимание диапазон измерений до 13 600 кг/дюйм^2, разрешающая способность в нашем случае составит около 3,3 кг/дюйм^2/уровень (13 600 кг/дюйм^2 / 4 096 уровней).

Если требуется более высокая точность, то необходимо воспользоваться внешним АЦП с большей разрешающей способностью. Однако в большинстве случаев, связанных с литьем пластмасс, 12-разрядного АЦП вполне достаточно.

Точность измерений

Еще один важный аспект — точность измерений. Она зависит от всей цепочки прохождения сигнала от датчика до АЦП и требует особого внимания к аналоговой схеме формирования сигнала, к генератору напряжения возбуждения, к точности АЦП, а также — к любой деградации сигналов из-за колебаний температуры. Погрешность в цепочке прохождения сигналов накапливается, поэтому важно учесть каждое звено. В качестве отправной точки отметим, что тензодатчики мостового типа обеспечивают точность примерно 0,1%, хотя эта величина может изменяться в зависимости от выбранной модели датчика.

Специфика

Специфика применения тензодатчиков мостового типа аналогична специфике обработки сигналов от термопар: низковольтная природа

выходного сигнала (по крайней мере, при низких нагрузках) и снижение точности датчика при колебаниях температуры.

Характеристики сигналов

Мы уже вскользь затронули характеристики сигналов от тензодатчиков, отметив, что эти сигналы являются относительно низковольтными (от нескольких милливольт до нескольких сотен милливольт). На диапазон сигнала влияет уровень напряжения возбуждения, однако разработчику обычно приходится придерживаться баланса между преимуществами более высокого напряжения возбуждения, расширяющего диапазон выходного сигнала, с недостатками эффекта самонагревания.

Хотя частотная характеристика для каждого типа тензодатчиков специфична, в общем случае можно сказать, что частотный спектр измеряемых сигналов составляет от десятков до сотен герц. В случае консервативного соотношения частоты дискретизации и частотного спектра "пять к одному", разработчик может рассчитывать на частоту дискретизации от 100 Гц до 1 кГц или выше, в зависимости от приложения. Если в течение периода измерения нагрузка не изменяется, то частота дискретизации, конечно же, может быть значительно ниже.

Термокомпенсация

Эффект самонагревания наиболее ярко проявляется для больших значений напряжения возбуждения, когда ограничены возможности рассеяния тепла (например, в случае герметичного корпуса). Поскольку мы используем низкое напряжение возбуждения (5 В), то проблем с самонагреванием возникнуть не должно. В тех же случаях, когда вопрос самонагревании актуален, его можно минимизировать, организовав пульсацию напряжения возбуждения с малым коэффициентом заполнения. При этом дискретизация сигнала должна быть синхронизирована с рабочим циклом напряжения возбуждения таким образом, чтобы выборка производилась только в те моменты, когда напряжение возбуждения "включено".

Эту методику можно также использовать для того, чтобы обеспечить возможность применения более высокого напряжения возбуждения (например, 10 В). Однако при этом следует удостовериться в том, что это напряжение не повредит другие схемы. Например, поскольку напряжение возбуждения используется в качестве опорного напряжения для АЦП dsPIC, то его значение ограничено электрическими спецификациями опорного напряжения для ЦКС dsPIC, хотя выходной сигнал самого датчика был бы приемлем и находился в пределах допустимого

диапазона входного сигнала АЦП и при более высоком напряжении возбуждения.

Линеаризация

В отличие от датчиков на основе термопары, для тензодатчиков на тензометрах линеаризация особой роли не играет, поскольку выходной сигнал моста в рабочем диапазоне нагрузок тензодатчика достаточно линеен. Это значительно упрощает задачу разработчика, поскольку можно обойтись вообще без каких-либо аппаратных или программных средств линеаризации.

Калибровка

Калибровка тензодатчиков на тензометрах может оказаться физически затруднительной. В некоторых случаях (особенно связанных со взвешиванием) разработчик может предположить, что сила нагрузки перпендикулярна земле. Предполагая, что максимальная нагрузка "приемлема" (что является очень размытым определением), сенсорную систему можно откалибровать с помощью двух эталонных грузов, вес которых находятся вблизи верхнего и нижнего пределов ожидаемого рабочего диапазона. Система зарегистрировала бы измеренное значение выходного напряжения датчика в каждой точке калибровки, а затем вычислила бы коэффициент усиления и смещение линейной характеристики, как в случае с системой на термопаре.

К сожалению, обычно откалибровать систему так просто не удается из-за "неприемлемости" нагрузок (например, практически нереально использовать в качестве точек калибровки значения 40 и 22 000 кг) или препятствий со стороны физических характеристик контролируемой системы. Например, в формовочных машинах датчики, измеряющие давление пластмассового расплава внутри формы, зачастую смонтированы вертикально. Невозможно (или, по крайней мере, очень затруднительно) приложить к таким датчикам калибровочную физическую нагрузку в 9 000 кг/дюйм^2, поэтому в таких случаях необходимо искать альтернативные подходы.

В подобных ситуациях к выходам тензодатчика обычно подключают известное сопротивление, а полученное выходное напряжение используют в качестве верхней точки калибровки (нижняя точка будет определяться ненагруженным выходным сигналом датчика). Хотя этот подход может показаться грубым, на практике он довольно успешен и зачастую — единственный возможный, нравится нам это или нет.

Источники электрического шума

Шум может поникать в выходной сигнал тензодатчика от нескольких различных источников. Так, шум от электросети может наводиться в проводниках подачи в датчик напряжения возбуждения и /или в проводниках выходного сигнала. Предполагая, что упомянутые проводники расположены близко друг к другу (в идеале, это должна быть витая пара), результирующий шум будет носить синфазный характер и, следовательно, может быть в основном удален, благодаря дифференциальному усилителю входного сигнала и рациометрической природе измерений с помощью АЦП. Это важно, поскольку сигнал электросети на 50 или 60 Гц может находиться прямо посреди допустимого частотного спектра измеряемого сигнала.

Если "врожденные" шумоподавляющие характеристики схемы не устраняют сетевой шум в достаточной мере, разработчик всегда может воспользоваться цифровым узкополосным режекторным фильтром с провалом спектра, отцентрированным на отметке в 50 или 60 Гц (в зависимости от частоты сети электропитания). Хотя это усложняет программное обеспечение и несколько ухудшает характеристики измеряемого сигнала, такая фильтрация может стать необходимой составляющей приложения.

Сбойные ситуации

Тензодатчики могут оказываться в различных сбойных ситуациях:

- разрыв проводника, ведущего к датчику или от него;
- повреждение элемента моста в датчике;
- реверсирование проводников напряжения возбуждения;
- реверсирование проводников выходного напряжения.

Все эти ситуации могут быть обнаружены путем замера сопротивлений на различных проводниках, ведущих к датчику или от него. Однако, если возложить эти функции на саму систему, то потребуется дополнительная схема, и уменьшится количество каналов АЦП, доступных для выходных сигналов от тензодатчиков, поскольку некоторые из этих каналов должны быть задействованы при диагностике.

Из перечисленных выше ситуаций три первые являются "фатальными" в том смысле, что система при их наличии не может функционировать. В случае же обнаружения четвертой ситуации (реверсированные проводники выходного напряжения) систему можно восстановить.

7.3. Разработка приложения

Мы разработаем довольно простые весы, использующие тензодатчик для точного измерения веса в диапазоне от 0 до 50 кг. Для упрощения калибровки воспользуемся подключением резистора известного номинала, формирующим верхнюю точку калибровки.

Спецификация системы

В качестве аппаратной платформы система использует плату Microchip dsPICDEM v1.1. При этом решаются следующие задачи:

- измерение веса между 0 и 50 кг с разрешением в 0,25 кг;
- максимальная нагрузка — 64 кг;
- дискретизация одного канала нагрузки с частотой 1 кГц (принимается, что максимальный частотный спектр составляет 200 Гц);
- фильтрация дискретных данных с целью удаления сетевого шума на частоте 60 Гц;
- предоставление пользователю возможности выполнять через последовательный порт RS-232 (38,4 Кбит/с, восемь информационных бит, один стоп-битом, без контроля четности и управления потоком данных) следующие функции:
  а) калибровать систему;
  б) задавать верхний и нижний пределы для измеряемого веса;
- ежесекундное уведомление об измеренном весе через последовательный порт (сообщения должны быть в текстовом виде);
- уведомление о выходе измеряемого веса за пределы допустимого диапазона путем включения светодиода LED1 макетной платы, если вес меньше нижнего предела, и светодиода LED2, если вес превысил верхний допустимый предел.

Как и в случае системы температурного контроля, если информация должна считываться не человеком, а другими электронными компонентами системы, то следует использовать более скоростной бинарный протокол обмена данными.



Программный код реализации этого протокола находится на прилагаемом к книге компакт-диске в файлах `BinaryProtocol.c` и `BinaryProtocolDef.h`.

Формирование сигналов от датчика

На рис. 7.2 показана структурная схема интерфейса тензодатчика, содержащая измерительный усилитель с коэффициентом усиления, обеспечивающим формирование выходного сигнала тензодатчика, ис-

точник напряжения возбуждения (в данном случае — системный блок питания на 5 В), и пару каскадированных фильтров Баттерворта второго порядка, предназначенных для ограничения частотного спектра усиленного сигнала перед его дискретизацией в АЦП. Кроме того, интерфейс включает реле, позволяющее процессору подключать в качестве калибровочного шунта резистор известного номинала.

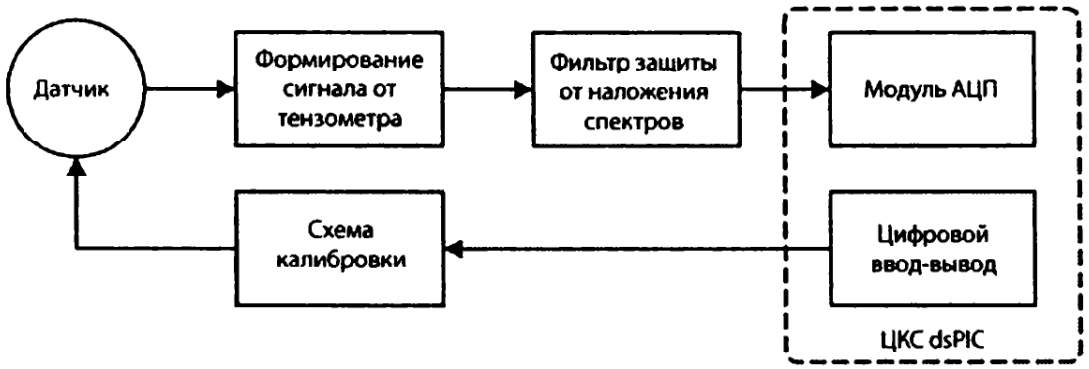


***Рис. 7.2.** Структурная схема интерфейса тензодатчика (для одного канала)*

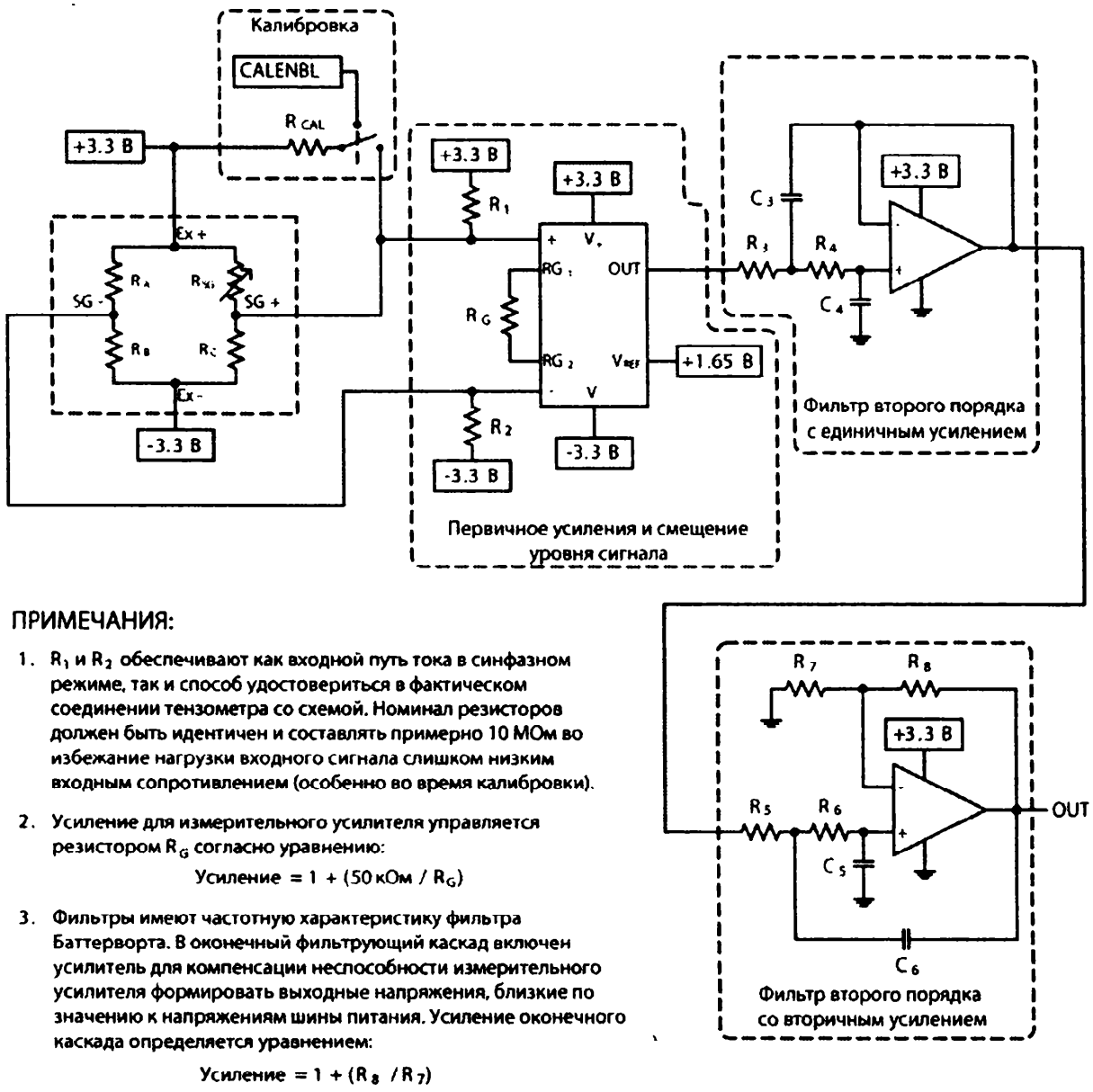
Обратите внимание на возможность использовать ту же схему дифференциального усиления (хотя и с другим коэффициентом усиления) и ту же топологию фильтра Баттерворта (хотя и с другими значениями компонент для расширения полосы пропускания), что и при формировании сигналов от термопары. Соответствующая принципиальная схема показана на рис. 7.3. Как и в случае с примером измерения температуры, схема рассчитана на питание 3,3 В. Для макетной платы dsPICDEM 1.1. все опорные напряжения необходимо заменить следующим образом: 3,3 В — на 5 В; –3,3 В — на –5 В; а 1,65 В — на 2,5 В.

Анализ цифровой фильтрации

Поскольку предполагается, что шум от сети электропитания находится в одной полосе частот с измеряемым сигналом, для его устранения придется воспользоваться узкополосным режекторным фильтром. Его частотная характеристика показана на рис. 7.4.

Алгоритмы анализа данных

Алгоритмы анализа данных чрезвычайно просты и, за исключением незначительных изменений, аналогичны алгоритмам, которые использовались при контроле температур. Они просто проверяют каждую отфильтрованную выборку данных, чтобы определить, не выходит ли значение за верхний или нижний допустимый предел. В случае выхода за допустимые пределы включается соответствующий светодиод.



ПРИМЕЧАНИЯ:

1. R_1 и R_2 обеспечивают как входной путь тока в синфазном режиме, так и способ удостовериться в фактическом соединении тензометра со схемой. Номинал резисторов должен быть идентичен и составлять примерно 10 МОм во избежание нагрузки входного сигнала слишком низким входным сопротивлением (особенно во время калибровки).
2. Усиление для измерительного усилителя управляется резистором R_G согласно уравнению:

   Усиление = 1 + (50 кОм / R_G)

3. Фильтры имеют частотную характеристику фильтра Баттерворта. В оконечный фильтрующий каскад включен усилитель для компенсации неспособности измерительного усилителя формировать выходные напряжения, близкие по значению к напряжениям шины питания. Усиление оконечного каскада определяется уравнением:

   Усиление = 1 + (R_8 / R_7)

Рис. 7.3. *Принципиальная схема интерфейса тензодатчика*

7.4. Микропрограммная реализация

Как и в случае с системой контроля температуры, программа, реализующая дискретизацию, фильтрацию и анализ сигнала, а также — обработку ошибок в рабочем диапазоне взвешивания, интенсивно использует 16-разрядную библиотеку функций для работы с периферийными устройствами от компании Microchip, а также библиотеку ЦОС. Это позволяет создать согласованный, оттестированный набор функций цифровой обработки сигналов для фильтрации.

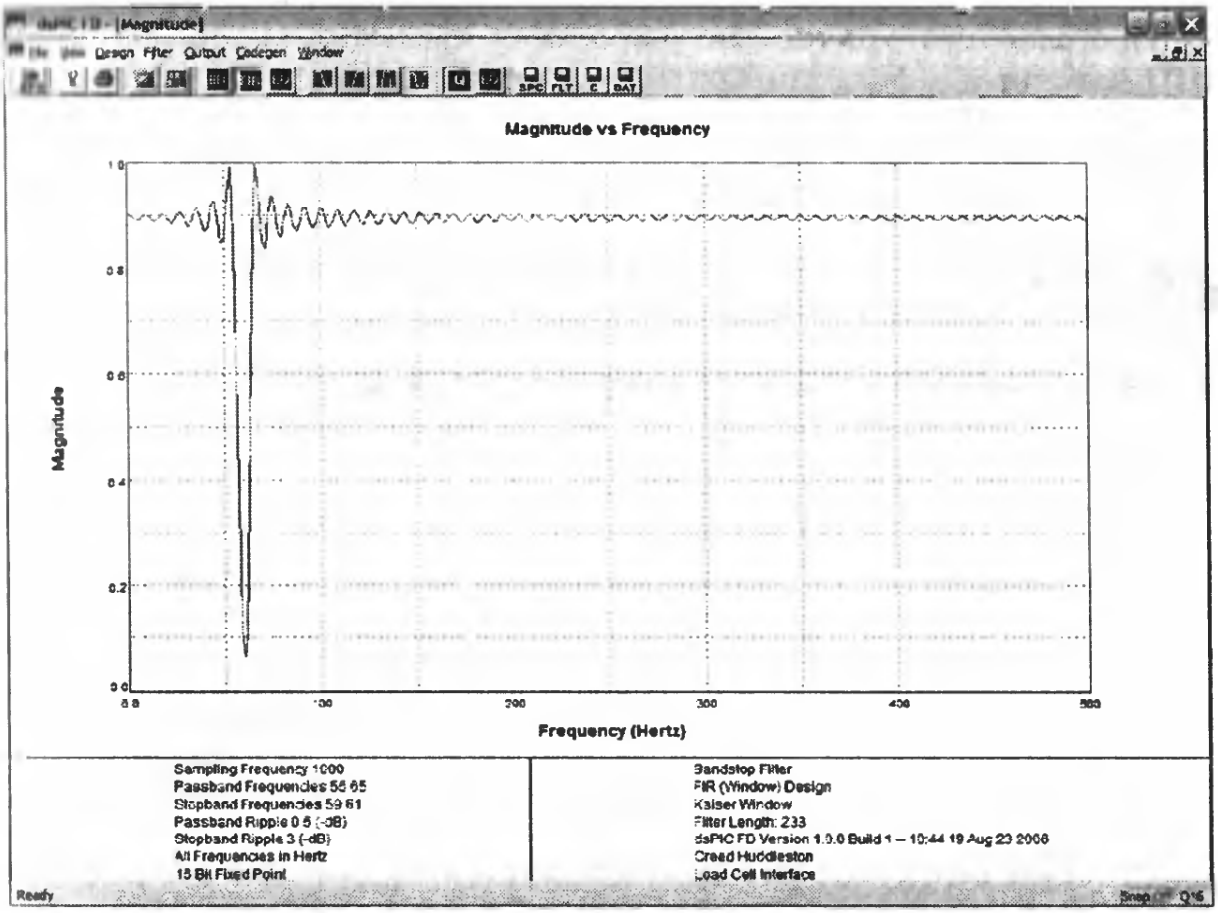


Рис. 7.4. Частотная характеристика узкополосного режекторного фильтра, устраняющего сетевой электрический шум

Дискретизация сигнала

Сигнал тензодатчика на аналоговом входе AN7 квантуется каждую миллисекунду в режиме автоматической дискретизации с применением таймера 3 dsPIC, обеспечивающего синхронизацию преобразований. Для максимизации разрешающей способности предположим, что входной сигнал униполярен (т.е. пользователь должен гарантировать, что напряжение возбуждения не будет реверсировано) и имеет диапазон от 0 до 5 В. Благодаря этому, мы можем использовать для выходных данных АЦП беззнаковое представление 1Q15. Кроме того, для снижения вычислительной нагрузки, обусловленной высокой частотой дискретизации, мы будем инициировать прерывание только после добавления к буферу каждой восьмой выборки. Это число может быть уменьшено для повышения скорости реакции системы или же увеличено для дальнейшего снижения загрузки процессора, связанной с обработкой прерываний.

Программный код конфигурирования параметров дискретизации АЦП находится в функции `ADCInit()` модуля `ADCIF.c`. При этом для настройки АЦП и таймера 3 используются функции `OpenADC12()` и `OpenTimer3()` из 16-разрядной библиотеки обслуживания периферийных устройств (листинги 7.1 и 7.2).



Файлы `ADCIF.c` и `Timer.c` находятся на прилагаемом к книге компакт-диске в папке `Коды\Тензодатчик`.

Листинг 7.1. Функция `ADCInit()` из модуля `ADCIF.c`

```
Uint8 ADCInit(void)
{
   //Локальные переменные
   Uint16
      ui16ADCON1,  // Данные конфигурации для регистра ADCONI
      ui16ADCON2,  // Данные конфигурации для регистра ADCON2
      ui16ADCON3,  // Данные конфигурации для регистра ADCON3
      ui16ADPinConfig,  // Данные конфигурации для ADPCFG
                        // (конфигурирование выводов)
      ui16ScanSelect;   // Данные конфигурации для ADCSSL
                        // (выбор канала для опроса)

   // Отключение модуля АЦП и блокировка прерываний от АЦП,
   // чтобы гарантировать завершение конфигурирования без
   // прерываний
   CloseADC12();

   // Установка данных по конфигурированию регистра ADCONI,
   // наиболее важными из которых является формат данных,
   // источник синхронизации и установка режима
   // автоматической дискретизации
   ui16ADCON1 = ADC_MODULE_ON     & // Включение модуля АЦП
                ADC_IDLE_STOP     & // Останов преобразований
                                    // в режиме ожидания
                ADC_FORMAT_FRACT  & // Беззнаковые дробные
                                    // данные
                ADC_CLK_TMR       & // Использовать таймер 3
                                    // для управления
                                    // преобразованием
                ADC_AUTO_SAMPLING_ON; // Активизировать режим
                                      // автоматической дискретизации

   // Установка данных по конфигурированию регистра ADCON2,
   // включая опорные напряжения АЦП, количество выборок
   // между прерываниями и задание порядка данных в буфере
```

Листинг 7.1. Продолжение

```
    ui16ADCON2 = ADC_VREF_AVDD_AVSS     & // Использовать
              // в качестве опорных напряжений АЦП AVdd и AVss
                 ADC_SCAN_ON             & // Активизация
                                         // режима опроса канала
                 ADC_SAMPLES_PER_INT_8 & // 8 выборок перед
                                           // прерыванием
                 ADC_ALT_BUF_OFF         & // Не менять буфер
                 ADC_ALT_INPUT_OFF;      & // Не чередовать вход

    // Установка данных по конфигурированию регистра ADCON3

    ui16ADCON3 = ADC_SAMPLE_TIME_31  & // Выборка на 31 Tad
                 ADC_CONV_CLK_SYSTEM & // Использование для
                                     // преобразований системного
                                     // тактового генератора
                 ADC_CONV_CLK_32Tcy;   // Разрешить 32 Tcy
                                       // для преобразования

    // Установка данных по конфигурированию регистра ADCSSL,
    // задающие каналы АЦП для опроса. Обратите внимание, что
    // ввилу способа реализации библиотек мы определяем в этом
    // списке сигналы ANx, которые НЕ собираемся опрашивать.
    // В данном случае мы опускаем сигнал AN7, поскольку он -
    // единственный, задействованный при дискретизации.

    ui16ScanSelect = SKIP_SCAN_AN0  & SKIP_SCAN_AN1  &
                     SKIP_SCAN_AN2  & SKIP_SCAN_AN3  &
                     SKIP_SCAN_AN4  & SKIP_SCAN_AN5  &
                     SKIP_SCAN_AN6  & SKIP_SCAN_AN8  &
                     SKIP_SCAN_AN9  & SKIP_SCAN_AN10 &
                     SKIP_SCAN_AN11 & SKIP_SCAN_AN12 &
                     SKIP_SCAN_AN13 & SKIP_SCAN_AN14 &
                     SKIP_SCAN_AN15;

    // Ганантируем, что выводы, связанные с используемыми
    // аналоговыми каналами, сконфигурированы как аналоговые
    // входы. Приложение использует вывод AN7 в качестве
    // сигнального входа. Плата dsPICDEM ставит в соответствие
    // выводы AN0 и AN1 интерфейсу ICD2, AN3 - входу цифрового
    // потенциометра, AN4-AN6 - входам аналогового
    // потенциометра, а AN8 - входу датчика температуры.

    ui16ADPinConfig = ENABLE_AN3_ANA & ENABLE_AN4_ANA &
                      ENABLE_AN5_ANA & ENABLE_AN6_ANA &
                      ENABLE_AN8_ANA;
```

Листинг 7.1. Окончание

```
    // Фактическое конфигурирование модуля АЦП. Обратите
    // внимание на то, что этот код активирует модуль АЦП,
    // но соответствующее прерывание должно быть разрешено
    // вызовом функции EnableIntADC() после того, как будет
    // инициализирована остальная часть системы.

    OpenADC12(ui16ADCON1, ui16ADCON2, ui16ADCON3,
              ui16ADPinConfig, ui16ScanSelect);

    // Инициализируем таймер 3 на генерирование прерывание
    // через 1 мс для выполнения АЦП-выборки

    Time3Init(ADC_SAMPLE_RATE);

    return ST_OK;
}
```

Листинг 7.2. Фрагмент из файла `Timer.c`

```
void Timer3Init(Uint16 ui16SampleRate)
{
    // Локальные переменные
    Uint16
       ui16Period,     // Период таймера в отсчетах
       ui16TimerCfg;   // Данные для конфигурирования таймера

    // Вначале отключаем таймер 3 и запрещаем соответствующее
    // прерывание, чтобы завершить инициализацию

    CloseTimer3();

    // Вычисляем период таймера в отсчетах на основании
    // частоты системного тактового генератора (вычисляется
    // путем умножения частоты кварцевого генератора FOSC на
    // коэффициент масштабирования схемы фазовой синхронизации
    // PLL с последующим делением на 4, поскольку на
    // выполнение каждой команды уходит четыре тактовых цикла)

    ui16Period = (Uint16)(((FOSC*PLL/4)/ui16SampleRate) + 1);

    // Конфигурируем сам таймер
    ui16TimerCfg = T3_ON          &  // Включаем таймер 3
                   T3_IDLE_STOP   &  // Переводим таймер 3
                                     // в состояние ожидания
                   T3_GATE_OFF    &  // Нет стробирования
                   T3_PS_1_1      &  // Установка делителя 1:1
```

Листинг 7.2. Окончание

```
                    T3_SOURCE_INT;  // Для таймера используем
                                    // внутренний тактовый генератор

    // Конфигурируем таймер
    OpenTimer3(ui16TimerCfg, ui16Period);

    return;
}
```

Обработчик прерывания — это подпрограмма ADCISR() (листинг 7.3), также расположенная в модуле ADCIF.c. Она сбрасывает флаг прерывания АЦП. Если этого не сделать, то по завершению функции процессор передаст управление по вектору обратно в обработчик прерывания, что введет приложение в бесконечный цикл постоянного обслуживания прерывания.

Листинг 7.3. Программный код обработчика прерываний от АЦП

```
void __attribute__ ((__interrupt__)) ADCISR(void)
{
    // Локальные переменные

    fractional volatile
        *pfrADCBuff,        // Указатель на буфер данных АЦП
        *pfrInputSignal;    // Указатель на глобальный буфер
                            // данных входных сигналов
    Uint8 ui8DataCount;     // Индекс данных АЦП

    IFS0bits.ADIF = 0;      // Сбрасываем флаг прерывания от АЦП

    // Копируем результаты аналого-цифрового преобразования
    // в буфер g_frSensorSignal[]. Обратите внимание на то,
    // что выборки по каждому каналу непрерывны, поэтому их
    // следует размещать в отдельных сигнальных массивах,
    // подлежащих фильтрации. Здесь мы этого не делаем, чтобы
    // минимизировать время обслуживания прерывания.

    pfrADCBuff = &ADCBUF0;  // Указывает на первую позицию
                            // в буфере данных АЦП
    pfrInputSignal= g_frInputSignal; // Указывает на первую
                    // позицию в буфере данных входного сигнала

    for (ui8DataCount = 0;
         ui8DataCount < ADC_SAMPLE_COUNT;
         ui8DataCount++)
        *pfrInputSignal++ = *pfrADCBuff++;  // Копируем
  // следующую выборку АЦП в глобальный буфер входных сигналов
```

Листинг 7.3. Окончание

```
    // Устанавливаем флаг события фильтрации для уведомления
    // главного цикла обработки о наличии новых данных для
    // фильтрации

    g_vui16SysEvent |= EVT_FILTER;
}
```

Реализация цифрового фильтра

Реализация цифрового фильтра почти идентична реализации для датчика температуры, рассмотренного в предыдущей главе. Единственное отличие связано с коэффициентами фильтра. Как и в случае датчика температуры, в данном примере используется КИХ-фильтр, требующий для поддержки связанных с фильтром переменных состояния, чтобы в приложении была создана и инициализирована структура `FIRStruct`. Фильтрация новых данных — это лишь вопрос вызова функции `FIR()` из библиотеки ЦОС с новой выборкой данных в качестве параметра.

В данном приложении мы будем использовать заграждающий фильтр со следующими параметрами:

- частота дискретизации — 1 000 Гц;
- диапазон частот в полосе пропускания — 5 565 Гц;
- диапазон частот в полосе заграждения — 5 961 Гц;
- пульсация в полосе пропускания — 0,5 дБ;
- пульсация в полосе заграждения — 3 дБ;
- тип фильтра — Кайзера.

Ввод этих параметров в программе dsPIC Filter Design и генерация кода для результирующего фильтра (файл `SensorFilter.s`) дает требуемую структуру данных фильтрации. Обратите внимание на то, что в данном случае фильтр будет намного длиннее, чем для примера с термопарой (233 отвода по сравнению с 51), поскольку требования к фильтрации теперь — намного строже.

Реализация анализа данных

Алгоритм анализа данных реализован в функции `AnalyzeData()` (листинг 7.4) из модуля `Analysis.c`. Эта простая функция последовательно исследует отфильтрованные данные и переключает соответствующие светодиоды, если сигнал оказывается вне пределов заданного диапазона или возвращается в этот диапазон.

Листинг 7.4. Программный код анализа данных

```
Uint16 AnalyzeData(void)
{
   // Локальные переменные
   float fSensorValue; // Текущее отмасштабированное значение
                       // сигнала от датчика
   PSensorCfg psenSensorCfg; // Указатель на конфигурацию
                             // датчика
   Uint16 ui16SensorIndex;   // Индекс датчика, начиная с 0

   // Проверяем датчики по одному, чтобы выяснить, не
   // превышают ли текущие значения сигнала установленные
   // допустимые пределы
   psenSensorCfg = g_senSensorCfg;  // Указывает на первую
                                    // конфигурацию датчика
   for (ui16SensorIndex = 0;
        ui16SensorIndex < MAX_CAL_SENSORS;
        ui16SensorIndex++, psenSensorCfg++)
   {
      // Преобразуем текущее отфильтрованное значение сигнала
      // от датчика в соответствующее значение параметра
      fSensorValue =
         ScaleData(g_frFilteredSensor[ui16SensorIndex],
         psenSensorCfg->fGain,
         psenSensorCfg->fOffset + 0.0005);
      g_fSensorValue[ui16SensorIndex] = fSensorValue;

      // Контролируется ли нижний допустимый предел?
      // Если да, то нарушает ли отфильтрованное значение
      // сигнала от датчика нижний допустимый предел?
      if ((psenSensorCfg->ui16Flags & SENFLG_ENBL_LOW_ALM) &&
     (fSensorValue < psenSensorCfg->fAlarmLevel[ALARM_LOWER]))
         psenSensorCfg->ui16Flags |= SENFLG_LOW_ALM_ST;
       // Выход за нижний предел, поэтому устанавливаем флаг

      // Контролируется ли верхний допустимый предел?
      // Если да, то нарушает ли отфильтрованное значение
      // сигнала от датчика верхний допустимый предел?
      if ((psenSensorCfg->ui16Flags & SENFLG_ENBL_HI_ALM) &&
     (fSensorValue > psenSensorCfg->fAlarmLevel[ALARM_UPPER]))
         psenSensorCfg->ui16Flags |= SENFLG_HI_ALM_ST;
       // Выход за верхний предел, поэтому устанавливаем флаг
   }
   return ST_OK;
}
```

Реализация обработки ошибок

Обработку ошибок в данном приложении можно реализовать относительно просто, а можно и усложнить, если требуется вызывать функции тестирования сопротивления каждого из плеч моста. В этом случае выводы мостовых соединений тензодатчика подключают к отдельным аналоговым входам, что значительно повышает сложность схемы и снижает число каналов, которые можно контролироваться с помощью внутренних ресурсов процессора одного ЦКС dsPIC.

Программный код обработки ошибок находится в функции `CheckForErrors()` в файле `Analysis.c`, (см. папку `Коды\Тензодатчик` на прилагаемом к книге компакт-диске). Он выполняет контроль на наличие закороченных или разорванных мостовых соединений — состояний, которые обнаруживаются по слишком низкому или почти максимальному значению сигнала. Если обнаружено одно из таких состояний, то устанавливается соответствующий флаг, что приводит к мерцанию светодиода LED или LED 2. Как только ошибочное состояние исчезает, приложение отключает светодиод.

7.5. Резюме

Завершив проектирование двух приложений, использующих АЦП dsPIC и цифровую фильтрацию, мы переходим к разработке третьего приложения, собирающего данные с применением совершенно иного подхода: подсчета количества переключений сигнала на входе внешнего источника тактирования таймера/счетчика с целью определения интенсивности потока жидкости через турбину.

8

Реальный пример: датчики потока

Процесс нельзя понять, остановив его. Понимание должно двигаться вместе с потоком, став с ним одним целым.
— Фрэнк Герберт, "Дюна"

В завершение обсуждения интерфейсов популярных датчиков, рассмотрим датчики потока (или расхода), позволяющие измерить поток вещества (обычно — жидкости) через систему. Хотя существует несколько подходов к определению расхода вещества, включая *массовый расход*, *объемный расход*, *ламинарный поток* и *турбулентный поток*, общая суть заключается в измерении количества проходящего вещества для решения конкретной задачи. Это обычно означает, что разработчика системы интересует, прежде всего, *массовый расход* (т.е. масса, проходящая через конкретную точку системы за единицу времени). Однако в случае с жидкостями, когда состояние и плотность вещества по сути постоянны (по крайней мере, в интересующей точке системы), разработчик может вместо массового расхода измерять *объемный расход* (объем вещества в единицу времени) без ухудшения характеристик системы. Такой подход — наиболее распространенный, и именно его мы рассмотрим в нашем последнем примере.

Как можно привести примеры измерения потока? Для большинства людей два самых очевидных и прозаичных примера — это счета за электричество и воду, составленные на основании измерений расхода упомянутых субстанций. Датчики потока применяют как в огромных системах типа нефтепроводов, так и в совсем небольших, наподобие систем мониторинга дыхания младенцев в отделении интенсивной терапии роддома. Все эти датчики отвечают за управление критически важными функциональными параметрами. Не будет преувеличением утверждение, что вся торговля по сути базируется на использовании надежных систем измерения потока, обеспечивающих точную поставку тысяч видов топлива, продовольствия и других товаров по всему миру.

8.1. Типы датчиков потока

Хотя мы опишем два популярных типа датчиков потока: турбинные (измеряют объемный расход) и *гравиметрические* (измеряют массовый расход), — система в нашем примере будет построена на турбинном датчике. И на то есть две причины, и первая из них заключается в широком распространении турбинных датчиков, вследствие чего разработчику крайне важно понимать принцип их работы. Вторая причина (пожалуй, более весомая) — формирование турбинным датчиком выходного сигнала, с которым мы еще не сталкивались в предыдущих двух приложениях. Речь идет о выходном сигнале, частота которого прямо пропорциональна потоку (т.е. чем больше поток, тем выше частота). Как мы увидим, этот новый тип выходного сигнала имеет как преимущества, так и недостатки.

Турбинные датчики

Турбинный датчик, который обычно используют при работе с жидкостями или газами, выглядит как небольшой вентилятор или пропеллер с осью, ориентированной по направлению потока, и лопастями, перекрывающими все поперечное сечение протока (рис. 8.1).

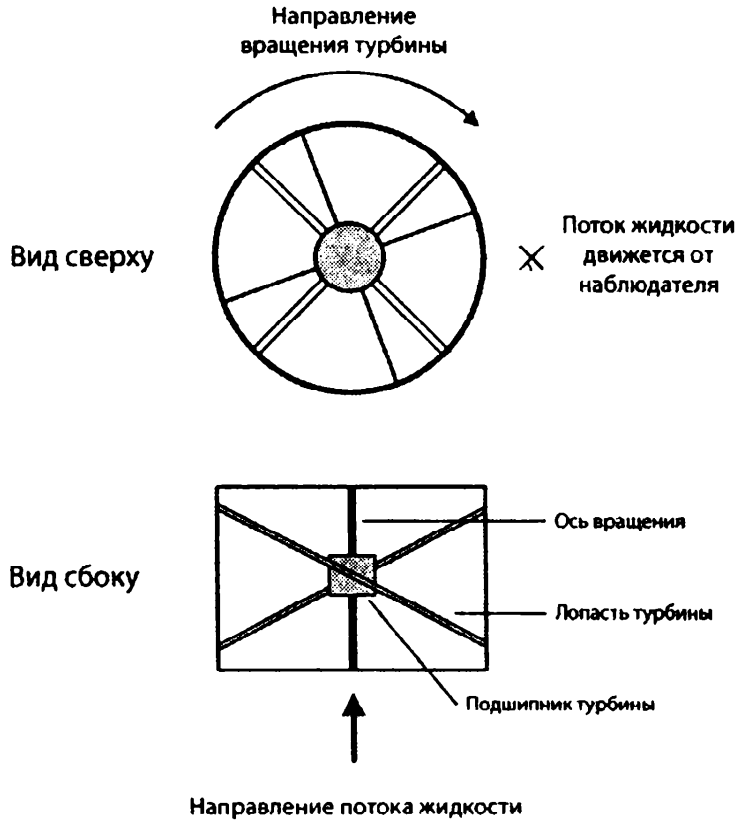


Рис. 8.1. *Пример турбинного датчика потока*

Такое расположение лопастей не только обеспечивает максимально возможное давление потока на датчик (это очень важно в случае слабого потока), но также гарантирует, что вещество при прохождении датчика будет встречать постоянное сопротивление (это предохраняет систему от возможного физического повреждения в случае очень интенсивного потока).

Поскольку ось вентилятора находится непосредственно внутри потока, измерить скорость осевого вращения напрямую может оказаться непросто. По этой причине турбинные датчики обычно оснащены окошками, позволяющими наблюдать за лопастями сбоку. Подсчитав количество прохождений лопастью фиксированной точки в окошке за заданный период времени, и зная плотность материала и площадь поперечного сечения протока, сенсорная система может определить объем жидкости, прошедшей через датчик.

На первый взгляд подсчет числа вращений лопастей может показаться устрашающей задачей, если учесть их электрическую изолированность от остальной части системы. К счастью, во многих случаях мы можем просто освещать окошко источником, который излучает свет на частоте, хорошо отражаемой лопастями (но не проходящим материалом и самим окошком датчика), а затем — измерить отраженный сигнал. В результате выходной сигнал датчика усиливается и ослабевает по мере прохождения лопастей точки подсветки (пример — на рис. 8.2).

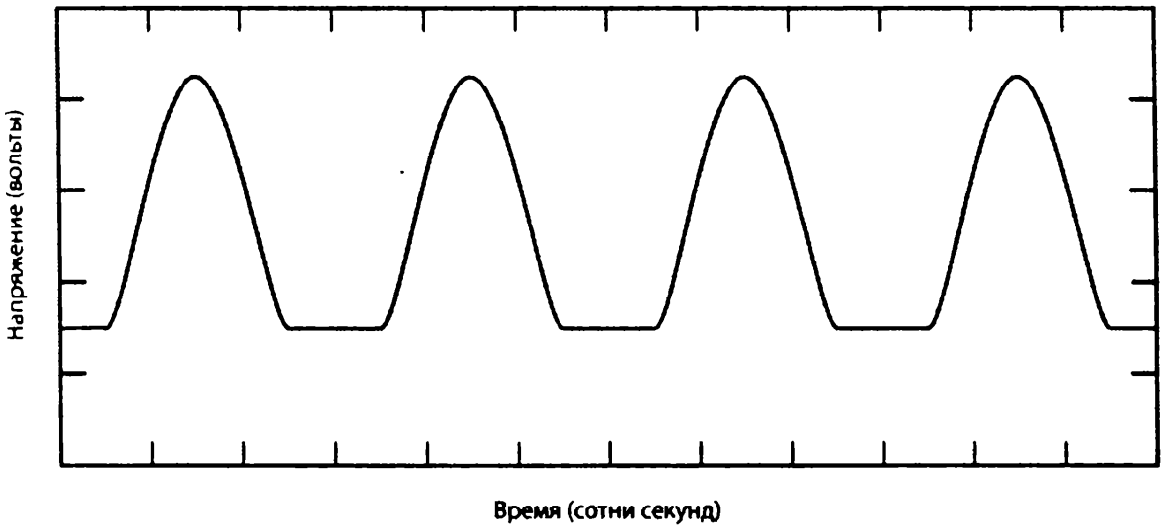


Рис. 8.2. Пример выходного сигнала турбинного датчика расхода

Подсчитав пики сигнала за заданный период времени, мы можем определить скорость вращения лопастей датчика. Более подробно об особенностях такого подхода речь пойдет в подразделе "Специфика измерений потока".

В отличие от датчиков температуры и тензодатчиков, преимущественно электрических по своей природе, расчет серьезной системы с турбинным датчиком потока требует от разработчика познаний не только в электронике, но и в механике. Хотя в этой главе основное внимание уделено электрическим аспектам интерфейса турбинного датчика, читатель должен понимать, что механические аспекты контроля потока не менее важны, а эффективная реализация системы невозможна без понимания того, каким образом установка датчика (или датчиков) может повлиять на контролируемую систему.

Гравиметрические датчики

В отличие от турбинных датчиков потока, которые измеряют объем вещества, прошедшего через область поперечного сечения протока за определенный период времени, гравиметрические датчики измеряют фактическую массу вещества. Хотя иногда они и дают более точные результаты, чем соответствующие измерения объемного расхода, гравиметрические датчики, как правило, на порядок дороже турбинных. По этой причине их использование, в основном, ограничено теми сферами, где требуется чрезвычайно высокий уровень точности, оправдывающий повышенную стоимость системы (например, в калибровочной лаборатории).

8.2. Основные аспекты измерений потока

Многие из ключевых аспектов измерения потока — общие для всех сенсорных систем (например, диапазон измерений и разрешающая способность), в то время как другие — уникальны для датчиков потока (например, трудности, возникающие в случае очень слабых или очень интенсивных потоков).

Диапазон измерений

Диапазон измерений зависит от физических характеристик контролируемого вещества, свойств турбины, используемой для выполнения измерений, а также природы системы, в которой протекает вещество. Выше некоторой интенсивности потока контролируемый материал может быть поврежден, поскольку поперечные силы начинают буквально разрывать его на части. Кроме того, характер материала может серьезно повлиять на срок службы самого датчика (особенно лопастей), поскольку абразивные материалы в условиях высокоинтенсивного потока могут быстро стирать лопасти турбины. Это неблагоприятно воздействует на качество показаний датчика, поскольку снижение площади лопастей по-

зволяет дополнительному материалу проходить без замера. Кроме того, уменьшение поверхности лопастей ухудшает способность датчика функционировать в условиях потока малой интенсивности из-за уменьшения вращательного давления на датчик.

Характеристики турбины определяют как нижний, так и верхний пределы измеряемого потока, поскольку при слишком слабом потоке турбина, возможно, не будет вращаться (давления от потока вещества будет недостаточно для преодоления силы трения турбины с осью вращения), а слишком сильный поток может физически деформировать лопасти, что приведет к ошибке измерений и износу крепежа турбины на оси (а значит — к вибрации лопастей, которая также искажает результаты измерений). Эта последняя проблема очень распространена, и потому мы к ней еще вернемся в подразделе "Специфика измерений потока", а более подробно рассмотрим в подразделе "Причины погрешностей".

И, наконец, спектр требований к измерению потока определяет система, как таковая. В ситуациях, которые требуют измерения уровней потока, превышающих возможности датчика, может быть использована пробоотборная трубка известного диаметра, которая отводит часть потока вещества, и измерения для нее соотносятся с общим потоком через систему. Следует отметить, что некоторые типовые семейства турбинных датчиков потока поддерживают диапазоны измерения от 0,8 до 230 литров в минуту.[58] Специализированные датчики могут иметь намного более высокий верхний предел измеряемого потока.

Разрешающая способность измерений

Ранее мы исследовали два типа сенсорных систем, для которых характерно формирование сигнала напряжения в непосредственной зависимости от значения замеряемого параметра. В отличие от них, турбинный датчик потока формирует сигнал, у которого измеряемый параметр определяет не напряжение, а *частоту*. Для считывания такого сигнала будет использован немного другой аналоговый интерфейс, преобразующий отраженный свет в цифровой частотный сигнал, который затем подается на один из модулей таймера/счетчика ЦКС dsPIC. Этот модуль подсчитывает число переключений сигнала на тактовом входе. Объединив его с модулем таймера 2, имеющим разрешение периода системной синхронизации (33,9 нс для конфигурации dsPICDEM v1.1), мы обеспечим измерения, соответствующие требованиям любого реального турбинного датчика потока.

---

[58] Семейство RotoFlow™ от компании CITO Products, Inc.

Точность измерений

Турбинные датчики потока могут обеспечить точность измерений до 2% от полной шкалы. Кроме того, полученные с их помощью измерения высоко периодичны при условии, что пространство вращения турбины не засорено, а сама турбина не повреждена из-за износа. Обычно точность уменьшается вблизи крайних значений, т.е. в условиях очень слабого или очень интенсивного потока. При этом это снижение точности может внезапно резко возрасти в случае значительного износа узла турбины — особенно в несущем подшипнике, крепящем турбину к оси. К счастью, существуют методы, позволяющие в некоторой степени компенсировать снижение точности измерений. И хотя они не могут фактически устранить саму причину ошибки, благодаря им характеристики системы ухудшаются постепенно. Зачастую это дает пользователю в ходе наблюдений за системой заранее спланировать требуемое техническое обслуживание, а не реагировать на проблему впопыхах.

Специфика измерений потока

Проблемы с турбинными датчиками возникают, в основном, на нижнем и верхнем пределах потока. Условия слабого потока ставят проектировщика перед дилеммой: ниже определенного уровня давление вещества на лопасти становится недостаточным для преодоления силы трения турбины об ее ось. В такой ситуации материал медленно обтекает неподвижные лопасти, т.е. поток существует, а показания датчика о каком-либо расходе отсутствуют. Выходом из положения может стать добавление в турбину лопастей, поскольку в таком случае увеличивается общее давление потока, а значит — снижается его минимальный допустимый уровень. Тем не менее, это приведет к повышению частоты выходного сигнала датчика (поскольку увеличение числа лопастей означает увеличение числа переключений сигнала на одно вращение), что влечет за собой внедрение дополнительных средств сбора данных.

На другом конце спектра (т.е. в условиях интенсивного потока) возникает проблема со сглаживанием отражательного сигнала: его точки экстремума значительно уменьшаются по амплитуде. В результате для преобразования аналогового отражательного сигнала в цифровое значение возникает необходимость в компараторных схемах, поддерживающих динамическую корректировку порога сравнения, а для правильной установки этого порога микропрограммное обеспечение должно отслеживать экстремумы аналогового сигнала.

Подшипник, крепящий лопасти датчика на оси вращения, имеет тенденцию к износу, который проявляется в увеличении посадочного

отверстия. В результате лопасти начинают вибрировать на оси в различных направлениях. Это означает, что лопасти могут проходить контрольную освещенную точку под различными углами, что приведет к небольшому смещению пиков результирующего сигнала во времени относительно истинного максимума. В зависимости от типа используемого материала и силы потока, подшипник может износиться довольно быстро, что является серьезной проблемой.

Характеристики сигналов

Значение аналогового отражательного сигнала лежит в диапазоне от сотен милливольт до нескольких вольт, поэтому его усиление не является насущной задачей. Однако, ввиду нагрузки сигнала датчика и необходимости устранять синфазные помехи, можно рекомендовать применение измерительного усилителя для буферизации сигнала перед его последующей обработкой. Частотный спектр сигнала турбины легко вычисляется как число лопастей, умноженное на максимальную частоту вращения, обеспечиваемую датчиком. Для соответствия критерию Найквиста минимальная частота дискретизации должна вдвое превосходить полученное значение. Не забывайте, что в этом примере мы квантуем сигнал только для того, чтобы определить его минимальный и максимальный уровни, а не для извлечения параметрической информации из дискретного сигнала напряжения. Задача извлечения значения параметра возлагается на счетчик, и наша главная забота в этом отношении — гарантировать, что счетчик не переполнится во время сбора данных.

Компенсация изменений в плотности материала

Для преобразования измерений объемного расхода в измерения массового расхода датчик должен применить коэффициент плотности материала — значение, зависящее от материала, которое умножается на измеренный объемный расход, чтобы получить соответствующий массовый расход. При этом предполагается, что плотность материала постоянна во всем диапазоне измеряемого потока. Если же это не так, то можно применить коэффициент компенсации изменений в плотности материала. При этом используют кусочно-линейную интерполяцию, аналогичную задействованной при измерениях с помощью термопары.

Линеаризация

В большинстве случаев никакого специального алгоритма линеаризации не требуется, поскольку частота, как правило, возрастает линейно с усилением потока. Эта линейность несколько ухудшается в случае

очень слабого и очень сильного потока, поэтому иногда для обеспечения необходимой точности вблизи границ диапазона измерений датчика может потребоваться простая кусочно-линейная интерполяция тремя отрезками.

Калибровка

Потребности в калибровке для турбинного датчика потока — минимальны, поскольку характеристики выходного сигнала датчика определяются его механикой, а не электрическими параметрами. В общем случае калибровочная характеристика датчика для каждой конкретной модели фиксирована и никакой дополнительной калибровки не требует.

Причины погрешностей

Можно назвать три основные причины деградации сигнала в случае использования турбинного датчика потока:

- износ подшипников;
- замутнение материала;
- электронный шум в аналоговой части интерфейса датчика.

Причины и следствия износа подшипников уже обсуждались выше. Что же касается замутнения материала, то основная проблема здесь заключается в том, что это может привести к значительному ослаблению отражения света от лопастей турбины, а значит — ухудшению способности сенсорной системы достоверно различать проход лопастей через контрольную зону.

Электрический шум также составляет проблему, хотя обычно — не столь серьезную, как в случае с датчиками температуры и нагрузки. Основная область, подверженная влиянию электрического шума, — это вход компаратора. Из-за него на выходе компаратора могут возникать ложные переключения сигнала, что в свою очередь приводит к завышению значений потока по отношению к реальным.

Методы решения перечисленных выше проблем будут рассмотрены в подразделе "Формирование сигналов от датчика". При этом следует отметить, что в пределах диапазона измерений все названные источники погрешностей могут быть надежно компенсированы.

Сбойные ситуации

Единственная сбойная ситуация, которая может быть обнаружена (и то — лишь при определенных обстоятельствах) в случае с турбинным датчиком потока, — это разрыв вывода датчика. Другие системные проблемы, наподобие чрезмерной интенсивности потока, также обнаружи-

ваемы, однако они отражают фактическое, допустимое состояние системы (хотя и могут находиться за пределами возможностей датчика) и потому будут рассмотрены не здесь, а в подразделе "Алгоритмы анализа данных".

Наилучший способ обнаружить разрыв вывода датчика — подключить контрольный резистор между выводом датчика и землей, а затем периодически контролировать напряжение на этом резисторе на предмет его соответствия минимальному ожидаемому уровню сигнала от датчика. В зависимости от уровня на выходе самого турбинного датчика, может потребоваться включение высокоомного усилителя между контрольным резистором и входом АЦП.

8.3. Разработка приложения

Как и в случае с предыдущими приложениями, разрабатываемая система на основе датчика потока довольно проста, однако она хорошо иллюстрирует важные аспекты измерений потока. Наша система будет измерять расход в диапазоне от 0,8 до 30 литров в минуту с помощью турбинного датчика. В случае выхода измеренных значений за нижний или верхний предел допустимого диапазона или же разрыва/отсоединения сигнального проводника включается соответствующий светодиод.

Спецификация системы

В качестве аппаратной платформы система использует плату dsPICDEM vl.l от Microchip и решает следующие задачи:

- измерение потока в диапазоне от 0,8 до 30 литров в минуту с точностью 0,04 литра в минуту;
- предоставление пользователю возможности выполнять через последовательный порт RS-232 (38,4 Кбит/с, восемь информационных бит, один стоп-бит, без контроля четности и управления потоком данных) следующие операции:
  а) устанавливать значение коэффициента потока для любого датчика или материала;
  б) задавать верхний и нижний пределы для измеряемого потока;
- ежесекундное уведомление о замеренном потоке через последовательный порт (сообщения должны быть в текстовом виде);
- уведомление о выходе измеряемого потока за пределы допустимого диапазона путем включения на демонстрационной плате светодиода LED1 и индикации "---", если расход меньше нижнего предела рабочего диапазона, или включения светодиода LED2 и индикации "+++" если расход превысил верхний допустимый предел;

- уведомление об аппаратном сбое (разрыв вывода датчика) путем мерцания светодиода LED3 на демонстрационной плате.

Как и в случае с двумя предшествующими системами, информация, предназначенная для считывания электронными компонентами системы, а не человеком, должна использовать более скоростной бинарный протокол обмена данными.



Программный код реализации этого протокола находится на прилагаемом к книге компакт-диске в файлах `BinaryProtocol.c` и `BinaryProtocolDef.h`.

Формирование сигналов от датчика

Учитывая особую природу сигнала от датчика потока, его формирование несколько отличается от двух предшествующих примеров. В частности, выходной сигнал датчика будет буферизован и пройдет через компаратор. В результате будет получен двоичный цифровой сигнал, который может быть измерен с помощью счетчика ЦКС dsPIC. Для устранения проблем, связанных с электрическим шумом, компаратор может использовать гистерезис, а также механизм корректируемого порога. Структурная схема процесса формирования аналогового сигнала представлена на рис. 8.3.

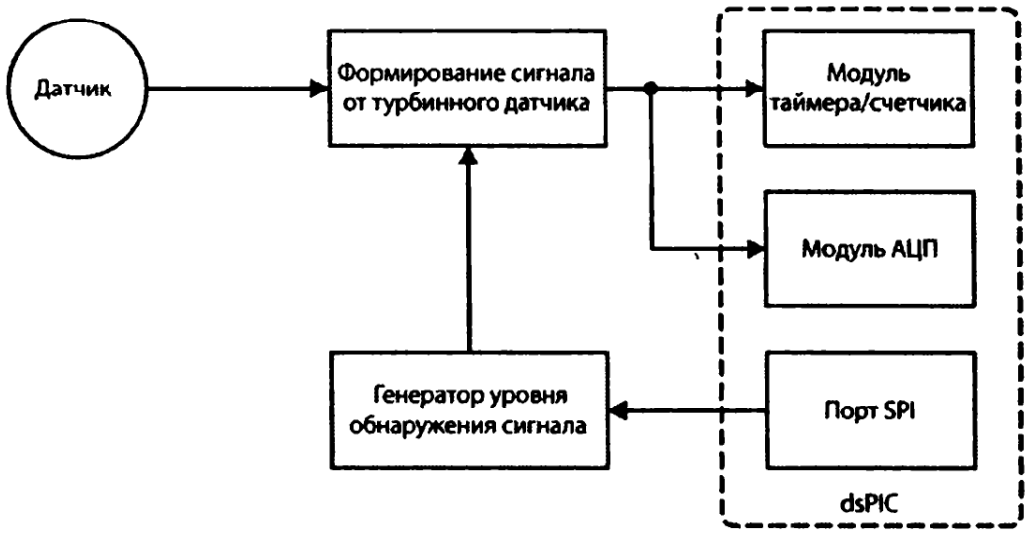


Рис. 8.3. Структурная схема процесса формирования сигнала от датчика потока

Анализ цифровой фильтрации

Система использует фильтр нижних частот для сглаживания "чистого" сигнала напряжения от датчика перед выполнением анализа на предмет попадания в диапазон допустимых значений. В отличие от предыдущих двух приложений, сейчас нет необходимости в узкополосном

режекторном фильтре для удаления сетевого электрического шума, поскольку для подавления этого шума мы можем добавить гистерезис в схему компаратора. Хотя переменная частота формируемого датчиком сигнала несколько усложняет систему, в этом есть и некоторые преимущества. Рассмотрим одно из них.

Нежелательные искажения сигнала, обусловленные износом подшипника, достаточно эффективно устраняются путем усреднения накопленных данных об оборотах за определенное время. Хотя период между отдельными переключениями сигнала при прохождении лопасти через контрольную точку может изменяться из-за вибрации лопастей, подсчитав общее количество переключений за определенный отрезок времени можно получить усредненное значение периода, компенсирующее погрешности. И чем больше время накопления, тем эффективнее усреднение, однако при этом увеличивается время реакции системы.

Разумеется, никто не мешает сделать период накопления отсчетов конфигурируемой величиной. При этом целесообразно предоставить лишь ограниченный набор вариантов, поскольку пользователи обычно незнакомы с нюансами фильтрации. На рис. 8.4 показан пример "нормального" сигнала потока с относительно постоянным периодом прохождения лопасти через контрольную точку, в то время как рис. 8.5 демонстрирует пример сигнала от того же датчика после износа подшипника. Как видим, хотя межпиковые интервалы на рис. 8.5 варьируются значительно сильнее, чем на рис. 8.4, значение среднего периода в обоих случаях практически совпадает.

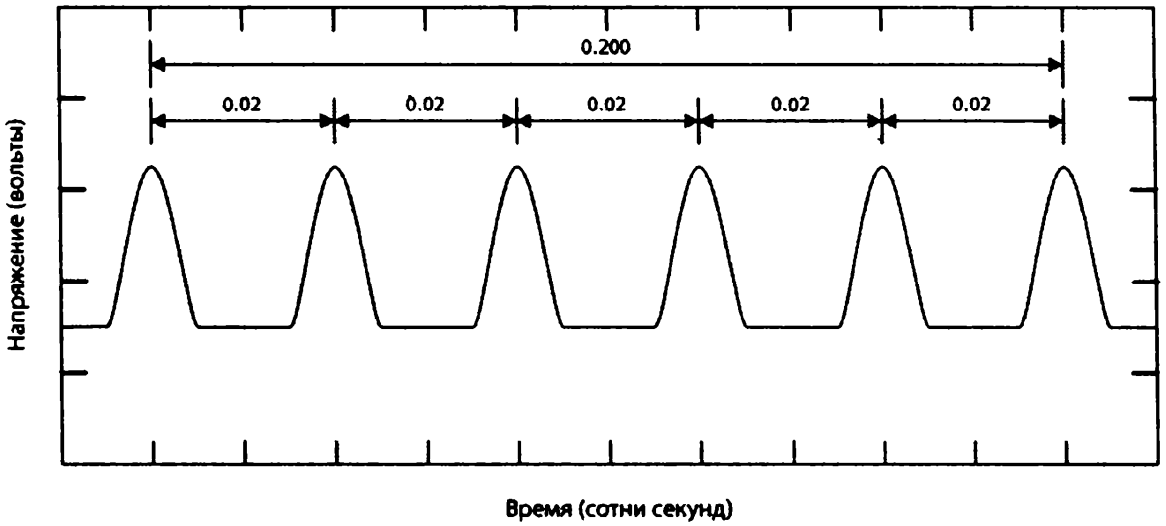


Рис. 8.4. Номинальный сигнал потока при отсутствии вибрации турбины

У читателя мог возникнуть вопрос: "Почему мы используем модуль таймера ЦКС dsPIC для вычисления среднего времени переключения

вместо того, чтобы использовать средства захвата на входе ЦКС dsPIC, позволяющие измерить время между отдельными переключениями сигнала?". Дело в том, что высокое разрешение, обеспечиваемое dsPIC, привело бы к значительному дрожанию измерений, которое постоянно усиливалось бы по мере износа подшипника турбины. Кроме того, на вычисление интенсивности потока, основанное на времени отдельных перемещений лопастей, обычно требуется много вычислительных ресурсов в случае сильного потока, поскольку период прохода лопасти через контрольную точку крайне мал. Накопив значительное количество проходов и усреднив их, мы получим более достоверные результаты, чем в случае накопления большого числа рассчитанных значений интенсивности потока с их последующим усреднением.

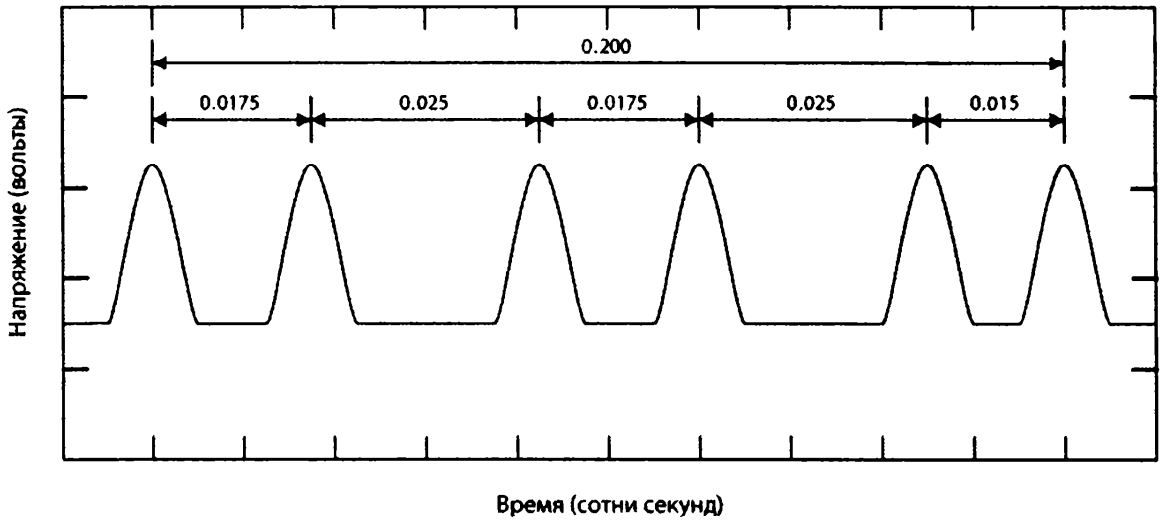
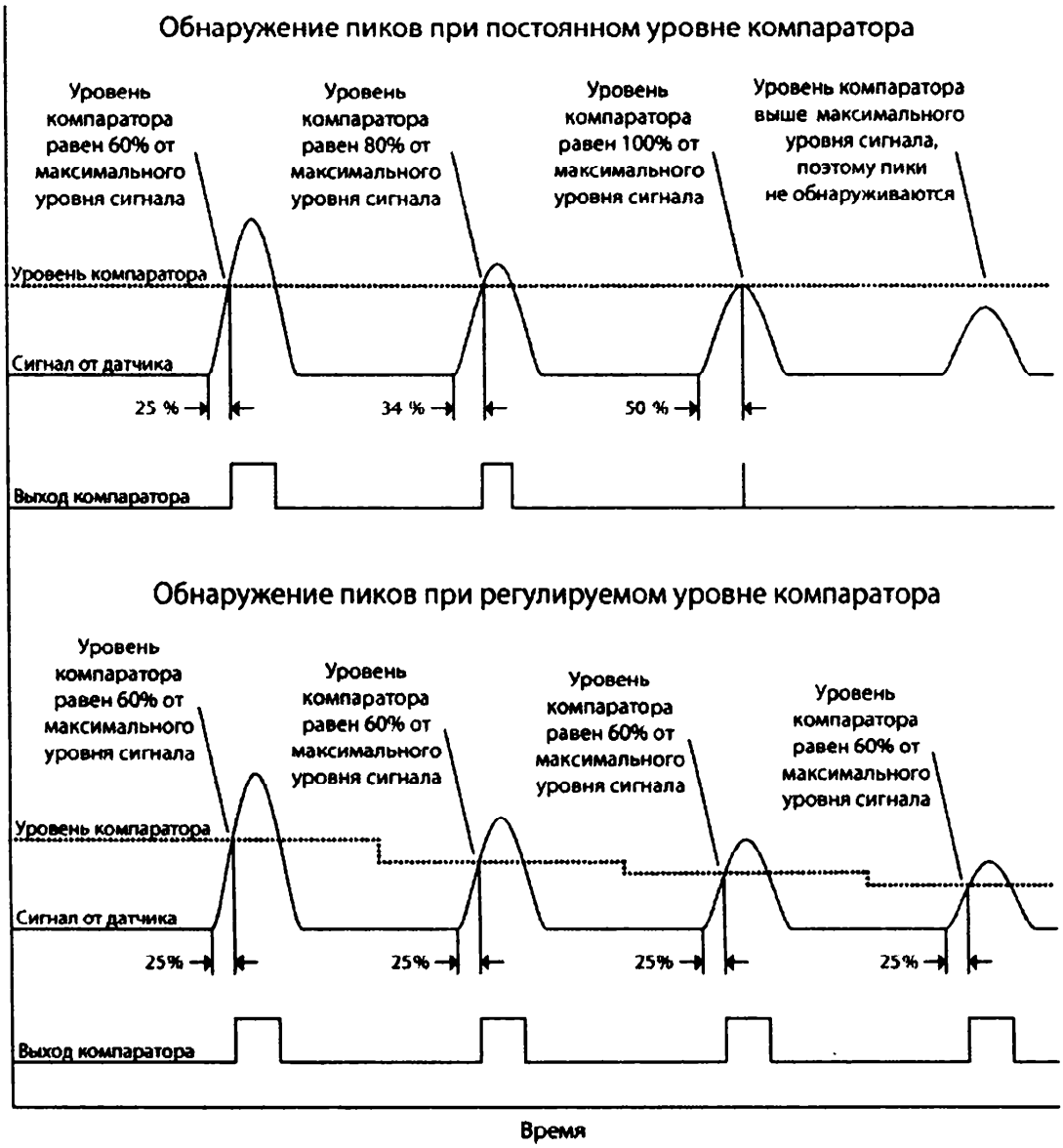


Рис. 8.5. Пример сигнала потока при наличии вибрации турбины

Алгоритмы анализа данных

Алгоритмы анализа данных для датчика потока чуть сложнее аналогичных алгоритмов для приложений контроля температуры и нагрузки, рассмотренных в предыдущих двух главах. Как и ранее, система измерения потока должна проверять отфильтрованные значения выходных сигналов датчика на их соответствие допустимому диапазону, но дополнительно — еще и периодически считывать уровень "чистого" аналогового сигнала от турбинного датчика, чтобы определить, необходима ли корректировка напряжения смещения для схемы компаратора, обнаруживающей пики сигнала. Напомним, что амплитуда "чистого" сигнала ослабевает по мере роста частоты вращения лопастей (т.е. по мере усиления потока). В идеальном случае, во избежание сдвига межпиковых интервалов, процессор должен устанавливать уровень напря-

жения сравнения на одну и ту же относительную точку в выходном сигнале датчика, даже если амплитуда этого сигнала изменилась (рис. 8.6).



Если уровень компаратора не корректируется для поддержания постоянного процентного соотношения с выходным сигналом датчика, то расстояние между передними фронтами выходных сигналов обнаружения пиков будет изменяться при изменении потока (поскольку высота пиков изменяется пропорционально частоте). Наихудший вариант — когда пики вообще не обнаруживаются, из-за чего считываемые значения потока будут значительно занижены по сравнению с реальными.

Обратите внимание на то, что в действительности заранее неизвестно, какой правильный уровень компаратора необходимо установить для первого пика, который значительно отличается от предшествующего, поэтому обнаружение первого пика может демонстрировать некоторое отклонение от оптимальной периода. Обычно, ввиду усреднения по всей группе обнаруженных пиков, это не столь важно.

***Рис. 8.6.** Пример нежелательных изменений межпикового периода*

Алгоритм отслеживания "чистого" выходного сигнала датчика и установки порога компаратора показан на рис. 8.7.

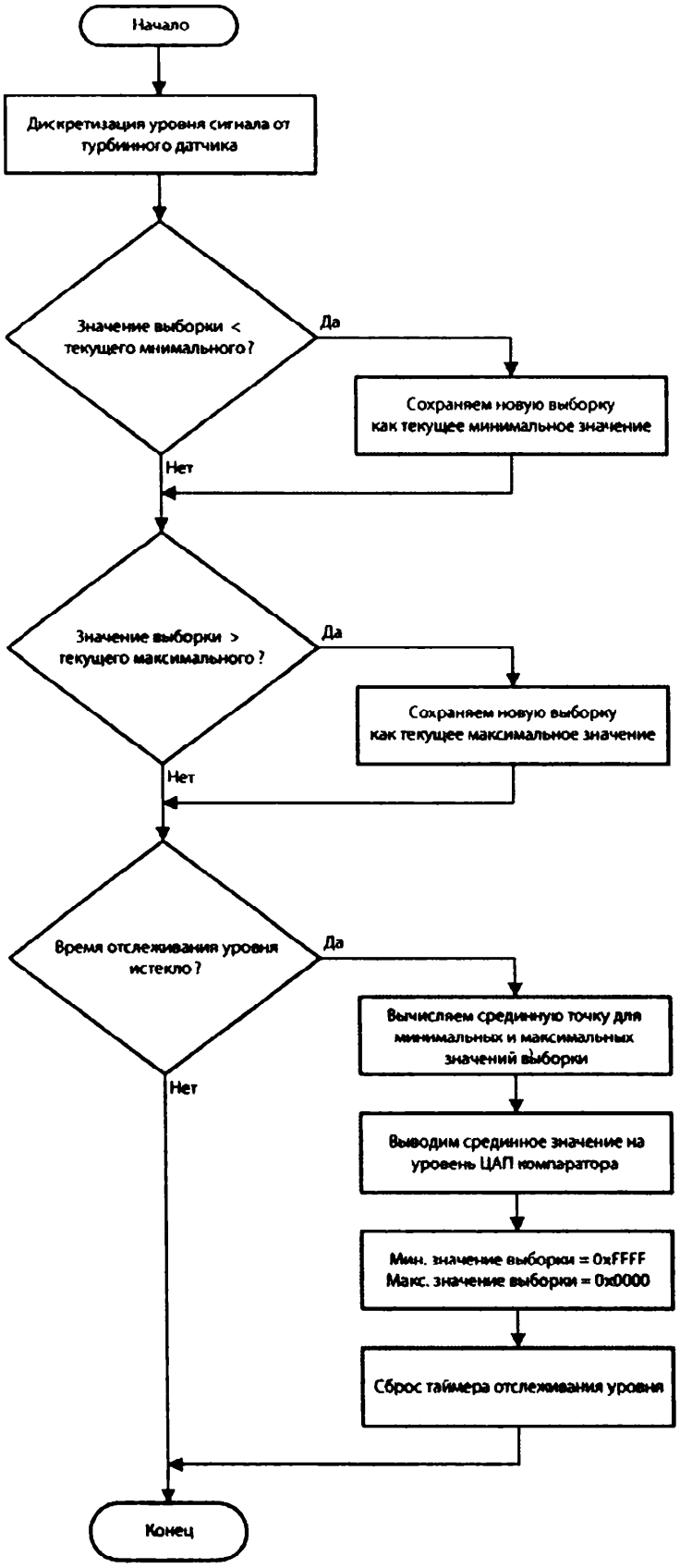


Рис. 8.7. Алгоритм вычисления порогового уровня компаратора

Этот алгоритм, в основном, реализует дискретизацию сигналов датчика, выполняет небольшую фильтрацию, а затем проверяет, не вышли ли какие-либо значения выборок за пределы ранее сохраненных пороговых значений сигнала. Всякий раз, когда алгоритм обнаруживает выборку, значение которой меньше текущего минимально допустимого или больше текущего максимально допустимого, он заменяет соответствующее пороговое значение новым значением выборки. Как только будет обработано требуемое число выборок, алгоритм вычисляет срединную точку для минимальных и максимальных обнаруженных значений и выдает ее в качестве нового уровня компаратора, после чего сбрасывает записанные минимальные и максимальные значения перед сбором и обработкой нового набора выборок.

При определении частоты корректировки уровня компаратора разработчик должен учитывать максимальную скорость изменения потока, т.е. насколько быстро может изменяться интенсивность потока. Это особенно важно при переходе от слабого к сильному потоку, потому что амплитуда выходного сигнала датчика в таком случае изменяется от относительно большой до малой. Если уровень компаратора не откорректировать достаточно быстро, то максимальный уровень выходного сигнала датчика может упасть ниже уровня компаратора, в результате чего на выходе компаратора не будет никаких переключений. Контроллер dsPIC будет воспринимать эту ситуацию как полное отсутствие потока, хотя на самом деле — все наоборот! Хотя на первый взгляд кажется, что уровень компаратора следует обновлять максимально быстро, на самом деле слишком быстрое обновление может вызвать проблемы в условиях слабого потока, поскольку период обновления может оказаться короче периода прохода лопастей. В этом случае обнаруженные экстремумы могут не отражать фактические минимумы и максимумы выходного сигнала датчика, поскольку период обновления недостаточно длительный для того, чтобы охватить весь цикл прохода лопасти.

Протокол обмена данными

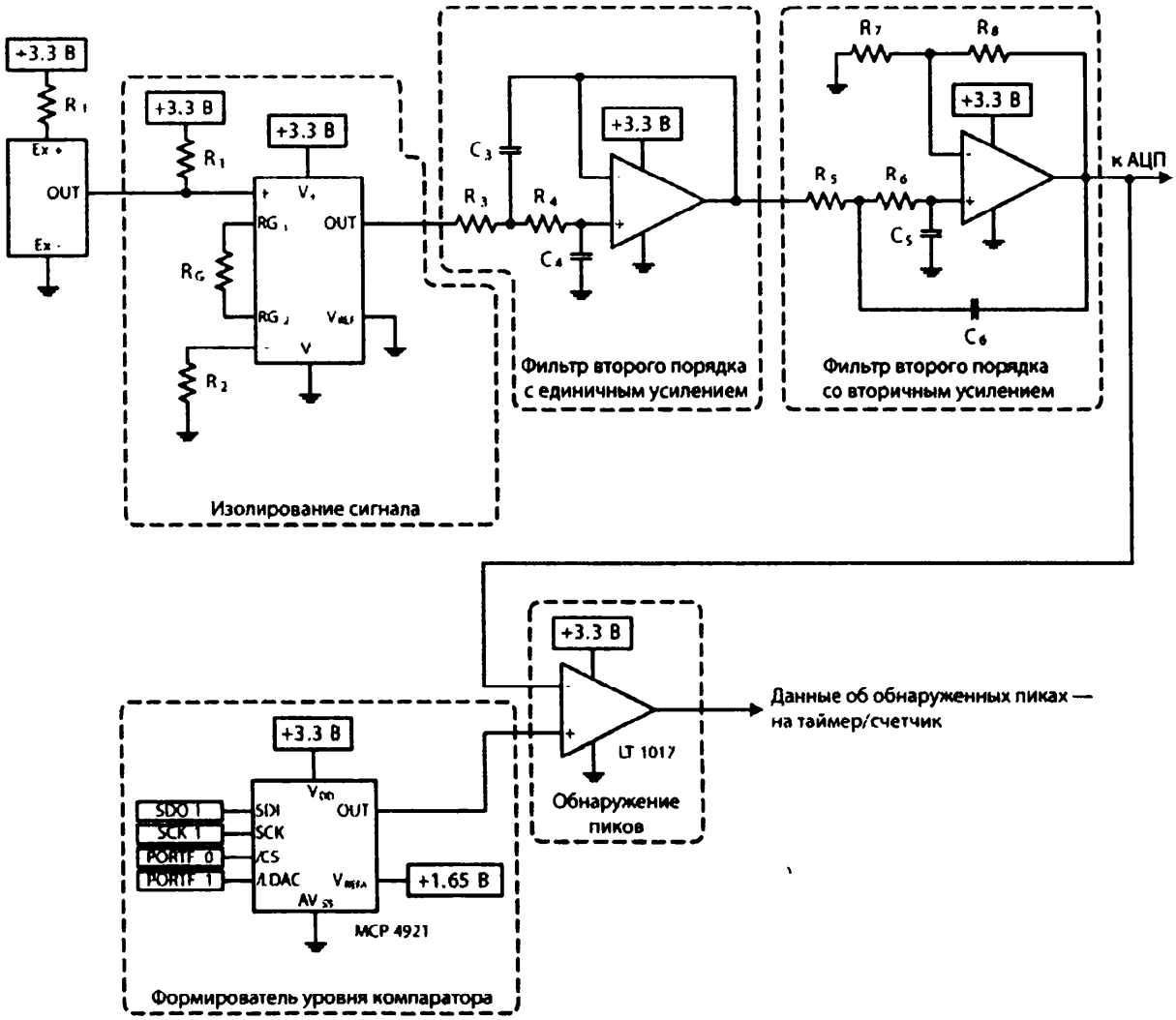
Приложение использует тот же самый базовый протокол обмена данными, что и рассмотренные ранее приложения.

8.4. Аппаратная реализация

Аппаратная реализация датчика потока — проста. Соответствующая структурная схема была показана ранее на рис. 8.3.

Интерфейсная схема турбинного датчика

Интерфейсная схема турбинного датчика (рис. 8.8) обеспечивает датчик питанием и выполняет формирование выходного сигнала для его последующей обработки ЦКС dsPIC.



ПРИМЕЧАНИЯ:

1. R_1 и R_2 обеспечивают как входной путь тока в синфазном режиме, так и способ удостовериться в фактическом соединении датчика со схемой. Номинал резисторов должен быть идентичен и составлять примерно 100 КОм во избежание нагрузки входного сигнала слишком низким входным сопротивлением.
2. Усиление для измерительного усилителя управляется резистором R_G согласно уравнению:

   Усиление = 1 + (50 кОм / R_G)
3. Фильтры имеют частотную характеристику фильтра Баттерворта. В оконечный фильтрующий каскад включен усилитель для компенсации неспособности измерительного усилителя формировать выходные напряжения, близкие по значению к напряжениям шины питания. Усиление оконечного каскада определяется уравнением:

   Усиление = 1 + (R_8 / R_7)

Рис. 8.8. Интерфейсная схема турбинного датчика

Как и в предыдущих приложениях, измерительный усилитель буферизует выходной сигнал датчика, однако его коэффициент усиления равен 1, поскольку турбинный датчик формирует сигнал достаточной амплитуды самостоятельно. Таким образом, измерительный усилитель просто выполняет роль буфера между датчиком и остальной частью системы. Выходной сигнал измерительного усилителя фильтруется каскадированной парой фильтров Баттерворта второго порядка, а затем поступает на вход компаратора, производящего цифровой частотный сигнал, и АЦП, контролирующего уровень сигнала датчика. Обратите внимание на то, что как и в случае с предыдущими двумя приложениями, эта принципиальная схема рассчитана на напряжение питания 3,3 В. Для того чтобы реализовать ее на макетной плате dsPICDEM 1.1 с напряжением питания 5 В, необходимо внести соответствующие изменения в уровни напряжения.

Для реализации корректируемого порога компаратора, в схеме присутствует ЦАП Microchip 4921 — устройство, которое, по сути, дополняет АЦП в том, что преобразовывает входной цифровой сигнал в уровень аналогового напряжения. Подобно АЦП, ЦАП рациометричен, т.е. его выходной сигнал равен диапазону выходного напряжения, умноженному на соотношение цифрового значения, записанного в ЦАП, и максимально возможного цифрового значения. Для 12-разрядного ЦАП с рабочим напряжением 5 В это означает, что выходное напряжение может изменяться от 0 до 4,9988 В для входных цифровых значений от 0 до 0x0FFF (от 0 до 4 095).

Хотя в данном случае 12-разрядный ЦАП вполне приемлем, мы могли бы легко увеличить разрешение ЦАП до 14 разрядов (16 384 уровней) или даже до 16 разрядов (65 536 уровней). Поскольку ЦКС dsPIC подключается к ЦАП по каналу SPI, увеличение разрешения ЦАП не потребовало бы никаких дополнительных аппаратных средств, а лишь — небольшой модификации командных данных, записываемых в ЦАП, для поддержки дополнительных информационных разрядов. С целью упрощения схемного проектирования при работе с макетной платой dsPICDEM 1.1 в примерах программ для установки уровня компаратора используется встроенный цифровой потенциометр SPI.

Поскольку нам необходимо отслеживать уровень сигнала датчика в том виде, в каком он появляется на входе компаратора, предварительный фильтр защиты от наложения спектров должен иметь характеристику с единичным усилением для интересующего диапазона частот. Благодаря оптимально плоской характеристике усиления в полосе пропускания, фильтр Баттерворта для такой задачи — идеален. Гарантиро-

вав, что полоса заграждения начинается далеко за пределами самой высокой составляющей частотного спектра сигнала датчика, мы можем эффективно устранять любые нежелательные частотные компоненты без неблагоприятного влияния на спектральное наполнение сигнала датчика.

8.5. Микропрограммная реализация

Микропрограммная реализация измерения потока, в основном, имеет ту же структуру, что и в двух предшествующих примерах. Ее основные компоненты:

- модуль сбора данных, состоящий из подпрограммы накопления проходов лопастей, подпрограммы обслуживания прерываний от таймера дискретизации и подпрограммы обслуживания прерываний от АЦП;
- модуль фильтрации данных;
- модуль анализа данных, состоящий из подпрограмм анализа количества лопастных проходов и корректировки уровня компаратора;
- модуль распознавания аппаратных сбоев;
- модуль обмена данными.

Модуль сбора данных

Модуль сбора данных отвечает за подсчет проходов лопастей через контрольную точку в течении заданного периода времени, а также за корректировку порогового уровня компаратора для оптимального распознавания таких проходов. Для решения последней из названных задач ЦКС dsPIC квантует уровень сигнала датчика, поступающего на компаратор, определяет соответствующее пороговое напряжение компаратора и выдает полученное значение через интерфейс SPI на цифровой потенциометр, который формирует аналоговое пороговое напряжение.

Отслеживание уровня сигнала датчика

Как и в системе контроля температуры и нагрузки, для формирования тактового сигнала дискретизации в АЦП частотой 1 кГц данное приложение использует таймер 3. Поскольку мы пытаемся определить предельные значения сигнала датчика в течение заданного периода времени, АЦП сконфигурирован таким образом, чтобы формировать запрос на прерывание после получения восьми выборок. Данные, поступающие от датчика потока, — униполярные, поэтому для их хранения мы воспользуемся беззнаковым вещественным форматом. Когда возникает запрос на прерывание от АЦП, подпрограмма обслуживания прерывания

просто копирует содержимое буфера выборок в буфер фильтрации g_frSensorSignal и устанавливает флаг события EVT_FILTER. Это вынудит главный цикл программы выполнить фильтрацию данных об уровнях сигнала от датчика и обновить выходное напряжение ЦАП.

Счетчик проходов лопастей через контрольную точку

Модуль таймера ЦКС dsPIC может быть сконфигурирован на подсчет количества нарастающих фронтов на входе внешнего синхросигнала таймера, который в данном случае будет соединен с выходом компаратора. Задействовав второй таймер для формирования точной временной развертки, мы можем определить число проходов лопастей через контрольную точку на протяжении заданного периода времени и таким образом выяснить соответствующую силу потока.

Частота прохода лопастей может сильно варьироваться и соответствовать предельно минимальной силе потока. Для формирования 16-разрядного синхронного счетчика, предназначенного для измерения этой частоты, мы воспользуемся таймером 5. Модуль сбора данных также использует таймер 3 для формирования тактовых импульсов с периодом 500 мс, предназначенных для накопления отсчетов проходов. Другими словами, считывание и сброс накопленного в счетчике таймера 5 значения происходит каждые 500 мс в течение прерывания от таймера 3 длительностью в одну секунду. Для датчиков потока, с которыми довелось работать автору (модели с четырьмя и шестью лопастями), период накопления отсчетов проходов лопастей в 500 мс был вполне приемлем, однако оптимальный период зависит от времени реакции конкретной системы и ожидаемой скорости потока. Периода накопления 500 мс дает частоту обновления вычисленного потока, равную двум обновлениям в секунду. Если требуется более высокая скорость реакции, то период накопления можно уменьшить (при этом обновление порога компаратора будет производиться чаще), однако это приведет к ухудшению точности усреднения и повысит значение минимальной обнаруживаемой скорости потока, поскольку для индикации ненулевого потока система должна зарегистрировать хотя бы один проход лопасти через контрольную точку за период накопления.

По истечении 500 мс подпрограмма обслуживания прерывания от таймера 3 записывает текущее 16-разрядное значение таймера 5 (содержит число проходов лопастей) в глобальную переменную g_ui16SensorCount и устанавливает флаг события EVT_ANALYZE, чтобы сигнализировать главному циклу программы о том, что необходимо вызвать подпрограмму анализа интенсивности потока.

Модуль фильтрации данных

В отличие от предыдущих двух примеров, в которых сигнал датчика фильтровался для повышения качества параметрического анализа, фильтрация в этом приложении выполняется для улучшения считывания уровня напряжения, чтобы сигнал можно было использовать для оптимальной установки уровня компаратора.

Фильтрация уровня сигнала от датчика

В качестве фильтра уровня сигнала датчика в данном приложении используется ФНЧ. В отличие от схемы фильтрации, использованной в приложении контроля нагрузки, в этой системе для устранения сетевого шума, наложенного на сигнал датчика, мы не будем пытаться применить узкополосный режекторный фильтр, поскольку сигналы такой частоты для данного приложения абсолютно приемлемы. Удаление даже узкой полосы частот в окрестности 50 или 60 Гц привело бы к появлению в области допустимых уровней потока "мертвой зоны", внутри которой выходной уровень компаратора не будет обновляться. Поскольку влияние шума от сети питания на сигнал компаратора — минимально, мы вполне можем им пренебречь.

Фильтр использует частоту дискретизации 1 кГц, что не должно создавать особых нагрузок на процессор, поскольку мы собираемся контролировать только один канал и не будем обрабатывать оцифрованные данные повыборочно. Минимизация потребления вычислительных ресурсов, связанных с оцифровкой, позволяет нам уделить больше времени фильтрации. Таким образом, мы можем реализовать фильтр с очень равномерным усилением во всей полосе пропускания (рис. 8.9).

Модуль анализа данных

Модуль анализа данных отвечает за две основные задачи:

- проверка последнего отфильтрованного значения потока на предмет его соответствия заданным пользователем допустимым пределам и управление соответствующей предупредительной индикацией;
- анализ уровня буферизированного выходного сигнала датчика с целью:

  а) определить, соединен ли датчик с системой;

  б) если датчик соединен с системой, — вычислить оптимальное порогового напряжения компаратора с последующей выдачей этого значения на ЦАП.

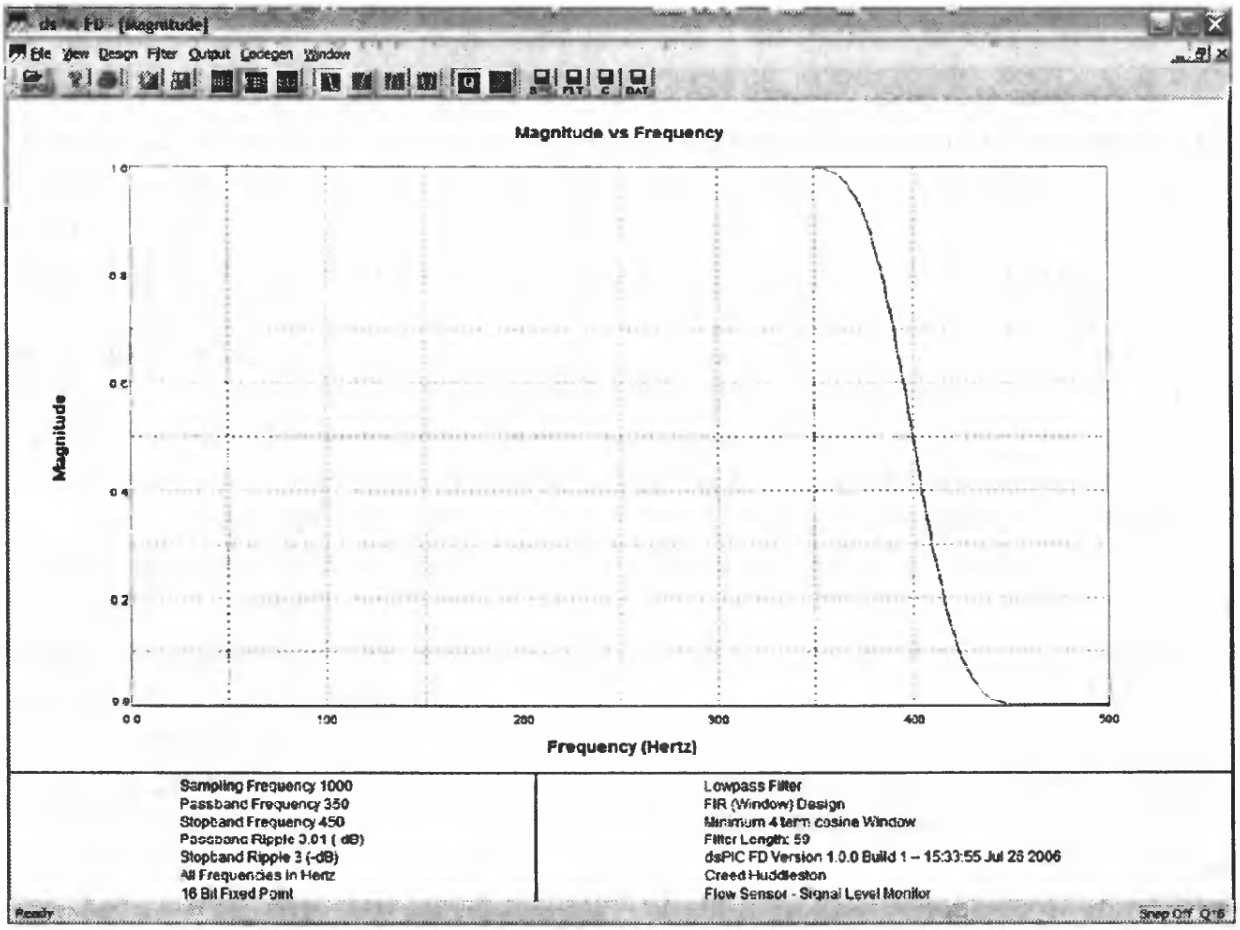


Рис. 8.9. *Частотная характеристика фильтра отслеживания уровня сигнала*

Анализ скорости потока

Анализ скорости потока носит такой же характер, что и в двух предыдущих приложениях: последнее отфильтрованное значение проверяется на соответствие заданным допустимым пределам, устанавливается состояние соответствующего светодиода, а затем прибавляется единица к глобальному счетчику, который вычисляет период с момента выдачи через коммуникационный порт сообщения о предыдущем значении потока. Если с момента последнего сообщения прошла секунда, то функция устанавливает флаг события `EVT_REPORT_RESULTS`, чтобы уведомить главный цикл программы о необходимости выдать через коммуникационный порт сообщение, содержащее последнее значение потока. В заключение функция сбрасывает флаг события `EVT_ANALYZE_DATA`, чтобы уведомить о завершении анализа данных для текущего отфильтрованного значения потока, и передает управление вызвавшей подпрограмме.

Анализ уровня сигнала

Функция анализа уровня сигнала датчика вначале проверяет, действительно ли датчик подсоединен ко входу системы. Если датчик не подключен правильно, то подтягивающий резистор на плюсовой стороне измерительного усилителя обеспечит формирование на выходе измерительного усилителя постоянного сигнала, близкого по значению напряжению на плюсовой шине питания. Во всех остальных случаях выходной сигнал измерительного усилителя будет ниже напряжения на этой шине, хотя он может временно приближаться к нему, когда лопасть турбины проходит через освещенную точку (в этот момент присутствует максимальный отраженный от лопасти сигнал).

Чтобы удостовериться в подключении датчика к системе, приложение просто исследует уровень отфильтрованного сигнала от датчика. Если уровень сигнала превышает предварительно заданную величину (например, 0,5 секунд), то датчик не подключен. Точный минимальный уровень напряжения обнаружения и период времени, достаточный для выдачи сигнала об аппаратном сбое, могут быть откорректированы разработчиком, однако эта комбинация должна находиться за пределами максимального уровня сигнала и периода наличия уровня в рабочем режиме системы.

Если предварительный анализ указывает на то, что датчик не соединен с системой, то устанавливается глобальный флаг `g_bHWError`, включается соответствующий светодиод, а этап анализа порогового уровня компаратора пропускается, поскольку сигнал датчика в любом случае недостоверен. Если же датчик подключен к системе, то функция сбрасывает флаг `g_bHWError`, выключает светодиод аппаратной ошибки и переходит к анализу порогового уровня компаратора.

Анализ порогового уровня компаратора крайне прост: исследуются повыборочно отфильтрованные значения уровня, чтобы определить минимальное и максимальное значения на протяжении заданного периода обновления порога, вычисляется срединная точка двух экстремумов, после чего вычисленное значение выдается на ЦАП с целью установки нового уровня компаратора.

Как только анализ порогового уровня компаратора завершен (или сразу же, если датчик не подключен), главный программный цикл сбрасывает флаг события `EVT_ANALYZE`.

Модуль реализации протокола обмена данными

Всякий раз при обнаружении события `EVT_REPORT_RESULTS` главный программный цикл вызывает функцию `FormatResultsMsg()`. Эта

функция форматирует последние данные о значении потока, после чего основной цикл передает отформатированное сообщение хосту с помощью функции `TransmitResults()`. Как только сообщение о результатах послано, главный цикл сбрасывает флаг `EVT_REPORT_RESULTS`, чтобы уведомить о завершении обработки события. В данном приложении используется тот же протокол обмена данными, что и в системах контроля температуры и нагрузки, поэтому здесь он не рассматривается.

8.6. Резюме

Приложение измерения потока, разработанное в этой главе, познакомило читателя с новой формой сбора данных, при которой представляющие интерес данные заключены в частоте, а не в уровне напряжения выходного сигнала датчика. Это — простой и эффективный способ кодирования данных, однако при использовании рассмотренного подхода в других приложениях следует учитывать некоторые тонкости. Так, необходимо гарантировать, что система сможет обработать минимально и максимально допустимые частоты сигнала. При этом особое внимание следует уделить сигналам с очень высокой частотой, поскольку в плохо спроектированной системе они могут легко заблокировать возможность обработки данных.

Очевидно, что мы могли бы повысить эффективность приложения, усовершенствовав его (например, реализовав чувствительное к скорости потока накопление отсчетов проходов лопастей, что позволило бы использовать более длительное накопление в случае слабого потока и более короткое в случае сильного потока, или же добавив резистор считывания тока в линию питания датчика, чтобы расширить диагностические возможности системы. Тем не менее, целью данной главы было лишь дать читателю общее представление о разработке интерфейса рассматриваемого датчика, а не создать исчерпывающий перечень его возможностей.

В последней главе этой книги мы проанализируем перспективные направления развития интеллектуальных сенсорных систем с технической и экономической точек зрения.

9

Взгляд в будущее

Лучший способ предсказать будущее — изобрести его.

— Алан Кей[59]

Теперь, когда мы в достаточной степени освоили интеллектуальные датчики и концепции, необходимые для их реализации, направим наш взгляд в будущее этих мощных устройств. Как и любая новая технология, интеллектуальные датчики обладают огромным потенциалом, но с ними также связан и ряд проблем, которые должны быть решены для повсеместного внедрения интеллектуальных датчиков. Каков прогноз для сферы интеллектуальных измерений на ближайшие несколько лет? Каковы стимулы продвижения интеллектуальных датчиков на рынок, и что им препятствует? На какие рынки могут проникнуть интеллектуальные датчики, и какие новые возможности они откроют? Ответы на эти вопросы мы и постараемся дать в данной главе. Хотя мы и не сможем рассказать все обо всем, у читателя, по крайней мере, сформируется понимание основных вех на пути развития интеллектуальных датчиков. Кроме того, в этой главе даны ссылки на ресурсы, которые позволят читателю более глубоко ознакомиться с этой увлекательной темой.

9.1. Технологические тенденции

Интеллектуальные датчики состоят из трех основных компонентов: чувствительных элементов, вычислительных элементов и коммуника-

---

[59] Алан Кей (Alan Kay) — гигант в области компьютерных наук. Он был одним из основателей знаменитого Исследовательского центра Xerox в Пало Альто (PARC). Являясь пионером в области современного объектно-ориентированного программирования и оконных пользовательских интерфейсов средствами компьютерного языка Smalltalk, Кей разработал графический объектно-ориентированный персональный компьютер во время своей работы над докторской диссертацией в университете штата Юта. Хотя сегодня таким никого не удивишь, не забывайте, что Кей сделал это в 1969 году. На тот момент это был настоящий подвиг и прорыв. Так что, этот человек знает, о чем говорит.

ционных интерфейсов, — и технологии, лежащие в их основе, стремительно развиваются. На протяжении следующих нескольких лет мы станем свидетелями значительного расширения возможностей интеллектуальных датчиков при одновременном снижении их стоимости. Рассмотрим подробнее достижения в каждой из трех перечисленных областей.

Тенденции развития чувствительных элементов

Может показаться, что разработка в сфере первичных преобразователей довольно статична, однако такое предположение в корне неверно. Новые наработки в этой сфере значительно повысили чувствительность различных типов датчиков — особенно в области химического анализа. Теперь разработчики могут создавать системы меньших физических размеров, потребляющие меньше электроэнергии и требующие меньшего количества анализируемого вещества для получения точных измерений.

Нигде эта тенденция не проявляется более явно, чем в мире жидкостного микроанализа. В данном случае, когда мы говорим "микро", то это означает действительно очень малые количества вплоть до нанолитров. И такой анализ находит самое широкое применение: от струйных принтеров до лабораторных устройств исследования крови. Одно из направлений жидкостного микроанализа: *цифровой микроанализ* — имеет дело с точным манипулированием долями капель (например, с целью определения химического состава жидкости).

Само собой разумеется, качество работы с такими малыми количествами вещества зависит от способности системы точно измерять важные параметры, связанные с капельками жидкости. Если система способна поддерживать требуемую среду, то выигрыш будет огромным: уменьшение на несколько порядков количества анализируемого вещества, выдача результатов уже через несколько минут, а не несколько дней, улучшение переносимости системы и снижение ее стоимости. Хотя жидкостный микроанализ — не единственная область, в которой были достигнуты значительные успехи в технологиях датчиков, эти успехи, конечно же, — одни из наиболее впечатляющих.

Разработчики могут рассчитывать на то, что тенденция миниатюризации и повышения чувствительности первичных преобразователей будет сохраняться и в обозримом будущем. Достижения в смежных областях, наподобие нанотехнологий, позволят расширить диапазон рабочих сред, а также приведут к созданию совершенно новых типов датчиков. Однако для эффективного использования этих достижений разработчики должны не только знать о технологических новинках, но и понимать

физические явления, лежащие в основе функционирования преобразователей.

Тенденции развития вычислительных элементов

Большинство из нас слышало о законе Мура, который в популярном изложении гласит, что вычислительная мощность удваивается каждые два года[60]. Хотя мы можем наверняка ожидать повышения вычислительной мощности компьютеров будущего, встроенные микропроцессорные системы, наподобие тех, которые были исследованы в этой книге, кроме "чистой" вычислительной мощности, характеризуются и другими важными параметрами. Потребляемая мощность, физические размеры и открытая аппаратная и программная архитектура — вот факторы, которые в ряде приложений могут оказаться важнее, вычислительной мощности.

Значительное снижение энергопотребления

Во многих случаях потребляемая мощность — это критический фактор продления срока службы батареи в портативных или удаленных системах, который, к тому же, определяет степень нежелательного самонагревания. Многие цифровые электронные схемы, включая ЦКС dsPIC и большинство процессоров, сконструированы таким образом, что потребляемая ими мощность (а значит и выделяемое ими тепло), пропорциональна их тактовой частоте. Удвойте тактовую частоту для повышения скорости выполнения приложения, и вы удвоите потребляемую микросхемой мощность. Хотя иногда это создает проблемы, из данной особенности вытекает и решение: отключив компоненты микросхемы, которые в данный момент не нужны, и/или уменьшив тактовую частоту в состоянии простоя, мы можем значительное уменьшить энергопотребление. Это особенно важно для систем с питанием от батареи, которые должны функционировать недели, месяцы и даже годы без перезарядки.

---

[60] Этот закон назван именем Гордона Мура (Gordon Moore) — соучредителя корпорации Intel, создавшей процессор PC. Гордон Мур сделал свое предсказание в 1965 году. На самом деле, согласно Web-сайту Intel, Мур установил, что *число транзисторов на кристалле* удваивается приблизительно каждые два года. Поскольку число транзисторов в микропроцессоре может служить грубой аппроксимацией вычислительной мощности кристалла, со временем "закон" трансформировался в свою более известную форму. Его детальный анализ можно найди по адресу `http://www.intel.com/technology/silicon/mooreslaw`.

Мы уже стали свидетелями резкого снижения мощности, потребляемой процессорами и электронными схемами, формирующими сигнал от датчика, и можем ожидать укрепление этой тенденция и в будущем в ответ на насущную потребность продления срока службы батарей в мобильных устройствах (он должен измеряться в годах, а не днях). В ходе одной из особенно ярких рекламных кампаний Microchip была продемонстрирована микропроцессорная система контроля температуры, источником питания в которой служил грейпфрут. Этим было показано, насколько непритязательны компоненты Microchip в отношении потребляемой мощности[61]. И такое низкой энергопотребление в будущем станет нормой, а не исключением.

К сожалению, ничего в этой жизни не дается даром. Для значительного снижения потребляемой мощности разработчики микросхемы должны уменьшить напряжение питания микросхемы до 3,3 В (а иногда — еще больше). Это чревато серьезными последствиями — особенно в цепи аналоговых сигналов. Понижая напряжения питания, мы в результате увеличиваем уровень шума в цепи аналоговых сигналов. Например, если у нас есть 10 мВ шума в аналоговом сигнале с размахом 5 В, то мы получим уровень шума 2% (10 мВ / 5 В = 0,02). Тот же самый шум в аналоговом сигнале с размахом 2,5 В, по сути, удваивается до 4%, что может вызвать довольно серьезные погрешности, если шум не будет правильно обработан. Этот эффект может усилиться для напряжения возбуждения, которое должно пройти значительное расстояние в электрически зашумленной среде. Фактически, это — одна из причин того, что во многих более ранних сенсорных системах в качестве напряжения возбуждения датчиков использовалось ±10 В и даже ±15 В. Такой размах напряжения минимизировал влияние шума на выходной сигнал.

Другой отрицательный аспект снижения напряжения питания заключается в затруднении сопряжения таких компонентов с более ранними системами, в которых используется напряжение питания 5 В или выше. Нельзя сказать, что сопряжение новых низковольтных систем со старыми высоковольтными абсолютно невозможно, однако в таком случае для обеспечения надежной работы требуется преобразование уровня напряжения того или иного вида. Хотя сигналы ввода-вывода новых электронных компонентов поддерживают напряжение 5 В, фактически выходные сигналы с более низким напряжением питания не

---

[61] В демонстрации использовался микроконтроллер PIC, изготовленный с применением изобретенной Microchip технологии nanoWatt™.

в состоянии нагрузить входы системы на 5 В достаточно высоким уровнем напряжения. Со временем эта проблема исчезнет, поскольку все большее и большее систем будет переходить на низковольтное питание, однако в ближайшем будущем этот фактор следует учитывать.

Непрерывное уменьшение размеров

Наряду со снижением потребляемой мощности в интеллектуальных сенсорных системах будет наблюдаться тенденция к дальнейшей миниатюризации этих систем. Уже сегодня на рынке доступны сенсорные системы с питанием от батареи, имеющие размер с монету в 25 центов, однако в будущем устройства станут еще меньше. Эти полностью самодостаточные устройства, которые иногда называют "пылинками", содержат встроенные радиоканалы, позволяющие формировать специализированные коммуникационные сети (подробнее этот вопрос освещен в подразделе "Тенденции в сфере коммуникаций"). Таким образом модно организовать обмен данными с другими "пылинками" в сети или с хостом. Звучит уже достаточно впечатляюще, но сегодня уже существуют "пылинки" со стороной в 5 мм и ведутся разработки систем объемом всего 1 мм^3.[62] Эти крошечные устройства размером с песчинку будут содержать все: датчик, процессор, радиосхемы и батарейку.

Такая миниатюризация приносит с собой много вопросов, связанных с компоновкой и работой устройств, — особенно в области создания источников энергии, которые имели бы столь малые размеры и в то же время обеспечивали приемлемое время функционирования системы. Вторая проблема — организация связи с такими маленькими устройствами, поскольку трудно себе представить, что кто-то сможет подключиться к ним с помощью стандартных кабельных соединений. Именно этой проблеме и посвящен следующий подраздел.

Тенденции в сфере коммуникаций

Не менее важным, чем достижения в области электронных схем, для будущего интеллектуальных сенсорных систем является невероятный прогресс в области коммуникационных технологий, ведущий к огромному распространению интеллектуальных датчиков. В будущем обмен данными будет беспроводным, организованным через специализиро-

---

[62] Согласно цитате Кристофера Пистера (Kristofer Pister) — профессора электротехники из университета Калифорнии в Беркли — в онлайновой версии журнала "ComputerWorld" от 23 марта 2004г., www.computerworld.com/mobiletopics/mobile/story/0,10801,79572,00.html.

ванные сети датчиков, которая формируется автоматически при минимальном вмешательстве пользователя или вообще без такового, обеспечивая возможность быстрого развертывания. Особенно это касается "пылинок" и других миниатюрных сенсорных систем. С физической точки зрения, большие сенсорные системы могут по-прежнему использовать проводные соединения, однако они все в большей степени будут переходить на высокоскоростные сетевые стандарты (100 Мбит/с и выше), облегчающие сосредоточение, распространение и хранение больших информационных массивов, формируемых этими системами. Рассмотрим некоторые из конкретных путей развития коммуникационных технологий интеллектуальных сенсорных систем.

Повсеместное распространение сети Internet

Глядя на современные тенденции, можно с уверенностью сказать, что в будущем доступ к сети Internet станет повсеместным. Уже сегодня такой доступ предоставляют самые разнообразные портативные устройства: мобильные телефоны, карманные организаторы, ноутбуки и т.д. Процесс подключения интеллектуальных сенсорных систем к Internet движется тем же проторенным путем. Хотя кто-то может рассматривать это как простую погоню за модой, истина заключается в том, что многие приложения извлекут реальную пользу из доступа ко всемирной сети. Однако прежде, чем такие системы станут реальностью, необходимо решить ряд серьезных проблем.

Первая из таких проблем довольно прозаична: необходимо выделить адреса для всех этих устройств. Используемый в настоящее время Internet-протокол IPv4 поддерживает до $4{,}3 \times 10^9$ уникальных адресов (4,3 миллиарда). Это выглядит внушительно, пока не поймешь, что такое адресное пространство предоставляет менее одного адреса на каждого жителя планеты. Учитывая, что каждый человек в своей личной и профессиональной жизнедеятельности управляет различными устройствами (автомобили, бытовые приборы, компьютеры и т.д.), и еще больше таких устройств находится в заведовании организаций (школ, корпораций, правительства и т.д.), можно сделать вывод о том, что используемая в настоящее время схема адресации совершенно неадекватна задаче объединения всех этих объектов. К счастью, протокол Internet следующего поколения IPv6 поддерживает намного больше адресов: $3{,}4 \times 10^{38}$. Этого вполне достаточно, чтобы удовлетворить как современные потребности, так и потребности обозримого будущего. Существуют и другие подходы к решению этой задачи, однако ни один из них не был столь же успешен, как IPv6.

Вторая проблема заключается в пропускной способности. Как мы сможем эффективно передать настолько большой объем информации, сгенерированной глобально взаимосвязанной системой? Хотя мы все являемся свидетелями стремительного развития широкополосных Internet-соединений (кабельный, DSL и т.д.), обмен данными предполагаемого масштаба потребует значительно большей пропускной способности, чем это возможно в настоящее время. В рамках проекта Internet2[63] разрабатывается намного более скоростная инфраструктура, обеспечивающая обмен данными со скоростью 10 Гбайт/с, но даже это — только начало. Значительную работу следует проделать в области высокоскоростной маршрутизации и выборки информации для снижения избыточности сетевого трафика до уровней, которые могут поддерживаться надежно и экономически эффективно.

Беспроводная связь

Если бы все новые системы для организации обмена данными требовали кабельных соединений, то ситуация стала бы безнадежной. Это было бы абсолютно неприемлемо с точки зрения стоимости, и кроме того, такие системы были бы непомерно громоздкими. К счастью, беспроводная связь приобретает все более широкое распространение. Она проста в реализации и очень экономически эффективна, поскольку во многих ситуациях позволяет избавиться от лишних проводов. Традиционно это относилось только к коммуникационным кабелям, однако сегодня все чаще появляется возможность избавиться также и от силовых кабелей за счет применения устройств с питанием от батарей или пассивным питанием.

В настоящее время доступны несколько беспроводных схем связи, приспособленных к различным классам приложений. Относительно новый протокол ZigBee™ хорошо подходит для маломощных устройств с незначительными скоростями передачи данных, от которых может потребоваться работа в течении нескольких лет на одной батарее. В таких случаях с целью снижения потребляемой мощности обмен данными ограничен как по продолжительности, так и по частоте сеансов.

Другие протоколы, используемые для беспроводных локальных сетей, наподобие популярного 802.11x, способны организовать непрерывный информационный обмен и применяются, например, в системах контроля и управления в режиме реального времени (конечно же, с возможностью прерываний). Как протокол ZigBee, так и протокол 802.11x

---

[63] Подробности о проекте Internet2 можно найти по адресу `www.internet2.edu`.

предназначены для обслуживания географически ограниченных областей. На другом конце спектра находятся беспроводные схемы на базе сотовых модемов и других методик подключения к устройствам в мировом масштабе.

В будущем можно ожидать гораздо более широкого распространения беспроводных технологий, обусловленного желанием снизить расходы на инфраструктуру, упростить монтаж и повысить эффективность систем.

Безопасность

С точки зрения грядущих перспектив огромную озабоченность вызывает вопрос защиты сетей устройств от несанкционированного слежения или, что еще хуже, — вмешательства со стороны. Учитывая тип информации, передаваемой системой, можно сказать, что эта озабоченность не менее справедлива для интеллектуальных датчиков, чем для финансовых учреждений. Так, секретная информация, используемая для управления технологическими процессами, должна быть защищена от преднамеренного искажения и хищения во избежание злонамеренного внешнего вмешательства в технологический процесс и утечки конфиденциальных данных. Необходимость в обеспечении безопасности характерна как для проводных, так и для беспроводных сетей.

Разработчики протоколов теперь признают всю важность обеспечения безопасности для организации надежной связи. В силу этого, в новых протоколах, наподобие ZigBee, элементы защиты — не опциональные, а основополагающие. К сожалению, средства защиты вносят избыточность в информационный поток, что снижает пропускную способность и, в зависимости от реализации, повышает стоимость аппаратной и/или программной части системы. Однако потенциальные потери в результате нарушения безопасности обычно значительно перевешивают издержки, связанные с внедрением средств защиты в конечный продукт.

Специализированные сети

Во многих сетевых задачах одна из наиболее значительных затратных статей (как в смысле времени, так и в смысле денег) — это администрирование. Сопровождение более-менее крупной сети обычно требует внимания (и связанных с этим затрат) высококвалифицированного администратора, — именно то, чего многие из приложений, ориентированных на использование интеллектуальных сенсорных систем, просто не могут себе позволить. Что требуется для процветания сетей датчиков — так это возможность быстрого развертывания таких сетей операто-

рами относительно низкой квалификации. Другими словами, сетевые узлы должны конфигурироваться для правильного функционирования автоматически. Такие сети известны как *специализированные* или *самоорганизующиеся*, и первые надежные системы такого типа уже в действии. Пример — организационная схема ZigBee, позволяющая узлам входить в сеть и выходить из нее без значительного реконфигурирования сети. Существуют и другие реализации данного подхода, хотя зачастую они — узкоспециализированные. Впрочем, в относительно недалеком будущем (в течении нескольких ближайших лет) можно ожидать значительного прогресса в этой области.

Итак, рассмотрев технологические тенденции, формирующие сферу интеллектуальных сенсорных систем, переходим к экономическим тенденциям, определяющим приемлемость технологических решений.

9.2. Экономические тенденции

Будущий спрос на интеллектуальные сенсорные системы будет обусловлен тремя ключевыми экономическими факторами:

- демография стареющего населения;
- повышение степени глобализации технологических процессов;
- создание новых бизнес-возможностей.

Все три фактора указывают на резкое повышение спроса на интеллектуальные сенсорные системы.

Демография стареющего населения

Во всех развитых странах рождаемость упала ниже уровня, который требуется для сохранения численности населения, и эта тенденция удерживается уже не первый год. Старение населения (и особенно его трудоспособной части) было распознано уже довольно давно, однако его влияние начинает ощущаться только сейчас в связи с выходом на пенсию поколения, родившегося в послевоенные годы (так называемый, период "бэби-бума"). В промышленности, здравоохранении и других сферах мы скоро потеряем значительную часть рабочей силы, причем ее адекватная замена в ближайшее время не предвидится.[64] Если мы хотим поддерживать экономику, обеспечивающую современный уровень производства товаров и услуг, то необходимо радикально повысить продуктивность труда каждого работающего, чтобы возместить ущерб в ре-

[64] Мы не хотим сказать, что молодые работники некомпетентны.,— просто количество таких работников недостаточно для того, чтобы заменить тех, кто уволился.

зультате потери работников, ушедших на пенсию. Наиболее эффективный (и, пожалуй, единственный) способ достичь этого заключается в использовании интеллектуальных сенсорных систем, которые самоорганизуются, самодиагностируются и в целом требуют намного меньше человеческого вмешательства, чем существующие системы.

Между прочим, эта проблема характерна не только для развитых стран. Тенденция снижения рождаемости наблюдается по всему миру, включая развивающиеся страны. Если с ней не начать бороться (что, очевидно, рано или поздно произойдет, хотя на это и понадобится некоторое время), то все страны в конце концов столкнутся с той же дилеммой.

Повышение степени глобализации производственных процессов

Независимо от вашей позиции в отношении глобализации, ее существование — неоспоримый факт, и в обозримом будущем в этом смысле ничего не изменится, поскольку компании размещают ресурсы в тех географических регионах, где это будет наиболее эффективно. Хотя глобализация помогает уменьшить издержки организаций, она затрудняет управленческие задачи. Рабочие группы рассредоточены, и становится сложнее собирать и обрабатывать информацию, необходимую для принятия эффективных решений. Для восполнения этого пробела компании все больше используют интеллектуальные сенсорные системы, чтобы предоставлять руководителям и техническому персоналу, ответственному за слаженную работу организации, информацию в режиме реального времени. Поскольку лица, принимающие решения, и технический персонал физически могут находиться вдалеке от непосредственных процессов (и зачастую именно так и есть), коммуникационные возможности интеллектуальных сенсорных систем могут оказаться критическим фактором для их способности хорошо выполнять свою работу.

Как и в случае со старением населения, глобализация — это не только проблема развитых стран. Такие регионы, как Индия и Китай, которые обычно рассматривались лишь как получатели технологических процессов извне, теперь сами вынуждены переносить производство в отдаленные области своих стран или даже за рубеж, чтобы сохранить экономическую эффективность.

Новые бизнес-возможности

Точно так же, как сеть железных дорог в западных штатах США в конце девятнадцатого столетия предложила предпринимателям новые

бизнес-возможности, резко снизив время и стоимость транспортировки товаров и людей, так и развитие интеллектуальных сенсорных систем предоставляет проницательным предпринимателям новые способы извлечения выгоды из распределенного вычислительного потенциала и использования устройств, подключенных к глобальной сети. Уже сегодня передовые компании применяют интеллектуальные сенсорные системы в широком спектре задач для поставки конечного продукта потребителям и возмещения части его стоимости в виде доходов. В своем одностраничном релизе исследовательская компания Harbor Research определяет почти сотню прикладных сегментов, которые могут выиграть от использования интеллектуальных устройств, подключенных к сети.[65]

Благодаря интеллектуальным сенсорным системам, поставщики датчиков и интеграторы теперь могут формировать непрерывные потоки доходов от услуг, связанных с контролем и использованием этих сенсорных систем вместо того, чтобы полагаться на традиционный розничный сбыт отдельных датчиков. Как уже упоминалось в главе 1, "Что такое интеллектуальные датчики, и зачем они нужны?", это может очень выгодно конечным пользователям и одновременно обеспечивать устойчивый доход поставщику и укреплять взаимодействие между продавцом и покупателем. В мире, в котором все больше возрастает товарная конкуренция, названный аспект дает поставщикам возможность выделить свои предложения, когда у покупателей не создается впечатление, что предлагаемые им товары и услуги — такие же, как и везде.

9.3. Резюме

Базовая предпосылка этой книги заключаются в том, что за интеллектуальными сенсорными системами — будущее огромного, постоянно расширяющегося рынка датчиков. А контроллер dsPIC создает превосходный фундамент для разработки самых разнообразных интеллектуальных сенсорных систем. По сути, функциональные возможности таких систем ограничены только воображением проектировщиков. Хотя создание интеллектуальных сенсорных систем требует от разработчика обширных познаний в целом ряде областей (формирование аналоговых сигналов, цифровая обработка сигналов и, отчасти, — цифровая схемо-

---

[65] Документ "Venue Segmentation Map for Intelligent Device Networking and Management" ("Карта сегментации для создания сети и администрирования интеллектуальных устройств") можно найти по адресу `www.harborresearch.com`.

техника[66]), dsPIC предоставляет холст, на котором творческий разработчик может создавать великолепные творения.

В конечном счете, как отметил Алан Кей во вводной цитате этой главы: "Лучший способ предсказать будущее — изобрести его". Теперь читатель может углубиться в конкретные проекты, задействовав в них полученные знания. Разработка интеллектуальных сенсорных систем — это не только прекрасное умственное упражнение, но еще и способ расширить понимание окружающего нас мира. Это может быть как коммерческая система, широко применяемая для производства миллиардов деталей по всему миру, так и устройство, которое улучшило жизнь всего лишь одному человеку, сделав для него мир более доступным с помощью интеллектуального протеза. Задействовав свою фантазию и свой опыт наряду с информацией, почерпнутой из этой книги, вы сможете действительно изменить мир к лучшему своими уникальными интеллектуальными датчиками.

[66] Это утверждение ни в коей мере не умаляет важность цифрового проектирования, а лишь констатирует тот факт, что значительная часть функциональности, который раньше требовал отдельных цифровых схем, теперь интегрирован непосредственно в кристалл dsPIC. Конечно же, для использования ЦКС dsPIC по-прежнему необходимы знания эффективных методов цифрового проектирования, однако степень системной интеграции уменьшила потребность в них.

A

Содержимое прилагаемого к книге компакт-диска

Прилагаемый к книге компакт-диск включает в себя Web-сайт со ссылками на ценные ресурсы Internet, а также исходные тексты программ и файлы проектов для трех приложений, которые разрабатывались в этой книге. Web-сайт находится в папке `Ресурсы`, и для его просмотра следует открыть в каком-либо браузере файл `index.htm`.

A.1. Web-сайт ресурсов

Web-сайт дает читателю возможность получить дополнительную информацию о рассмотренных в книге вопросах. Он содержит ссылки на справочные материалы, а также — на сайты поставщиков полезных компонентов и оборудования. Как и в случае любого другого сайта, содержащего указатели на внешние ресурсы, некоторые ссылки могут устареть и больше не действовать. Поиск такой информации в Internet можно осуществить по фразе из описания ссылки.

Обращаем внимание на то, что ссылки включены на компакт-диск только для того, чтобы помочь читателю быстро найти информацию — их не следует трактовать как поддержку конкретных изделий или производителей.

A.2. Исходные тексты для трех приложений

Полные исходные тексты для всех трех приложений, включая файлы проектов для использования с компилятором Microchip C, распределены по трем каталогам:

- `Коды\Термопара` — сенсорная система, использующая термопару;
- `Коды\Тензодатчик` — сенсорная система, использующая тензодатчик;
- `Коды\Поток` — сенсорная система, использующая датчик потока.

Все эти приложения требуют использования компилятора C версии 2.02 от компании Microchip, работающего под управлением интегрированной среды разработки MPLAB v7.40 от компании Microchip, пред-

назначенной для компиляции и компоновки программ (более поздние версии также приемлемы, но более ранние — нет). Программу MPLAB можно загрузить бесплатно с Web-сайта `www.microchip.com`. На этом же сайте бесплатно доступна учебная версия компилятора С.

Если необходимо изменить цифровые фильтры, используемые в одном из приложений, то придется приобрести копию программного пакета dsPIC Filter Design™, который также доступен на Web-сайте Microchip. Для запуска приложений понадобиться использовать универсальную макетную плату dsPICDEM 1.1 и встроенный отладчик ICD 2 для загрузки программных файлов в плату. Их тоже можно приобрести через сайт `www.microchip.com`.

Б

Инициализация ЦКС dsPIC и код запуска системы

Когда ЦКС dsPIC запускается после поступления сигнала сброса, он начинает выполнять программный код, на который ссылается вектор Reset (00000H). При этом все прерывания запрещены. Для правильной работы программный код должен перепрыгнуть через таблицу векторов прерываний и альтернативную таблицу векторов в пространство кода приложения и выполнить так называемый *код запуска*. Будучи критически важным компонентом микропрограммы, код запуска конфигурирует среду С таким образом, чтобы приложение (написанное в основном, если не полностью, на языке С) могло выполняться корректно. Этот шаг должен быть выполнен до обращения к программному коду на С, иначе могут возникнуть сложные для диагностики проблемы.

Поскольку код запуска инициализирует исполнительную среду С, он специфичен для используемого компилятора и распространяется, как часть компилятора. Для компилятора Microchip поставляются два файла программного кода запуска:

- `crt0.o` — инициализирует все инициализируемые данные значениями, считанными из памяти программ, а затем обнуляет все неинициализируемые данные;
- `crt1.o` — не инициализирует никаких данных.

Во всем остальном эти два модуля совершенно идентичны и выполняют следующие действия:[67]

- инициализируют указатель вершины стека (регистр W15) и регистр ограничителя указателя вершины стека (SPLIM), используя значения, сгенерированные компоновщиком;
- если определена секция `.const`, то код запуска отображает ее на окно видимости пространства программ, конфигурируя соответствующим образом регистры PSVPAG и CORCON;
- выполняют инициализацию данных (только `crt0.o`);

---

[67] Эта информация почерпнута из раздела "Startup and Initialization" руководства пользователя "MPLAB C30 C Compiler".

- вызывают точку входа `main()` для запуска пользовательского приложения.

Обратите внимание на то, что стандартный код запуска, поставляемый вместе с компилятором C от Microchip, не выполняет никакой аппаратной инициализации, кроме установки регистров, упомянутых выше. За конфигурирование аппаратных средств и любую другую программную настройку отвечает прикладной программный код.

В большинстве случаев код запуска "прозрачен" для прикладного программиста. В стандартные файлы компоновщика включены инструкции указать по адресу вектора Reset операторы `GOTO`, выполняющие переход к загрузочному коду. Там же представлены инструкции по компоновке со стандартным кодом запуска модуля `crt0.o`, который выполняет необходимую инициализацию системы C, а затем вызывает пользовательскую программу. Иногда, однако, приложение может потребовать немедленной конфигурации каких-то критических ресурсов системы. В этом случае пользователь может изменить файл исходной программы на языке ассемблера `crt0.s` или `crt1.s` для реализации соответствующих изменений. Если требуется такая модификация, то программист должен включить файл `crtx.s` (где `x` — это `0` или `1`) в список исходных файлов проекта, чтобы гарантировать его правильную компоновку в результирующем приложении.

B

Управляемый по прерываниям последовательный ввод-вывод

При обсуждении последовательной связи автор предполагает, что читатель имеет доступ к подпрограммам, которые надежно передают и получают данные, поэтому эта тема рассматривается на высоком уровне. Разработка надежного низкоуровневого интерфейса обмена данными — задача, решение которой зачастую далеко неочевидно, и которая изменяется в зависимости от процессора, даже если речь идет об устройствах одного производителя. Ключом к созданию надежной последовательной связи в реальном мире является использование буферизированной, управляемой по прерываниям структуры ввода-вывода. Рассмотрим, что же имеется в виду под этими труднопроизносимыми техническими терминами.

Структура ввода-вывода — это просто архитектура, которую мы используем для выполнения всех операций ввода и вывода при последовательном обмене данными. Придерживаясь этой структуры, мы сможем гарантировать, что устройство принимает и передает все данные, которые оно должно обработать.

Структура с управлением по прерываниям — это архитектура, в рамках которой мы разрешаем коммуникационному оборудованию прерывать работу процессора с тем, чтобы он немедленно приступил к обработке новых условий связи и вернулся к прерванной работе после завершения этой обработки. Если бы мы воспользовались альтернативным вариантом: архитектурой с опросом, в рамках которой процессор периодически опрашивает коммуникационное оборудование, чтобы узнать, должен ли он что-либо сделать, — то нам пришлось бы разработать специальный программный код, гарантирующий, что процессор завершит все другие задачи вовремя и сможет контролировать самые высокоскоростные коммуникационные события. Учитывая, что эти события могут наступать очень быстро и бессистемно, архитектура с опросом может оказаться большим ограничителем для всех коммуникационных каналов, исключая, разве что, самые медленные.

И, наконец, *буферизированная структура с управлением по прерываниям* — это архитектура, в рамках которой данные, подлежащие передаче между приложением и коммуникационным каналом, сохраняются в очереди. Доступ к этой очереди осуществляет подпрограмма обслуживания прерывания и управляющая прикладная программа таким образом, чтобы гарантировалось, что данные будут *когерентными* (т.е. точными и не вызывающими конфликт доступа со стороны подпрограмм обслуживания прерываний или приложения). Этот подход обеспечивает максимальную гибкость приложения и гарантирует высоконадежную связь с минимумом непроизводительной загрузки процессора.

В сердце такой структуры находится очередь данных, которая состоит из очереди приема данных, полученных устройством из других систем, и очереди передачи данных, которые будут посланы устройством другим системам. Очевидно, что объем данных, на обработку которых поступает запрос, может намного превышать ограниченную память любого процессора, поэтому очереди устроены как *кольцевые буферы*. Данные при такой организации добавляются последовательно до тех пор, пока не будет достигнут конец буфера, после чего новые данные начинают опять добавляться в первые ячейки буфера. Удаление данных из буфера происходит подобным же образом. Традиционно точка, в которой данные добавляются в буфер, называется "головой", а точка, из которой данные считываются из буфера, — "хвостом".

Для работы с очередями функции на уровне приложения и функции в подпрограммах обслуживания прерываний поддерживают два набора индексов, предназначенных для определения позиций буферов, которые должны использоваться следующими. Конкретное назначение каждого индекса зависит от уровня (приложение или прерывание) функции и очереди, с которой работает эта функция. Например, когда приложение передает данные в другую систему, функция уровня приложения *записывает* эти данные в очередь передачи и обновляет индекс, поддерживаемый приложением, сдвигая его на следующий байт в очереди передачи, в который можно записывать данные. Когда подпрограмма обслуживания прерывания, управляющая передачей данных, заметит наличие новых данных, она *считает* данные, записанные на прикладном уровне, обновит свой индекс, сдвинув его на следующий байт, подлежащий считыванию из очереди, а затем передаст данные аппаратным средствам связи. В этом случае функции уровня приложения отвечают за поддержку "головы" кольцевого буфера передачи данных, а функции подпрограмм обслуживания прерываний — за поддержку "хвоста" буфера. Для очереди приема данных — ситуация обратная: подпрограммы

обработки прерываний отвечают за поддержку "головы", а прикладные подпрограммы — "хвоста" буфера.

Этот подход работает хорошо до тех пор, пока не возникают какие-либо препятствия для одной или обеих очередей освобождаться так же быстро, как они заполняются. В этом случае "голова" буфера в конечном счете может догнать "хвост" — состояние, называемое *переполнением буфера*. Если дать буферу переполниться, то все данные между "хвостом" и "головой" будут утеряны. С учетом этого в рассмотренных реализациях было решено игнорировать данные, полученные после заполнения буфера. Однако, как только из буфера будет считан хотя бы один символ, прием данных возобновляется.

B.1. Псевдокод для структуры

Фактический программный код для последовательного интерфейса содержится в файлах `CommIF.c` и `CommIFDef.h`. Псевдокоды в нем обеспечивают высокоуровневые ссылки на задачи, выполняемым интерфейсом. Для того чтобы воспользоваться интерфейсом, приложение должно прежде инициализировать коммуникационное оборудование и связанные с ним глобальные переменные состояния, а также разрешить соответствующие прерывания процессора. Затем приложение может обмениваться данными с последовательными портами, используя представленные ниже функции.

B.2. Инициализация системы

Для инициализации коммуникационной системы пользователь должен вызвать функцию `CommInit()` с параметрами, задающими используемый модуль UART (1 или 2), а также — необходимые коммуникационные настройки (скорость передачи, тип четности, число стоп-битов).

Заголовок функции:

```
Uint16 CommInit (Uint8 ui8Port, Uint16 ui16BaudRate,
                 Uint16 ui16Parity, Uint16 ui16StopBits)
```

Пример

Инициализируем UART 1 для обмена данными со скоростью 19,2 Кбит/с без контроля четности и с одним стоп-битом.

```
CommInit(UART_1, 19200, PARITY_NONE, STOP_BITS_1);
```

B.3. Считывание данных из интерфейса

Для того чтобы считать данные из коммуникационного интерфейса, приложение вызывает функцию `CommGetRxPendingCount()`, которая определяет наличие данных, доступных для чтения. Если эта функция возвращает значение (количество ожидающих байтов данных) больше 0, то программа считывает следующий байт из очереди приема путем вызова функции `CommGetRxChar()` с параметром-указателем на восьмиразрядную ячейку для хранения считываемых данных. Если функция `CommGetRxChar()` возвращает значение состояния, отличное от нуля, то это означает, что произошла ошибка (полный перечень значений состояния находится в файле `StatusDef.h`), и приложение не должно использовать данные, возвращенные в буфер, заданный параметром.

Заголовки функций:

```
Uint16 CommGetRxPendingCount(void);
CommGetRxChar(Uint8 *pui8Data);
```

Пример

Проверить наличие данных, ожидающих чтения, и считать их

```
Uint8 ui8Data;  // Буфер для хранения данных
                // из очереди приема
if (CommGetRxPendingCount() > 0)
   CommGetRxChar(&ui8Data);
```

B.4. Запись данных в интерфейс

Для записи данных в коммуникационный интерфейс приложение вызывает функцию `CommPutChar()` для передачи одного байта данных или `CommPutBuff()` для передачи всего буфера. В любом случае, если функция возвращает значение состояния, отличное от нуля, то это означает, что произошла ошибка (см. файл `StatusDef.h` для ознакомления с полным списком значений состояния).

Заголовки функций:

```
Uint16 CommPutChar(Uint8 ui8Data) ;
Uint16 CommPutBuff(Uint8 *pui8Data, Uint16 ui16Length) ;
```

Пример

Передаем символ "d":

```
Uint8 ui8Data; // Буфер для хранения данных для
               // очереди передачи
ui8Data = 'd' ;
CommPutChar(ui8Data);
```

Пример

Передаем строку "dsPIC" (без символа конца строки):

```
CommPutChar("dsPIC", 5);
```

Издательство "МК-Пресс" представляет



Анна и Манфред Кёниг

Полное руководство по PIC-микроконтроллерам (+CD)

ISBN 966-8806-21-2
256 стр., мягкая обложка

На основании своего многолетнего опыта разработки проектов с использованием микроконтроллеров PIC, авторы рассматривают данную тему с точки зрения конструкторов-практиков. Затрагиваются все вопросы, связанные с периферийными устройствами, для взаимодействия с которыми, собственно, и предназначены микроконтроллеры. В частности, описаны аналоговые периферийные модули, инструментальная среда MPLAB, а также отладчик и демонстрационные платы.

Книга рассчитана на тех, кто уже имеет опыт работы с микроконтроллерами PIC и делает основное ударение на новых разработках последних лет. К ним, в первую очередь, относится серия микроконтроллеров PIC18, которая, благодаря своему 16-разрядному ядру, не только расширяет возможности программирования, но и открывает множество новых технических возможностей. Кроме того, в книге рассказано о многих нововведениях в микроконтроллерах PIC с 14- и 12-разрядным ядром.



Дитер Кохц

Измерение, управление и регулирование с помощью PIC-микроконтроллеров (+CD)

ISBN 966-8806-15-8
304 стр., мягкая обложка

Книга посвящена применению PIC-микроконтроллеров семейства PIC16C5X, а также PIC16C71 и PIC16F84 в схемах измерения, управления и регулирования. Рассмотрены следующие примеры: программируемый счетчик-частотомер, измерение температуры, реле времени, гигрометр с реле, температурное реле для регулирования нагрева, управление шаговым электродвигателем, регулирование уровня заполнения.

Книги издательства "МК-Пресс" можно заказать:
по адресу: **02002, г.Киев, а/я 294**; по телефону/факсу: **(044) 517-73-77**,
по e-mail: **info@mk-press.com**
или приобрести в магазине **"Микроника"** по адресу: **г.Киев, ул. М.Расковой, 13**
Посетите наш Internet-магазин: **http://www.mk-press.com**

Издательство "МК-Пресс" представляет



Вольфганг Трамперт

AVR-RISC микроконтроллеры (+CD)

ISBN 966-8806-07-7
464 стр., твердая обложка

Эта книга очень наглядно и подробно описывает архитектуру, внутреннее устройство, аппаратные ресурсы и систему команд микроконтроллеров семейства AVR. Цель книги — научить читателя проектировать собственные схемы с применением микроконтроллеров прогрессивной RISC-архитектуры, а также разрабатывать и тестировать программы для этих проектов. Кроме всего прочего, подробно рассматривается использование программатора для загрузки разработанных программ в память Flash-EPROM нового поколения микроконтроллеров. На прилагаемом к книге компакт-диске, кроме полнофункционального Ассемблера и эмулятора для отладки программ, находятся все исходные коды рассмотренных примеров, которые могут использоваться читателем в собственных проектах.



Вольфганг Трамперт

Измерение, управление и регулирование с помощью AVR микроконтроллеров (+CD)

ISBN 966-8806-14-X
208 стр., мягкая обложка

Книга описывает особенности применения AVR-микроконтроллеров в технике измерения, управления и регулирования. При этом основной акцент поставлен на измерении напряжения, выводе и отображении результатов измерений, а также на регулировании аналоговых напряжений. Изложенный материал дает возможность поэтапно проследить весь процесс разработки устройства, понять, почему программное и аппаратное обеспечение скомпоновано именно таким, а не каким-либо другим образом, и суметь в случае необходимости выполнить самостоятельную разработку.

Книги издательства "МК-Пресс" можно заказать:
по адресу: **02002, г.Киев, а/я 294**; по телефону/факсу: **(044) 517-73-77**,
по e-mail: **info@mk-press.com**
или приобрести в магазине **"Микроника"** по адресу: **г.Киев, ул. М.Расковой, 13**
Посетите наш Internet-магазин: **http://www.mk-press.com**

Издательство "МК-Пресс" представляет



Марти Браун

Источники питания. Расчет и конструирование

ISBN 966-8806-01-8
288 стр., мягкая обложка

В книге поэтапно показано проектирование широкого набора источников питания. С ее помощью каждый, кто обладает базовыми познаниями в области электроники, сможет проектировать и создавать сложные источники питания. Кроме общих подходов к конструированию, представлено проектирование линейных, импульсных и квазирезонансных источников питания. Сложные вопросы, наподобие магнетизма и контроля электромагнитных помех, разъясняются простым и доступным языком. Издание является практическим пособием, организованное таким образом, чтобы как опытные, так и начинающие разработчики могли быстро найти и применить требуемую им информацию. В книгу включена информация о проектировании выходных каскадов, о выборе ИС контроллера и других функциях, имеющих отношение к источникам питания, наподобие управления импульсными источниками питания, потери электрического сигнала и др. Кроме того, рассмотрены методики формирования сигналов, уменьшения основной потери, демпферы и квазирезонансные преобразователи.



Вовк П.Ю.

Зарубежные электромагнитные реле

ISBN 966-96415-0-0
400 стр., мягкая обложка

Книга представляет собой справочное пособие по зарубежным электромагнитным реле. За основу взят материал о сигнальных, промышленных и высокочастотных реле Fujitsu, на основании которых рассматривается общая теория и практика применения электромагнитных реле. В книге представлены таблицы перекрестных ссылок для аналогов всех основных зарубежных производителей реле рассматриваемых типов.

Книги издательства "МК-Пресс" можно заказать:
по адресу: **02002, г.Киев, а/я 294**; по телефону/факсу: **(044) 517-73-77**,
по e-mail: **info@mk-press.com**
или приобрести в магазине **"Микроника"** по адресу: **г.Киев, ул. М.Расковой, 13**
Посетите наш Internet-магазин: **http://www.mk-press.com**

Издательство "МК-Пресс" представляет



Швец В.А.

Одноплатные микроконтроллеры. Проектирование и применение

ISBN 966-96415-4-3
304 стр., мягкая обложка

На базе микропроцессоров КР1810 и К1816 представлены основные методы проектирования одноплатных микроконтроллеров для информационных и управляющих систем. Подробно рассматриваются вопросы проектирования узлов одноплатных микроконтроллеров, программного обеспечения, применение микропроцессоров в цифровой обработке информации и измерительных устройствах: частотомерах; фазометрах; вольтметрах; фильтрах. В примерах представлена реализация вычислительных алгоритмов на Ассемблере.



Шпак Ю.А.

Программирование на языке С для AVR и PIC микроконтроллеров (+CD)

ISBN 966-8806-16-6
400 стр., твердая обложка

В книге рассмотрено программирование на языке С микроконтроллеров AVR с использованием компилятора WinAVR, а также микроконтроллеров PIC с использованием компилятора CCS-PICC. Кратко рассмотрена архитектура и аппаратное обеспечение микроконтроллеров AVR и PIC. Дано описание средств программной разработки в среде WinAVR и CCS-PICC, включая эмуляцию программ с помощью AVR Studio и MPLAB. Кратко рассмотрен стандартный синтаксис и директивы препроцессора языка С, а также особенности программирования на этом языке для микроконтроллеров. Книга содержит множество программных примеров на С, а также справочник с описанием системы ассемблерных команд микроконтроллеров AVR и PIC.

Книги издательства "МК-Пресс" можно заказать:
по адресу: **02002, г.Киев, а/я 294**; по телефону/факсу: **(044) 517-73-77**,
по e-mail: **info@mk-press.com**
или приобрести в магазине **"Микроника"** по адресу: **г.Киев, ул. М.Расковой, 13**
Посетите наш Internet-магазин: **http://www.mk-press.com**

Издательство "МК-Пресс" представляет



Авраменко Ю.Ф.

Мощные биполярные транзисторы для импульсных источников питания, TV-приемников и мониторов

ISBN 966-8806-17-4
544 стр., мягкая обложка

В справочнике представлены электрические параметры на мощные биполярные транзисторы, имеющие высокую скорость переключения. Данные приборы применяются в импульсных источниках питания различного назначения, в промышленном оборудовании, в бытовой и профессиональной видео- и аудиотехнике. Указаны технические данные на изделия следующих ведущих производителей полупроводниковых приборов: FAIRCHILD, HITACHI, MOTOROLA (ON SEMICONDUCTOR), PANASONIC, PHILIPS, SANKEN, SAMSUNG, SANYO, SHINDENGEN, ST-MICROELECTRONICS, TOSHIBA и ZETEX. Таблица аналогов полупроводниковых приборов составлена на основании руководства Master Replacement Guide.



Тяпичев А.Г.

Персональный компьютер в радиолюбительской практике

ISBN 966-8806-18-2
400 стр., мягкая обложка

Книга предназначена для любознательного читателя и любителей мастерить своими руками. В ней описаны различные варианты специального использования персонального компьютера для выполнения «нетрадиционных» для него работ, таких как управление различными удаленными аппаратами, кодирование и декодирование различных сигналов, создание принципиальных электрических схем и проверка работоспособности этих схем, создание звуковых эффектов и многое другое. Большое внимание уделено процессу программирования микроконтроллеров. Даны описания специальных компьютерных программ, и подробное описание аппаратов, которые могут подключаться к компьютеру и применяться в совместной с ним работе.

Книги издательства "МК-Пресс" можно заказать:
по адресу: **02002, г.Киев, а/я 294**; по телефону/факсу: **(044) 517-73-77**,
по e-mail: **info@mk-press.com**
или приобрести в магазине **"Микроника"** по адресу: **г.Киев, ул. М.Расковой, 13**
Посетите наш Internet-магазин: **http://www.mk-press.com**

Издательство "МК-Пресс" представляет



Безверхний И.Б.

TV-приемники на шасси VESTEL 11AKxx (+CD)

ISBN 978-966-8806-11-7
304 стр., мягкая обложка

В данной книге рассмотрены телевизионные приемники на основе шасси 11AK20, 11AK30 11AK36 и PT92 турецкой компании VESTEL. Сегодня на базе этих шасси десятки различных производителей поставляют на рынок под различными торговыми марками сотни моделей телевизоров. Подробно описаны: элементная база, схемы, сервисные режимы, регулировка и ряд вопросов практического характера. Отдельная глава посвящена современным видеопроцессорам производства PHILIPS. В приложении представлены справочные данные на современные видеопроцессоры различных производителей.



Конахович Г.Ф., Бабак В.П., Фисенко В.М.

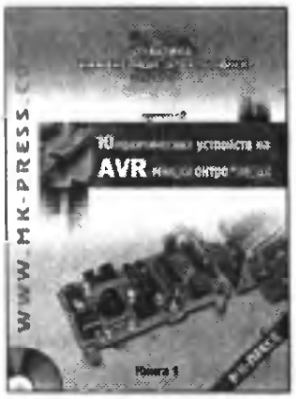
Специальный радиомониторинг

ISBN 978-966-8806-37-7
584 стр., мягкая обложка

С возрождением рынка и появлением конкуренции появилось такое отсутствующее ранее явление, как промышленный шпионаж, который, прежде всего, представляет собой несанкционированный доступ к коммерчески важной информации с помощью разведывательных технических систем негласного ее съема, скрытно установленных в помещениях интересующих объектов. Книга посвящена специальному радиомониторингу — контролю всех радиоизлучений в зоне расположения объекта. Основное внимание уделяется вопросам организации работ по РМ, обработке получаемых в процессе его проведения результатов, рассмотрению параметров и характерных признаков различных источников радиоизлучений, оптимальному подбору аппаратуры, необходимой для эффективного ведения радиомониторинга.

Книги издательства "МК-Пресс" можно заказать:
по адресу: **02002, г.Киев, а/я 294**; по телефону/факсу: **(044) 517-73-77**,
по e-mail: **info@mk-press.com**
или приобрести в магазине "**Микроника**" по адресу: **г.Киев, ул. М.Расковой, 13**
Посетите наш Internet-магазин: **http://www.mk-press.com**

Издательство "МК-Пресс" представляет



Кравченко А.В.

10 практических устройств на AVR-микроконтроллерах. Книга 1 (+CD)

ISBN 978-966-8806-41-4
224 стр., мягкая обложка

Данная книга открывает серию сборников с практическими примерами применения микроконтроллеров. В ней рассмотрены десять завершенных устройств на базе микрокон-троллеров AVR, которые можно легко собрать в домашних условиях и применять в быту или профессиональной деятельности: генератор световых эффектов; счетчик событий; музыкальный звонок; индикатор уровня звука; повышающий преобразователь, схема управления шаговым двигателем; цифровой термометр и др.

Благодаря подробному анализу аппаратной и программной части устройств, книга будет интересна и полезна как начинающим, так и опытным радиолюбителям, желающим изучить методы эффективного применения микроконтроллеров.



Авраменко Ю.Ф.

Качественный звук — сегодня это просто

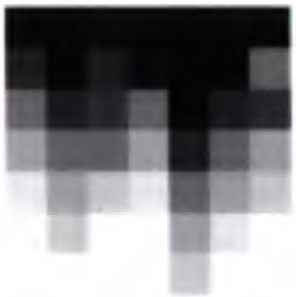
ISBN 966-8806-27-1
286 стр., мягкая обложка

В книге максимально подробно приведены все рекомендации разработчиков – инженеров NSC, как правильно построить усилительный тракт на основе мощных ОУ. Современный подход, основанный на рекомендациях инженеров AD и TI, к топологии печатной платы, к выбору «правильных» пассивных компонентов для звуковоспроизводящего тракта поможет реализовать основной принцип: как можно меньше ухудшить качество записи. Большое количество примеров построения качественных УМЗЧ будет наглядным пособием для реализации собственной конструкции в короткие сроки с небольшими материальными затратами и главное, с предсказуемым результатом.

Книги издательства "МК-Пресс" можно заказать:
по адресу: **02002, г.Киев, а/я 294**; по телефону/факсу: **(044) 517-73-77**,
по e-mail: **info@mk-press.com**
или приобрести в магазине "**Микроника**" по адресу: **г.Киев, ул. М.Расковой, 13**
Посетите наш Internet-магазин: **http://www.mk-press.com**

Крид Хадлстон

ПРОЕКТИРОВАНИЕ ИНТЕЛЛЕКТУАЛЬНЫХ ДАТЧИКОВ С ПОМОЩЬЮ Microchip dsPIC



Научитесь проектировать интеллектуальные датчики с помощью самых популярных в мире микросхем!

В этой книге читатель найдет:

- Подробное описание методики проектирования, практические примеры, подробные схемы и применимый программный код
- Главы, посвященные практической разработке сенсорных систем, в которых задействованы датчики температуры, давления/нагрузки и расхода
- Компакт-диск с программными моделями на языке С и ассемблере.

Интеллектуальные датчики совершили революцию в сфере системного проектирования во всех областях применения: от спортивных автомобилей до сборочных конвейеров. Эти новые датчики обеспечивают возможности, немыслимые для их предшественников. Они не только точно и эффективно замеряют нужные параметры, но и способны обрабатывать и интерпретировать результаты измерений, преобразуя "сырые" данные в полезную информацию.

На страницах этой книги эксперт в области системного проектирования Крид Хадлстон обучает способам применения наиболее популярных на рынке цифровых контроллеров сигналов Microchip dsPIC, в которых вычислительный потенциал мощных цифровых процессоров сигналов удачно объединен с возможностями вездесущих микроконтроллеров PIC. Практика показала, что цифровые контроллеры сигналов - оптимальный способ реализации интеллектуальной обработки данных в датчиках, поскольку они объединяют в себе возможности управления, характерные для микроконтроллеров, с быстродействием цифровых процессоров сигналов. Благодаря конкретным примерам и подробной информации о реальных сенсорных системах, книга "Проектирование интеллектуальных датчиков с помощью Microchip dsPIC" позволяет разработчикам легко получить точный программный код даже в сжатые термины технологического цикла проектирования систем.

В отличие от многих других изданий, посвященных встроенным системам, в этой книге рассмотрены вопросы не только программирования, но и проектирования электронного оборудования. Таким образом, читатель получает полное представление о процессе создания интерфейса для трех конкретных типов датчиков: температуры, давления/нагрузки и расхода. Эта практичная, легкая в восприятии книга раскрывает реальные проблемы, возникающие в повседневной работе разработчиков, и показывает решения, позволяющие реализовать все сильные стороны такого мощного средства, как интеллектуальные датчики.

ELSEVIER

ДМК Пресс

WWW.MK-PRESS.COM

ISBN-13: 978-966-8806-38-4

9 789668 806384