Питер Блюм

LabVIEW: стиль программирования

Peter A. Blume

The LabVIEW Style Book

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Питер Блюм

LabVIEW: стиль программирования

Москва, 2008

УДК 621.38
ББК 32.973.26-108.2
Б 68

Б68 **Блюм П.**
LabVIEW: стиль программирования. Пер. с англ. под ред. Михеева П.– М.: ДМК Пресс, 2008 – 400 с. : ил.

ISBN 978-5-94074-444-3

Автор книги предлагает практические советы по улучшению каждой грани вашего приложения, созданного на LabVIEW: эффективности, удобочитаемости, простоты работы, использования и поддержки, производительности и надежности. Блюм подробно объясняет каждое правило, иллюстрирует их жизненными примерами. Есть даже примеры «от противного»: что именно не надо делать и почему.

Издание должно стать настольным справочником каждого специалиста, работающего в среде LabVIEW. Эта книга необходима каждому, кто хочет добиться высокого качества разработки программного обеспечения в среде LabVIEW: менеджерам, разработчикам и конечным пользователям.

УДК 621.38
ББК 32.973.26-108.2

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc. RUSSIAN language edition published by DMK PUBLISHERS, Copyright © 2007.

Все права защищены. Любая часть этой книги не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами без письменного разрешения владельцев авторских прав.

Материал, изложенный в данной книге, многократно проверен. Но поскольку вероятность технических ошибок все равно существует, издательство не может гарантировать абсолютную точность и правильность приводимых сведений. В связи с этим издательство не несет ответственности за возможные ошибки, связанные с использованием книги.

ISBN 978-0-13-145835-2 (англ.)
ISBN 978-5-94074-444-3 (рус.)

© Pearson Education, Inc., 2007
© Перевод на русский язык, оформление, ДМК Пресс, 2008

СОДЕРЖАНИЕ

Введение 11

Предисловие 13
Обращение к читателю 13
Организация 14
Соглашение о приоритете правил 15
Примечания 15

Благодарности 16
Об авторе 18

▼ 1

Зачем нужен стиль 19
1.1. Зачем нужен стиль? 19
1.1.1. Простота использования 24
1.1.2. Эффективность 25
1.1.3. Читабельность 27
1.1.4. Простота поддержки 29
1.1.5. Надежность 30
1.1.6. Простота 34
1.1.7. Производительность 34
1.1.8. Поддержание стиля 36
1.2. Стиль или быстрота? 37

▼ 2

Подготовка к хорошему стилю 39
2.1. Техническое задание 39

2.1.1. Советы по документации ... 42
2.1.2. Проектная документация в LabVIEW ... 44
2.2. Проектирование ... 47
2.2.1. Поиск полезных источников ... 48
2.2.2. Разработка пробной версии ... 49
2.2.3. Вернитесь к ТЗ ... 50
2.3. Настройка среды LabVIEW ... 51
2.3.1. Диалоговое окно опций LabVIEW ... 51
2.3.2. Повторное использование кода ... 53
2.4. Структура проекта, именование файлов и управление ... 57
2.4.1. Расположение файлов ... 58
2.4.2. Проект LabVIEW ... 62
2.4.3. Именование файлов ... 64
2.4.4. Управление исходниками ... 65
Ссылки ... 66

3

Стиль лицевой панели ... 67
3.1.Расположение ... 68
3.1.1. Общие правила ... 69
3.1.2. Панель пользовательского ВП ... 71
3.1.3. Лицевая панель подприбора ... 79
3.2. Текст ... 80
3.2.1. Общие правила ... 80
3.2.2. Метки элементов ... 85
3.2.3. Текст в подприборах ... 88
3.2.4. Текст промышленных ВП ... 88
3.3. Цвет ... 90
3.4. Навигация по приложению ... 92
3.4.1. Элементы управления ... 92
3.4.2. Ваш стиль ... 95
3.5. Примеры ... 96
3.5.1. ВПП из блок-диаграммы ... 96
3.5.2. Служебный диалог ... 98
3.5.3. Тестирование и сортировка конденсаторов ... 99
3.5.4. Центрифуга ... 101
3.5.5. Спектральный анализатор ... 103
3.5.6. Интерфейс управления парапланом ... 104
Ссылки ... 105

4

Блок-диаграмма ... 106
4.1. Расположение ... 107
4.1.1. Основные части ... 107
4.1.2. Блоки в ВПП ... 108
4.2. Соединения ... 112
4.2.1. Секреты аккуратного соединения ... 112
4.2.2. Использование кластеров ... 116
4.3. Поток данных ... 120
4.3.1. Основы потока данных ... 120
4.3.2. Когда нужны переменные и последовательности ... 124
4.3.3. Здесь переменные и последовательности не нужны ... 126
4.3.4. Оптимизируем поток данных ... 131
4.4. Примеры ... 136
4.4.1. ВПП из участка кода ... 136
4.4.2. ВП Excessively Nested ... 137
4.4.3. ВП Haphazard ... 139
4.4.4. ВП Right to Left ... 140
4.4.5. ВП Left to Right ... 141
4.4.6. ВП Centrifuge DAQ ... 142
4.4.7. ВП Screw Inspection ... 143
4.4.8. ВП Optical Filter Test ... 144
Ссылки ... 147

5

Иконка и контакты ... 148
5.1. Иконка ... 150
5.1.1. Основные правила ... 151
5.1.2. Хитрости создания иконок ... 154
5.1.3. Международные иконки ... 158
5.2. Соединительная панель ... 159
5.3. Примеры ... 165
5.3.1. Доказательства от противного ... 165
5.3.2. Драйверы приборов ... 167
5.3.3. Разные примеры ... 171
5.3.4. Показательные примеры ... 173
Ссылки ... 175

6

Структуры данных 176
6.1. Методология разработки конструкций данных 177
6.1.1. Выбор элементов управления и типов данных 177
6.1.2. Настройка свойств 189
6.1.3. Создание конструктов данных 190
6.2. Простые типы данных 193
6.2.1. Логические переменные 193
6.2.2. Численные элементы 196
6.2.3. Специальные численные данные 198
6.2.4. Строка, путь и изображение 201
6.3. Конструкты данных 203
6.3.1. Простые массивы и кластеры 203
6.3.2. Специальные конструкты данных 210
6.3.3. Вложенные структуры данных 211
6.4. Примеры 217
6.4.1. ВП Thermometer 217
6.4.2. Вариант OpenG 219
6.4.3. Случайные данные 222
6.4.4. ВП Centrifuge DAQ 222
Ссылки 225

7

Обработка ошибок 226
7.1. Основы обработки ошибок 227
7.1.1. Отслеживание ошибок 228
7.1.2. Отчеты об ошибках 234
7.1.3. Коды ошибок 239
7.2. Обработка ошибок в ВПП 241
7.3. Определение приоритетов ошибок 246
7.4. Советы по обработке ошибок 252
7.4.1. Соединение структур 252
7.4.2. Слияние ошибок 252
7.4.3. Очищение ошибок 254
7.4.4. Автоматическая обработка ошибок 256
7.5. Примеры 257
7.5.1. Постоянное получение данных и запись в файл 257
7.5.2. Suss Interface Toolkit 258

7.5.3. Слияние параллельных ошибок 259
7.5.4. ВП Screw Inspection 260
7.5.5. ВП Test Executive 262
Ссылки 265

8

Шаблоны 266
8.1. Простые шаблоны 268
8.1.1. Шаблон ВПП Immediate 268
8.1.2. Шаблон Functional Global 271
8.1.3. Шаблон Continuous Loop 272
8.1.4. Цикл с обработкой событий 278
8.2. Конечные автоматы 283
8.2.1. Классический конечный автомат 287
8.2.2. Конечный автомат с очередью 288
8.2.3. Событийно управляемый конечный автомат 292
8.2.4. Автомат событий 295
8.3. Составные шаблоны 298
8.3.1. Параллельные циклы 300
8.4. Объектные структуры сложных приложений 304
8.4.1. Динамическая объектная структура 304
8.4.2. Объектная структура приложения со многими циклами 311
8.4.3. Модульная объектная структура приложения со многими циклами 316
8.5. Примеры 320
8.5.1. ВП Elapsed Time 320
8.5.2. ВП Poll Instrument Response 321
8.5.3. Нетрадиционный конечный автомат 323
8.5.4. ВП Centrifuge DAQ 326
8.5.5. Утилита управления датчиком 327
8.5.6. Распределенная управляющая система 328
Ссылки 330

9

Документация 331
9.1. Документация лицевой панели 333
9.2. Блок-диаграмма 337
9.3. Описание иконки и ВП 345

9.4. Online-документация 346
9.5. Примеры 350
9.5.1. ВПП из участка блок-диаграммы 350
9.5.2. ВП Filter Test 350
9.5.3. Тщательное описание элементов управления 351
9.5.4. Профиль температуры 352
Ссылки 353

▼ 10

Экспертная оценка программы 354
10.1. Самостоятельная экспертная оценка программы 355
10.1.1. ВП Analyzer Toolkit 355
10.1.2. Контрольный список ручной проверки 366
10.2. Экспертные проверки 370
Ссылки 375

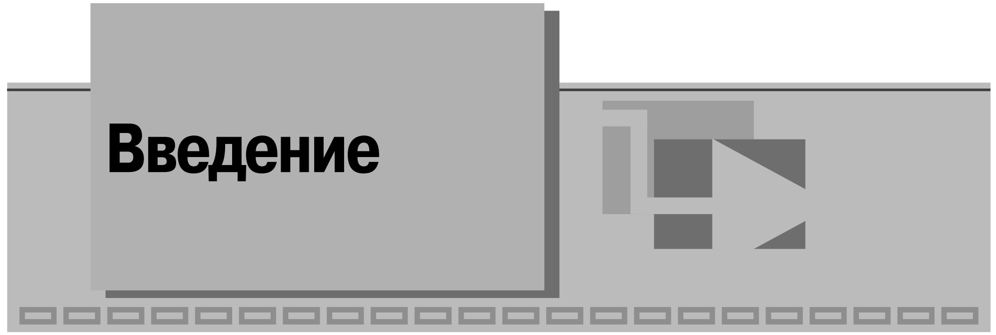
▼ Приложение А

Глоссарий 376

▼ Приложение Б

Сводка основных правил стиля 388

Предметный указатель 397



Введение

В течение длительного времени я работал над созданием ВП в R&D-отделе LabVIEW в National Instruments. Во время первой недели моего обучения в LabVIEW я решил написать LabVIEW-версию моей любимой карточной игры «Set». Мне потребовалось немало времени, но в конце концов я закончил ее, и это был, вероятно, самый ужасный код LabVIEW, который вы когда-либо видели. Как сказал бы Питер Блюм (Peter Blume), мой код был образцом «Спагетти». Хуже того, на лицевой панели было множество ярких, кричащих элементов неоновых цветов. Мой код функционировал, но его было невозможно использовать и уж тем более поддерживать. Спустя несколько лет, когда я стал куда лучше разбираться в LabVIEW, я попытался добавить к старому коду несколько новых функций, но быстро сдался, потому что у меня не было ни малейшего понятия, как работает мой код.

Много лет спустя я могу с уверенностью сказать, что мой стиль программирования улучшался не по дням, а по часам. Однако я также без сомнения могу сказать, что все мои проблемы со стилем программирования на LabVIEW в то время исчезли бы, если бы тогда существовала книга «Стиль LabVIEW». Эта книга полностью самосогласованный источник информации, затрагивающий каждый аспект стиля программирования ВП от верхних уровней (прогнозирование и планирование) до мелких деталей (проводники данных со слишком большим количеством изгибов). Если бы я прочел эту книгу во время изучения LabVIEW, мой код было бы во много раз проще использовать и поддерживать с самого начала.

Книгу «Стиль LabVIEW» необходимо прочитать каждому разработчику программного обеспечения на LabVIEW. Не только потому, что она содержит основные правила стиля для новичков, в ней также находятся необходимые обновления для ветеранов программирования. В частности, в главах 6 «Структуры данных», 7 «Обработка ошибок», 8 «Шаблоны» обсуждаются ключевые моменты и техники программирования, недоступные в других источниках. В качестве одного из адвокатов стиля ВП в LabVIEW R&D я настоятельно рекомендую всем новым разработчикам в моей команде прочитать эту книгу, и я уверен, что буду не раз к ней обращаться в ходе дискуссий с моими более опытными коллегами.

Другой поистине уникальной особенностью этой книги является впечатляющее количество примеров, которые Питер использует для иллюстрации хорошего

(а иногда и плохого) стиля программирования. За свой пятнадцатилетний опыт программирования на LabVIEW Питер накопил огромную библиотеку ВП, которые написали он, его работники и клиенты. И он использует эту библиотеку, чтобы проиллюстрировать подходящие моменты в каждой главе. Более того, иногда он рассматривает один пример на протяжении нескольких глав и постепенно применяет к нему все новые и новые правила стиля. Таким образом, мы можем «в реальном времени» видеть, как хороший стиль положительно влияет на разработку ВП.

В то время как LabVIEW отмечает 20-летие, вдохновляя инженеров, ученых и даже детей по всему миру, я абсолютно уверен, что читатели этой книги по стилю LabVIEW оценят то невероятное количество времени и усилий, которое они сэкономят, разрабатывая ВП по правилам хорошего стиля. Так что, если вы новичок в LabVIEW, приготовьтесь к тому, что ваши ВП станут – как сказал бы Питер – «внушающими благоговейный ужас», а если вы эксперт в LabVIEW, то вы тоже узнаете много нового. Как и я!

Дарен Наттингер
Штатный разработчик программного обеспечения, LabVIEW R&D
National Instruments Corporation.

Дарен Наттингер (Darren Nattinger) проработал в National Instruments восемь лет. Сейчас он является ведущим разработчиком VI Analyzer Toolkit и Report Generation Toolkit for Microsoft Office. Он также был рецензентом книги «Стиль LabVIEW».

Предисловие

Книга «Стиль LabVIEW» содержит правила стиля, направленные на улучшение производительности, читабельности, эффективности; упрощение технической поддержки приложений в LabVIEW. В книге приведены объяснения всех правил, с примерами и иллюстрациями. В книге используются работы пионеров LabVIEW-сообщества, эволюционировавшие за годы использования в Bloomy Controls, которые были отрецензированы уважаемыми членами LabVIEW-сообщества. Я призываю вас учиться на моем опыте и опыте сотрудников Bloomy Controls, читая книгу «Стиль LabVIEW». Я надеюсь, что вы получите такое же удовольствие от чтения этой книги, какое получил я, когда писал ее!

Обращение к читателю

Предполагаемые читатели этой книги – это, прежде всего, разработчики, менеджеры и организации, занимающиеся разработкой приложений на LabVIEW. Вы должны обладать практическим знанием фундаментальных принципов LabVIEW и терминологии на уровне курсов LabVIEW Basics I и II, кроме того, необходим опыт разработки и внедрения приложений. Опытные новички могут использовать эту книгу, чтобы сформировать хорошие навыки программирования на LabVIEW в самом начале своей карьеры. Более опытные разработчики, которые овладели основами и готовы перейти на новый уровень, получат наибольшую пользу от прочтения этой книги. Без сомнения, вы уже познали мощь и гибкость LabVIEW и готовы сконцентрироваться на стиле. Продвинутые разработчики подкрепят свои знания и опыт, получат полезную тему для обсуждения с коллегами. Вы можете использовать книгу «Стиль LabVIEW», чтобы уменьшить затраты сил на обучение и поддержку внутри организации. Менеджеры и организации, которые нанимают множество разработчиков и пользователей, могут получить максимум прибыли, приняв данные правила стиля как стандарт для всей организации. Этот подход гарантирует качество и согласованность во всей организации и поможет вам удовлетворить всем стандартам качества.

Организация

Главы в книге «Стиль LabVIEW» содержат правила стиля и примеры, разбитые по темам. В главе 1 «Зачем нужен стиль» обсуждается связь между стилем и простотой использования, эффективностью, читабельностью, простотой поддержки, устойчивостью к ошибкам и производительностью. В главе 2 «Приготовка к хорошему стилю» представлены некоторые соображения, влияющие на стиль еще до того, как вы начнете программировать, к ним относятся спецификация, настройка среды LabVIEW, организация проектов и файлов. Кроме того, в ней представлен специализированный стандарт проектной спецификации в LabVIEW. В главах 3 «Стиль лицевой панели», 4 «Блок-диаграмма» и 5 «Иконка и контакты» рассмотрены основы разработки ВП. В главе 3 находятся правила, касающиеся компоновки, текста, цвета и навигации. В ней подчеркивается различие между правилами для лицевых панелей ВП с графическим интерфейсом и для ВПП. Правила в главе 4 относятся к расположению, соединению элементов и организации потока данных наряду с методами оптимизации потока данных. В главе 5 обсуждается опыт создания хороших иконок и настройки ярлыков, в том числе затрагиваются вопросы шаблонов соединительной панели, назначений и общепринятых норм.

В главе 6 «Структуры данных» представлены правила выбора типа данных и работы с массивами и кластерами. В этой главе также находится несколько таблиц для упрощения выбора типа данных. В этой главе также представлены правила и примеры оптимизации ВП с использованием сложных структур данных. Главы 7 «Обработка ошибок», 8 «Шаблоны», 9 «Документация» выходят за рамки базовых понятий. В главе 7 представлены согласованные правила для последовательной обработки ошибок в приложении, наряду с некоторыми специфичными моментами обработки ошибок внутри ВПП. В главе 8 обсуждается обычная архитектура ВП, обеспечивающая хороший стиль, начиная с простых шаблонов ВПП и кончая шаблонами со многими циклами. В этой же главе обсуждается несколько вариантов конечного автомата в LabVIEW. Кроме того, в главе 8 представлены 3 сложные структуры приложений, включая динамическую структуру, использующую плагины; структуру со многими циклами и модульную структуру со многими циклами, использующую цикличные ВПП. В главе 9 представлены правила по документированию вашего исходного кода, включая лицевую панель, блок-диаграмму, иконку и описание ВП. Помимо этого, обсуждается создание и размещение документов on-line. В главе 10 «Экспертная оценка программы», предложено несколько методов проверки исходного кода и усиления правил стиля, включая самопроверки по собственному контрольному списку, автоматические проверки с использованием LabVIEW VI Analyzer Toolkit и проверки другими специалистами. Рассматривается эволюция приложения на каждой стадии.

Приложение включает глоссарий и суммарный свод правил. В Приложении А «Глоссарий» находится список терминов и определений, многие термины в LabVIEW эволюционируют и зависят от контекста. Каждому появляющемуся в тексте специфическому термину или термину, который можно воспринять неоднозначно, дается определение, и в последующих главах он используется уже именно в этом значении. Для удобства все определения собраны в глоссарии. В Приложении Б «Сводка основных правил стиля» представлены все правила стиля из каждой главы книги.

Соглашение о приоритете правил

Во всей книге «Стиль LabVIEW» используется два уровня приоритета правил, которые различаются шрифтом (полужирный курсив или просто курсив).

Правила с высшим приоритетом – это законы, которые должны соблюдаться всегда, за редким исключением. Они выделены жирным курсивом:

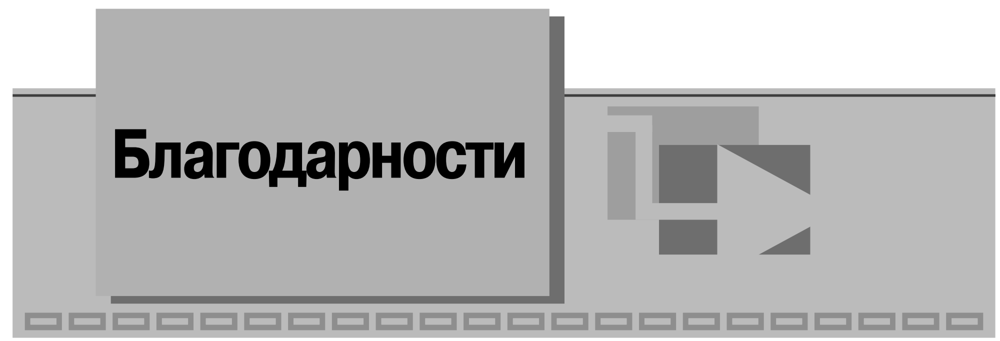
Правило 2.2 Составьте список требований, запишите техническое задание

Правила с обычным приоритетом носят рекомендательный характер, следовать которым считается хорошим тоном, но они не так критичны, как правила с высоким приоритетом, и допускают больше исключений. Они выделены простым курсивом:

Правило 2.3 Придерживайтесь хорошего стиля программирования в LabVIEW

Примечания

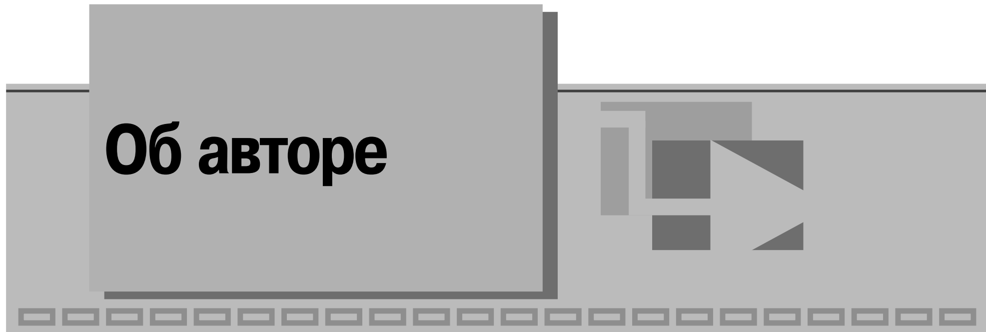
1. В разделе «Благодарности» приведен список рецензентов и людей, внесших свой вклад в развитие стиля LabVIEW.
2. Bloomy Controls является партнером National Instruments с офисами в Виндзоре, Коннектикуте, Милфорде, Массачусетсе; и Форте Ли, Нью Джерси. Более подробная информация доступна на сайте www.bloomy.com.
3. LabVIEW Basic I и II – это недельные курсы, предлагаемые сертифицированными центрами обучения NI. Более подробная информация доступна на сайте www.ni.com/training и LabVIEW.ilc.edu.ru.



Я бы хотел поблагодарить следующих рецензентов и разработчиков за ключевой код для иллюстраций:

Darren Nattinger, National Instruments
Greg Burroughs, Bloomy Controls, Inc.
John Compton-Smith, Dover Technology Limited
Crystal Drumheller, National Instruments
Greg McKaskle, National Instruments
Brian Powell, National Instruments
Heather Eisenbraun, National Instruments
Bob Hamburger, Bloomy Controls, Inc.
James Fowler, Bloomy Controls, Inc.
Bart Craft, National Instruments
Alex Khazanov, Harris Corporation
Robert Cornwell, Bloomy Controls, Inc.
Keith Brainard
Anthony Conaci, CiDRA
Robert Gough, JDS Uniphase
Ernie St. Louis, Ciencia
Jim Kring, James Kring, Inc.
Ken Tumidajski, Testand Corporation
Alan Blankman, LeCroy Corporation
Daniel L. Press, Prime Test Corporation
Danny Allard, Videotron
Brian Gangloff, DataAct Incorporated
Dave Galanis, UTC Power
Roger Emerick, Pioneer Aerospace Corp.
Robert Greene, NSK
Robert Breidenthal, Precision Optics

Кроме того, я хотел бы поблагодарить пионеров стиля LabVIEW. Мег Кей (Meg Kay) и Гарри Джонсон (Gary Johnson) написали руководство по стилю LabVIEW, которое затем переросло в *Рекомендации по разработке приложений в LabVIEW от National Instruments*, часть поставляемой с LabVIEW документации. Ренди Джонсон (Randy Johnson) написал *Правила соединения (Rules to Wire By)*, которые были напечатаны в информационном бюллетене *LabVIEW Technical Resource* в 1999 г. Недавно Дарен Наттингер из NI разработал LabVIEW VI Analyzer Toolkit – дополнение к LabVIEW, которое автоматически анализирует стиль ВП. Наконец, Ноэль Адроно (Noel Adrono) написал *Рекомендации по разработке драйверов устройств (Instrument Driver Development Guidelines)*, которые можно бесплатно скачать на сайте www.ni.com.



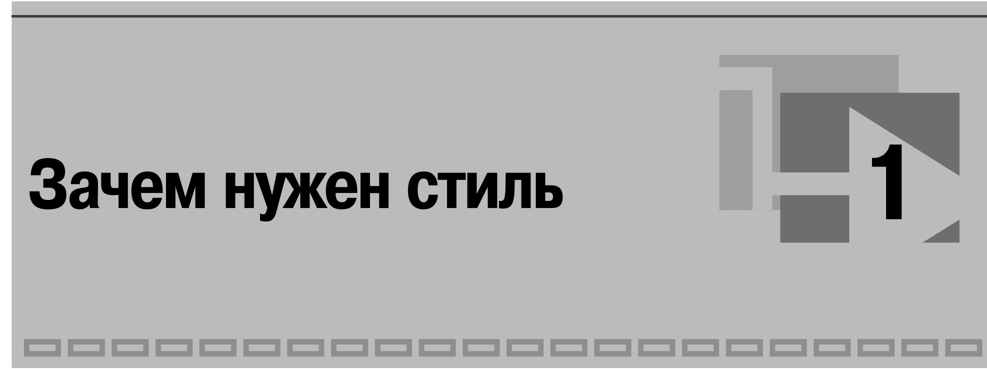
Об авторе

Питер Блюм (Peter Blume) – основатель и президент компании Bloomy Controls, Inc, партнере National Instruments, которая занимается разработкой приложений на основе LabVIEW. Начиная с версии LabVIEW 2.5, Блюм и его сотрудники создали более тысячи промышленных приложений для клиентов во всех северо-восточных штатах США. Для обеспечения неизменного качества работы большого числа разработчиков во многих офисах Блюм установил и в дальнейшем развивал практику разработки приложений на основе LabVIEW, разработанных компанией.

Блюм написал и представил огромное количество презентаций по стилю LabVIEW, включая *Bloomy Controls' Professional LabVIEW Development Guidelines (Рекомендации по профессиональному программированию на LabVIEW от Bloomy controls)* на NIWeek 2002 и *Five Techniques for Better LabVIEW Code* (*Пять секретов хорошего кода LabVIEW*) на NIWeek 2003. Он также опубликовал технические статьи в таких изданиях, *как Test & Measurement World, Evaluation Engineering, Electronic Design* и *Desktop Engineering*.

Блюм получил степень бакалавра как инженер-электроник в Университете Коннектикута. Он также является сертифицированным разработчиком и инструктором LabVIEW (National Instruments Certified LabVIEW Developer and Certified Professional Instructor). У его компании есть офисы в Коннектикуте, Массачусетсе, Нью-Джерси. За более подробной информацией обращайтесь на сайт www.bloomy.com.

Читатели, желающие связаться с Блюмом по вопросам, предложениям и т.п., связанным со стилем программирования, могут сделать это по электронной почте lvstyle@bloomy.com. Читатели, заинтересованные в работе с Bloomy Controls для разработки проектов на LabVIEW, могут обращаться непосредственно в компанию по телефону или через сайт www.bloomy.com/quote.



1 Зачем нужен стиль

LabVIEW – это среда программирования для разработки всевозможных приложений почти во всех отраслях промышленности. Блок-диаграмма представляет собой уникальное средство представления кода, отличное от большинства других языков программирования. Концепция потока данных основывает программу на таких элементах, как проводники, терминалы, структуры и узлы. У каждого элемента есть свое назначение и уникальные возможности. Они полностью раскрываются в среде LabVIEW, которая почти не ограничивает возможности и воображение программиста при написании программ. Это очень плохо! В результате появилось множество стилей создания программ в LabVIEW.

1.1. Зачем нужен стиль?

Одну и ту же задачу можно решить множеством способом, но функционально программы будут эквивалентными. На первый взгляд кажется, что предпочтения, стиль программирования и оформление – личное дело каждого, но это только кажется. На самом деле все это напрямую влияет на качество программы. За всю свою карьеру профессионального консультанта, менеджера и преподавателя LabVIEW я видел множество стилей программирования и неоднократно принимал участие в обсуждении достоинств и недостатков каждого. Могу сказать, что стиль – ключевое понятие для многих разработчиков. Однако необходимо понимать, что хороший стиль складывается не только из личных предпочтений. Следование действительно хорошему стилю позволит вам создать более надежную программу и код, в котором просто разобраться, повысить производительность приложения.

Теорема 1.1 не требует доказательства.

Теорема 1.1 *Стиль программирования в LabVIEW прямо связан с простотой использования, эффективностью, надежностью программы и читабельностью, простотой изменения и структуры кода всего приложения.*

На следствиях из теоремы 1.1 и основана вся книга. Немногие будут возражать, что легче читать простую и понятную диаграмму, чем сложную и запутанную. Аккуратный пользовательский интерфейс удобнее и приятнее использовать. Но знаете ли вы, что виртуальные приборы с аккуратными лицевыми панелями и блок-диаграммами работают эффективнее и содержат меньше неполадок? Принимаете ли вы во внимание, что с вашей программой будут работать люди, не знакомые с вашим стилем? Интересно, что этим человеком может оказаться вы через 6 месяцев или год, или через несколько лет после создания программы, когда вы уже забудете, что вы в ней понаписали. Я встречался с разработчиками, которые не могли объяснить, что делает их код уже через неделю, а тем более через год после его завершения. И наоборот, многие разработчики эффективно работают в команде, пользуются программами коллег в своих программах и не считают, что написать свое – проще, чем воспользоваться уже созданным. Единственное отличие между ними – стиль программирования.

Рассмотрим несколько примеров. На рис. 1.1–1.3 приведен ВП верхнего уровня для трех приложений. В каждом виден свой стиль программирования. Теперь вопросы: Каким приложением вы предпочтете пользоваться или изменять? Какой ВП эффективнее? А в каком наиболее вероятны неполадки и проблемы с памятью?

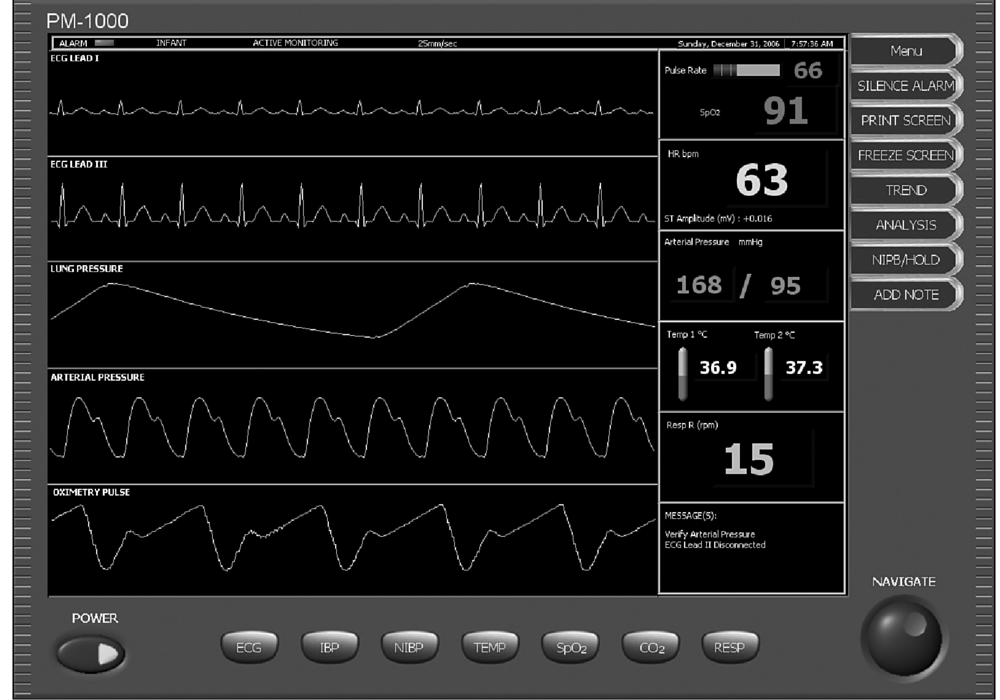


Рис. 1.1. ВП «Дотошный» для сложного приложения. Отличается большим вниманием даже к мельчайшим деталям

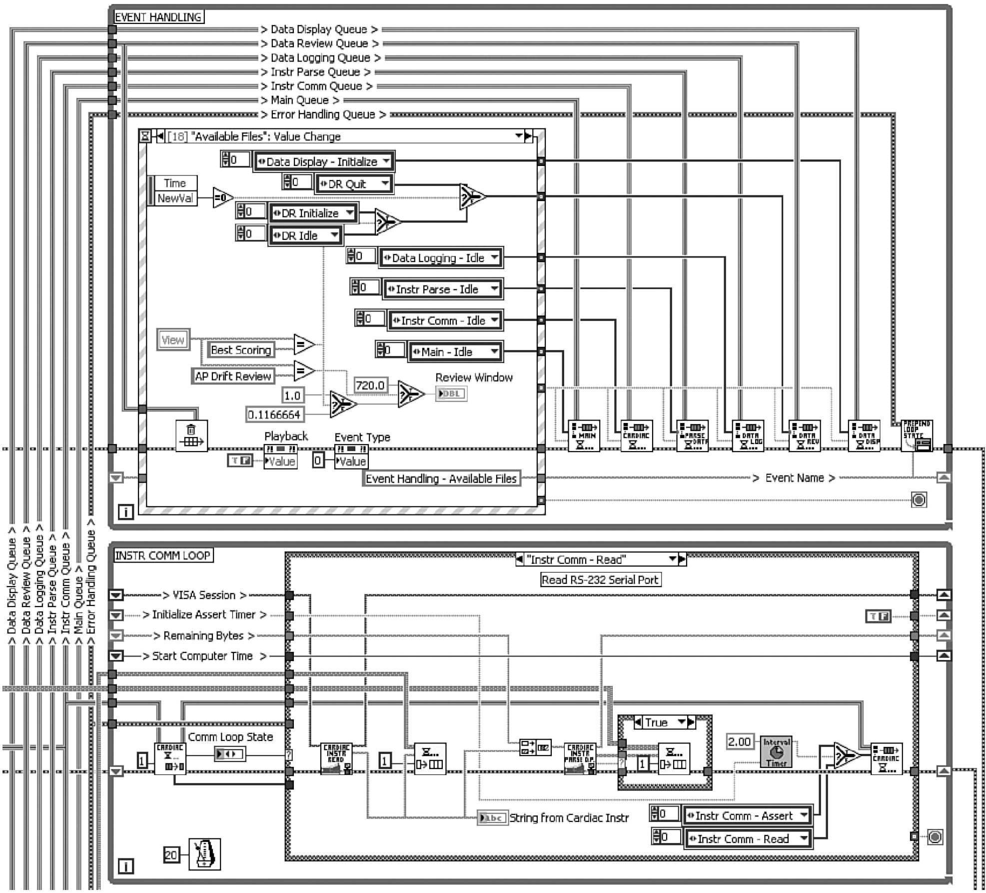
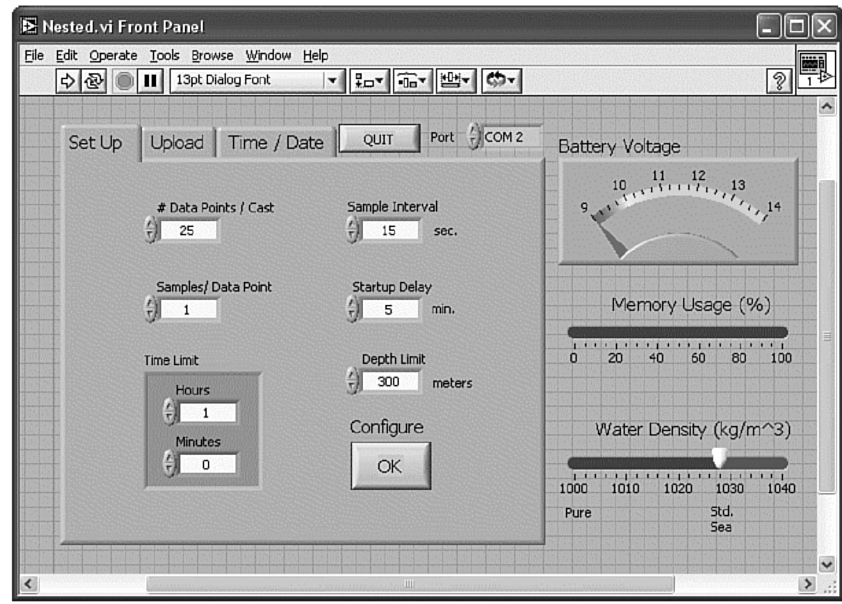


Рис. 1.1. Продолжение

ВП «Дотошный» на первом рисунке – очень сложное медицинское приложение для наблюдения за состоянием пациента. Очень профессиональная и сложноорганизованная структура. Многофункциональная лицевая панель в то же время ясная и понятная. Блок-диаграмма тоже впечатляет: множество параллельных циклов, групп проводников с поясняющими метками, сложных шаблонов и описаний каждого шага.

ВП «Многоуровневый» предназначен для относительно простого приложения управления лабораторным прибором. Ясная, чистая лицевая панель и блок-диаграмма. Однако необходимо отметить возможно излишне многоуровневую структуру.

ВП «Спагетти» – приложение для автоматического тестирования средней сложности. Лицевая панель пестрая, слишком много цветов, которые только путают пользователя. Блок-диаграмма – это просто лабиринт проводов и узлов



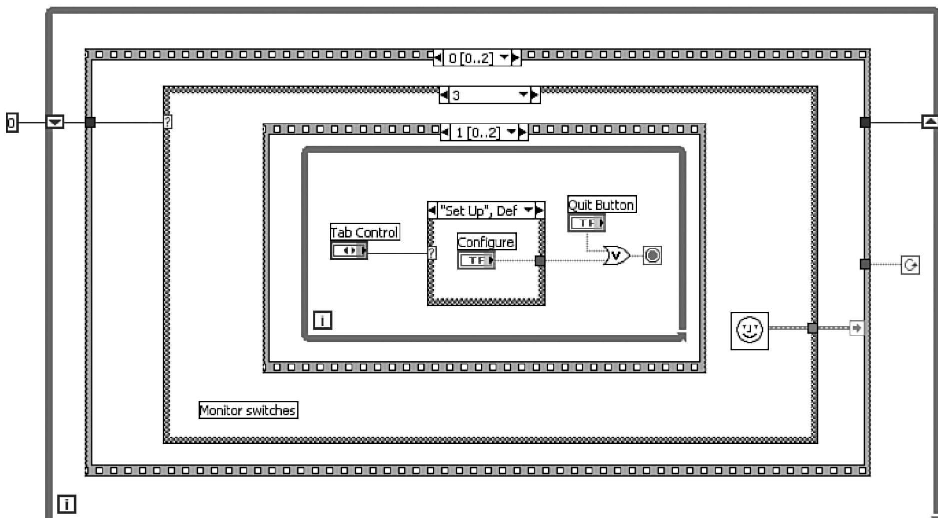
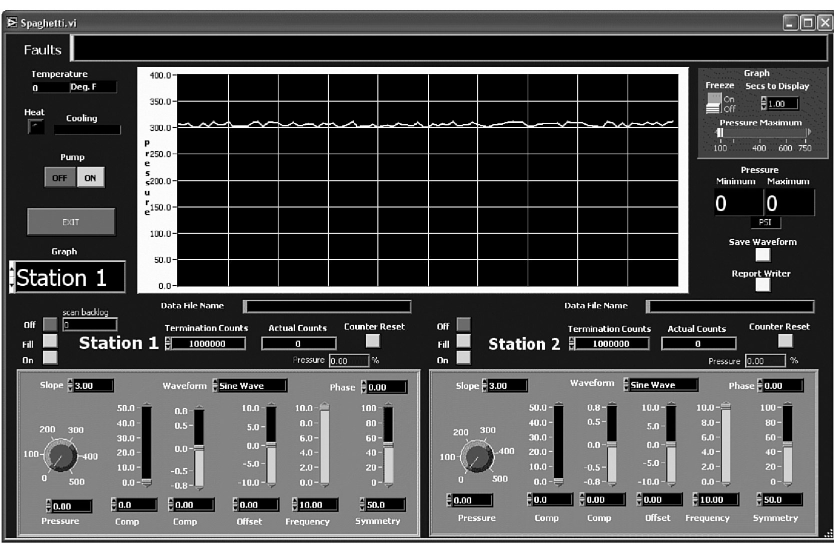


Рис. 1.2. ВП «Многоуровневый» для простого приложения. Очень высокая степень вложенности структур на блок-диаграмме

в одном цикле. Программы отличаются очень сильно, но основное отличие – стиль программирования. Все программы – коммерческие, написаны профессиональными разработчиками.



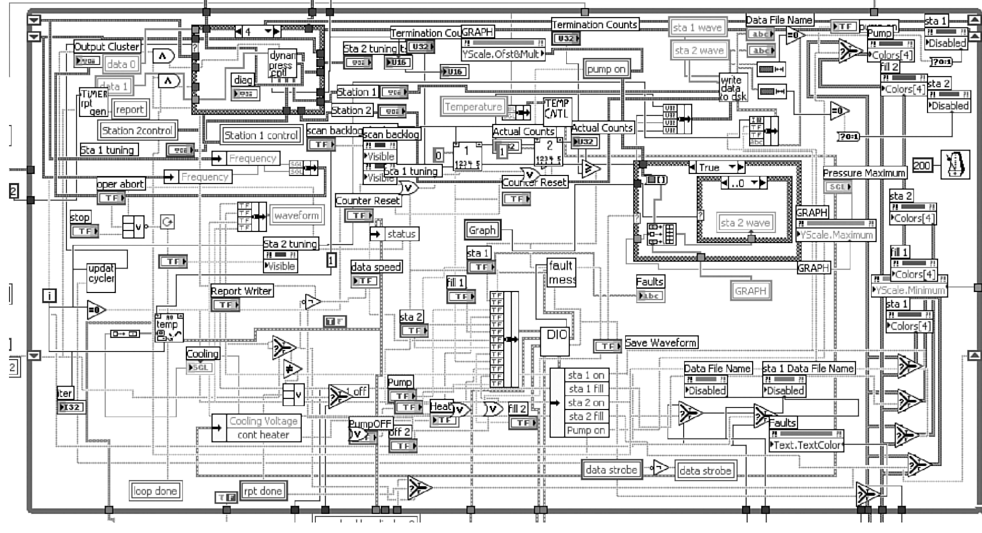


Рис. 1.3. ВП «Спагетти» для не сложного приложения. Комментарии излишни

Опять вернемся к теореме 1.1 и выделим в ней семь составляющих, каждую из которых рассмотрим отдельно. Это – простота использования, эффективность, читабельность, простота поддержки, надежность, простота и качество. Термины слишком общие, иногда перекрываются, могут иметь несколько значений. Поэтому каждый раздел начинается с точного определения термина и только потом анализируется его связь со стилем программирования.

1.1.1. Простота использования

Простота использования определяется количеством затруднений, с которыми встречается пользователь при работе с программой, и относится исключительно к пользовательскому интерфейсу. На простоту использования можно обращать мало внимания, если с программой работает только ее создатель, она может играть заметную роль в промышленных приложениях, когда программой будут пользоваться обученные рабочие одного предприятия. В то же время она может быть одним из основных критериев, например, в приложениях, связанных с безопасностью, или коммерческих для широкого круга пользователей. В последнем случае обязательно прочитайте правила создания интерфейса программ для операционной системы, для которой вы пишете программу. При создании особенно важных приложений обратитесь к литературе, где описывается влияние человеческого фактора, и руководствуйтесь этими рекомендациями. В этой книге содержатся общие правила, которые выполняются практически в любых приложениях: от научных до заводских, на большинстве платформ: от персональных компьютеров с ОС Windows до встраеваемых контроллеров с ОС реального времени.

Простота использования определяется читабельностью и понятностью пользовательского интерфейса. Лицевая панель на рис. 1.1 интуитивно понятна. Основные физиологические данные отображаются на графиках, один под другим, и больших числовых индикаторах в основной части лицевой панели. На дополнительных экранах представлены менее важные данные. Удобное управление реализовано с помощью нескольких кнопок. Все приложение похоже на настоящий прибор, отклик на действия оператора происходит без задержек.

Объекты лицевой панели ВП «Многоуровневый» объединены в логические группы на разных закладках панели. Метки управляющих элементов легко читаются, сами элементы отделены друг от друга. Однако назначение каждой закладки и порядок работы с интерфейсом ясны не сразу. Операции выполняются в структурах последовательности, действия пользователя на них никак не влияют. Время до закрытия программы после команды **Quit** (Выход) не известно, оно зависит от стадии выполнения инструкций.

Объекты лицевой панели ВП «Спагетти» (рис. 1.3) также объединены в логические группы в виде кластеров. Структура логичная и четкая, но количество цветов, шрифтов и отсутствие свободного пространства между элементами убивают все преимущества. Более того, надежность работы оставляет желать лучшего. Как бы ни был привлекателен интерфейс, если он не выполняет своего назначения, положительной оценки он не заслуживает.

В главе 3 «Стиль лицевой панели» перечислены простые правила для создания простого и понятного, но функционального интерфейса. Правила глав 6 «Структуры данных» и 8 «Шаблоны» помогут оптимизировать скорость отклика элементов.

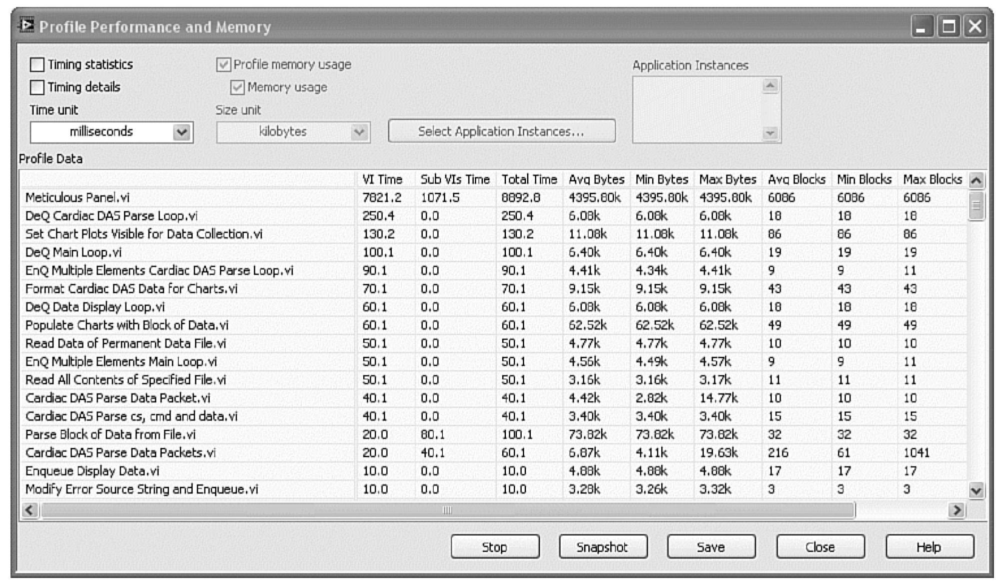
1.1.2. Эффективность

Эффективность определяется количеством использованных ресурсов: времени центрального процессора, оперативной памяти и каналов ввода-вывода. Эффективное приложение исполняется быстро, не выполняет лишних операций, особенно в циклах, и не занимает лишней памяти. Для экономии памяти необходимо следить за четырьмя основными элементами программы LabVIEW: лицевой панелью, блок-диаграммой, данными и кодом. Все изображения элементов лицевой панели и блок-диаграммы хранятся в памяти – это первые два элемента. В следующем элементе памяти хранятся все данные: двигающиеся по проводникам, константы, исходные значения элементов, переменные и значения индикаторов. Код представляет собой компилированную для исполнения программу. Также в эффективном приложении количество операций с элементами ввода-вывода минимально необходимо, но достаточно. К таким операциям относятся: обновления элементов пользовательского интерфейса, связь с приборами и по сети, вызовы сбора данных.

Время выполнения и количество занимаемой памяти связаны между собой. В большинстве современных приложений основные источники задержек – операции передачи данных между оперативной памятью и жестким диском. Если программа требует дополнительной памяти (в любой момент во время работы), менеджер LabVIEW вынужден выделить дополнительный блок. Это, во-первых, занимает определенное время, во-вторых, приводит к фрагментации памяти и низкой эффективности ее использования. Менеджер памяти LabVIEW – очень мощный инструмент, который автоматически выделяет память по мере необходимости и следит за ее использованием. Однако разработчики обязаны знать, в каких случаях он запускается, и свести количество таких операций к минимуму.

Для измерения времени работы и объема памяти загруженных ВП в LabVIEW служит инструмент **Performance and Memory Profiler**. Он запускается из меню **Tools** ⇒ **Profile** ⇒ **Performance and Memory** (Инструменты ⇒ Измерить ⇒ Производительность и память); основное окно приведено на рис. 1.4. Это очень удобное средство оптимизации эффективности приложения. Вы можете определить ВП, который выполняется слишком долго или занимает слишком много памяти, и улучшить его. За несколько итераций эффективность приложения может вырасти очень сильно.

Обратите внимание, что эффективность таких разных приложений, как ВП «Дотошный», «Многоуровневый» и «Спагетти», нельзя сравнивать напрямую. Время работы, занимаемая память и использование ресурсов зависят в первую очередь от самого приложения, но во вторую очередь от стиля программирования. Например, может оказаться, что ВП «Многоуровневый» занимает меньше всего памяти и ресурсов. Но он выполняет самую простую задачу, и именно в нем может оказаться больше всего моментов, требующих оптимизации. Лучше всего **Performance and Memory Profiler** подходит для сравнения эффективности одного приложения во время его оптимизации.



	VI Time	Sub VIs Time	Total Time	Avg Bytes	Min Bytes	Max Bytes	Avg Blocks	Min Blocks	Max Blocks
Meticulous Panel.vi	7821.2	1071.5	8892.8	4395.80k	4395.80k	4395.80k	6086	6086	6086
DeQ Cardiac DAS Parse Loop.vi	250.4	0.0	250.4	6.08k	6.08k	6.08k	18	18	18
Set Chart Plots Visible for Data Collection.vi	130.2	0.0	130.2	11.08k	11.08k	11.08k	86	86	86
DeQ Main Loop.vi	100.1	0.0	100.1	6.40k	6.40k	6.40k	19	19	19
EnQ Multiple Elements Cardiac DAS Parse Loop.vi	90.1	0.0	90.1	4.41k	4.34k	4.41k	9	9	11
Format Cardiac DAS Data for Charts.vi	70.1	0.0	70.1	9.15k	9.15k	9.15k	43	43	43
DeQ Data Display Loop.vi	60.1	0.0	60.1	6.08k	6.08k	6.08k	18	18	18
Populate Charts with Block of Data.vi	60.1	0.0	60.1	62.52k	62.52k	62.52k	49	49	49
Read Data of Permanent Data File.vi	50.1	0.0	50.1	4.77k	4.77k	4.77k	10	10	10
EnQ Multiple Elements Main Loop.vi	50.1	0.0	50.1	4.56k	4.49k	4.57k	9	9	11
Read All Contents of Specified File.vi	50.1	0.0	50.1	3.16k	3.16k	3.17k	11	11	11
Cardiac DAS Parse Data Packet.vi	40.1	0.0	40.1	4.42k	2.82k	14.77k	10	10	10
Cardiac DAS Parse cs, cmd and data.vi	40.1	0.0	40.1	3.40k	3.40k	3.40k	15	15	15
Parse Block of Data from File.vi	20.0	80.1	100.1	73.82k	73.82k	73.82k	32	32	32
Cardiac DAS Parse Data Packets.vi	20.0	40.1	60.1	6.87k	4.11k	19.63k	216	61	1041
Enqueue Display Data.vi	10.0	0.0	10.0	4.88k	4.88k	4.88k	17	17	17
Modify Error Source String and Enqueue.vi	10.0	0.0	10.0	3.28k	3.26k	3.32k	3	3	3

Рис. 1.4. Характеризация памяти ВП «Дотошный»

Еще один способ исследования эффективности приложения – просто убедиться в отсутствии явных недостатков самого приложения. Поищите необязательные операции внутри циклов, для которых требуются новые блоки данных. Например, на блок-диаграмме ВП «Спагетти» (рис. 1.3) есть множество узлов в основном цикле. Это чтение локальных переменных (при этом создается дополнительная копия данных), запись в узлы свойств значений, возможно, совпадающих со старыми.

На блок-диаграмме ВП «Многоуровневый» несколько элементов управления во внутреннем цикле опрашиваются с максимально возможной частотой. Высокоскоростная обработка элементов пользовательского интерфейса – это совсем неэффективное использование процессора. Отклики интерфейса в 100 мс и 1 мс для большинства людей практически не отличимы. Гораздо эффективнее поставить функцию задержки или воспользоваться структурой событий для обработки действий пользователя. Так вы освободите процессор для более важных задач.

В ВП «Дотошный» (рис. 1.1) – несколько параллельных циклов на одной блок-диаграмме, множество сдвиговых регистров и структура событий. Параллельно работают следующие задачи: обработка событий пользовательского интерфейса, передача, протоколирование, обработка, отображение данных и обработка ошибок. Всего 8 параллельных циклов, работа каждого оптимизирована. Однако блок-диаграмма ВП верхнего уровня получилась огромной, с большим количеством данных и компонентов. Занимаемая каждым из четырех компонентов ВП память приведена на рис. 1.5 в окне свойств ВП. Это окно открывается из пункта меню **File ⇒ VI Properties** (Файл ⇒ свойства ВП), категория **Memory usage** (Использование памяти).

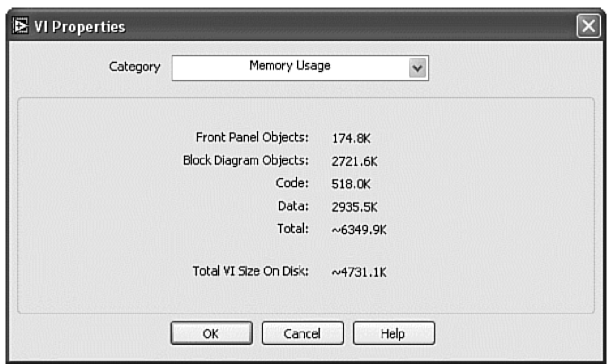


Рис. 1.5. Окно свойств ВП с четырьмя основными категориями памяти ВП «Дотошный»

Наиболее эффективный способ обработки событий пользовательского интерфейса – структура событий (Event structure). Пока не происходит никаких из указанных событий, обработка данной структуры не работает, не занимает процессорного времени. Это и эффективнее и быстрее опроса элементов, чтобы определить, а не изменились ли они. Таким образом, ВП «Дотошный» обеспечивает наилучший отклик на действия пользователя. Также в этом ВП используются сдвиговые регистры для передачи данных между итерациями цикла и очереди для передачи данных между параллельными циклами. Сдвиговые регистры гораздо эффективнее локальных переменных, а очереди – функциональнее.

В ВП «Многоуровневый» эффективно используются ВПП, что эффективно при работе с памятью, но структуры последовательности снижают эффективность обработки ВПП: уменьшают количество узлов блок-диаграммы и экономят память. Также LabVIEW может передавать участки памяти, которые были заняты ВПП, и не выделять дополнительного места. Однако вложенные структуры, особенно структура последовательности, могут вызвать дополнительные задержки. После начала работы структуры все ее кадры должны исполниться один за другим, прервать ее исполнение невозможно. Если отказаться от структуры или перегруппировать ее кадры, приложение может стать более эффективным.

Правила стиля, перечисленные в этой книге, отличаются максимальной эффективностью. Правила наиболее эффективного использования процессорного времени приведены в главах 4 «Блок-диаграмма» и 8 «Шаблоны». Как эффективнее работать с памятью, рассказано в главе 6 «Структуры данных».

1.1.3. Читабельность

Читабельность определяет, насколько просто разработчику понять исходный код программы. Это понятие относится как к лицевой панели, так и к блок-диаграмме. Метки объектов лицевой панели должны быть четкими и понятными. Блок-диаг-

рамма должна быть аккуратной, структурированной и последовательной. Необходимо документировать все приложение.

Лицевая панель ВП «Дотошный» одновременно и простая, и немного таинственная. На рис. 1.6 она показана в режиме редактирования. Элементы управления меню – это кнопки с измененным внешним видом и внедренным текстом. Для быстрой навигации по меню и графикам служит ручка управления. Основное окно отображения – это набор индикаторов на черном фоне, разбитых на две группы: большая панель слева и перекрывающиеся индикаторы справа. Можно загрузить один из двенадцати вариантов отображения, загрузив соответствующий ВП. Перекрывающиеся индикаторы становятся видимыми, их фон прозрачный, а дополнительные элементы – черные, кажется, что цифры сделаны отдельно. Диаграмму ВП легко читать: данные двигаются слева направо, проводники изгибаются не больше одного раза, если нужно обогнуть структуры на пути. Документация к диаграмме также достаточно подробная: это и метки терминалов, и свободные метки на проводниках и в каждой структуре, и итерации циклов с пояснениями.

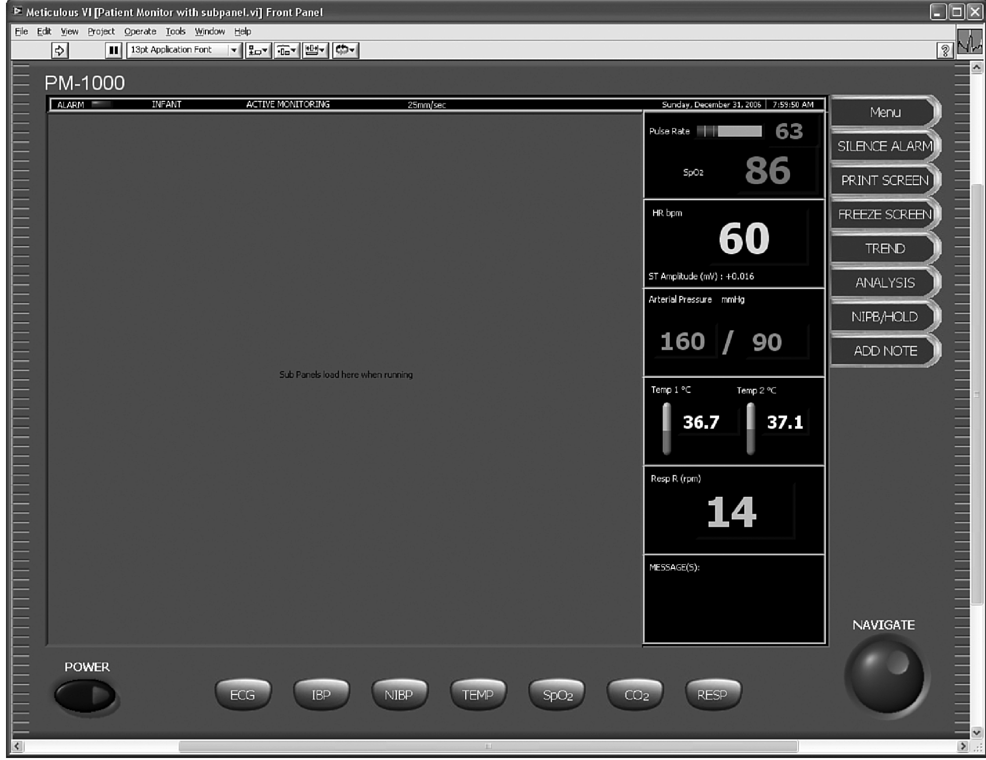


Рис. 1.6. Лицевая панель ВП «Дотошный» в режиме редактирования. На ней находятся измененные кнопки вдоль нижней и правой границ окна, субпанель и несколько перекрывающихся индикаторов

Лицевая панель ВП «Многоуровневый» (см. рис. 1.2) понятная и легко читается. Объекты лицевой панели и блок-диаграммы – с понятными именами и отделены друг от друга. На каждом кадре каждой структуры есть поясняющие свободные метки. Однако ни для ВП, ни для его элементов управления и отображения нет никаких описаний, хотя это важная часть документации любого ВП (подробнее смотрите в главе 9 «Документация». На рис. 1.7 приведено окно контекстной помощи для ВП и элемента управления: никаких описаний.

Рис. 1.7. Окно контекстной помощи: ни для ВП «Многоуровневый», ни для его элемента управления нет описаний

Лицевая панель ВП «Спагетти» (см. рис. 1.3) состоит из простых элементов: индикаторов, элементов управления, кластеров и элементов оформления. Имена большинства из них описательные, но не для всех. Например, в кластере каждой станции у двух элементов имена одинаковые: **Comp** – либо один элемент лишний, либо имена должны быть разными, причем аббревиатуру лучше заменить понятным описательным словом. Правила для текста лицевой панели приведены в главе 3.

Блок-диаграмма ВП выглядит, как спагетти. Причина этого – случайное расположение элементов, данные текут в разных направлениях, архитектура программы не соответствует сложности приложения. Очень сложно проследить хотя бы один проводник от начала до конца. Правила соединения и организации потока данных приведены в главе 4.

1.1.4. Простота поддержки

Программа на LabVIEW обладает этим качеством, если другие разработчики, не ее автор, способны понять код, изменить и дополнить его, чтобы расширить существующие или добавить новые возможности. Таким образом, исходный код должен быть легко читаемым, необходимо пользоваться модульными и расширяемыми шаблонами, которые в будущем позволят увеличить функциональность задачи.

На лицевой панели ВП «Дотошный» расположена субпанель управления и несколько перекрывающихся индикаторов (см. рис. 1.6). Субпанель легко изменить: компоненты загружаются из ВПП и отображаются на лицевой панели, следовательно, простота поддержки очень высокая. Это очень гибкая и модульная

конструкция элементов отображения, редактирование лицевой панели сводится к изменению ВПП. Однако для изменения перекрывающихся индикаторов нужно поработать: скрыть или показать их, аккуратно задать размер и положение. Эта часть лицевой панели отличается более сложной поддержкой.

Блок-диаграмма ВП «Дотошный» состоит из нескольких параллельных циклов, связанных в сложную, но шаблонную структуру. Большинство циклов, за исключением INSTR COMM LOOP, основано на стандартном конечном автомате со структурой выбора по списку, каждый кадр которого соответствует соответствующему состоянию. Конечный автомат легко читается: название каждого кадра описывает его действие. Расширить возможности конечных автоматов также несложно: просто добавить кадры в структуру и элементы в список. Шаблоны «Конечный автомат» и «Параллельные циклы» обсуждаются в главе 8.

На лицевой панели ВП «Многоуровневый» (см. рис. 1.2) расположена панель с закладками, новые страницы добавляются без особых трудностей. Пользователь может в любое время выбрать нужную ему закладку. Блок-диаграмма этого ВП сложнее: вложенные друг в друга структуры последовательности и выбора заставляют разработчика не один раз пройти по всем своим кадрам, чтобы понять принцип работы программы. К тому же этот шаблон не общепринятый. По сравнению со знакомым конечным автоматом его сложнее расширить.

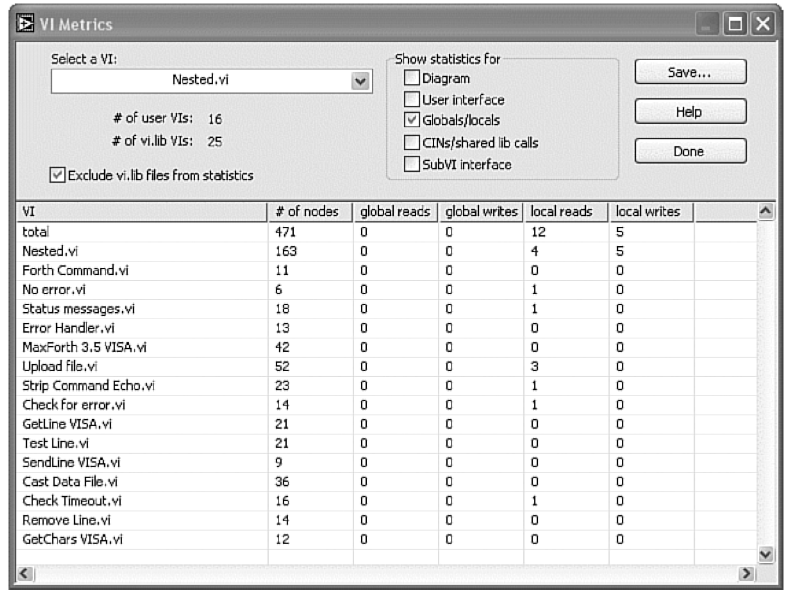
Простота поддержки ВП «Спагетти» отсутствует абсолютно: лицевая панель и диаграмма не модульные, соединения запутаны. Архитектура основана на единственном цикле, все узлы, структуры и проводники внутри цикла перемешаны. Этот шаблон «непрерывного цикла» описан в главе 8. Если нужно добавить новые возможности, разработчик либо расширяет границы цикла (которые и так больше размеров экрана даже при большом разрешении), либо втискивает новые элементы в старые границы. Создать новые элементы и узлы в этом месиве очень сложно. С лицевой панелью также возникнут трудности: чтобы добавить, например, новую станцию, нужно скопировать кластер и несколько дополнительных элементов управления, однако, для этого уже нет места. К счастью, внедрить панель с закладками можно без значительных усилий. Каждую закладку можно назначить одной станции и добавлять новые по мере необходимости. Все элементы управления одной станцией лучше внести в кластер и сохранить его как определение типа (type definition). После этого все изменения типа отражаются на всех станциях, которые им пользуются. Закладки описаны в главе 3, а определения типов – в главе 6. Все правила, приведенные в данной книге, позволяют обеспечить простоту поддержки приложения.

1.1.5. Надежность

Надежное приложение на LabVIEW работает без ошибок, причем так, как задумал разработчик. Фундаментальные конструкции LabVIEW, например подприборы и система обработки ошибок, обеспечивают надежную работу вашего приложения. Другие базовые элементы могут вызвать ошибки, но только при неправильном использовании. Профессиональные разработчики LabVIEW всегда стараются

выделить участки программы, которые выполняют выделенную задачу, и реализовать их в виде ВПП. Это повышает модульность приложения и простоту поддержки. Тестирование и отладка ВПП также не вызывают затруднений: это выделенные функции с небольшим числом узлов. Модульные приложения, состоящие из проверенных и отлаженных ВПП, обычно качественнее созданных с нуля. Модульная структура обеспечивает быструю локализацию и выделение ошибок.

Инструмент **LabVIEW VI Metrics** (Метрика ВП) подсчитывает число использованных ВПП, переменных и узлов диаграммы. Этот инструмент открывается из меню **Tools ⇒ Profile ⇒ VI Metrics** (Инструменты ⇒ Измерить ⇒ Метрика ВП). Информация о ВП «Многоуровневый» приведена на рис. 1.8.



VI	# of nodes	global reads	global writes	local reads	local writes
total	471	0	0	12	5
Nested.vi	163	0	0	4	5
Forth Command.vi	11	0	0	0	0
No error.vi	6	0	0	1	0
Status messages.vi	18	0	0	1	0
Error Handler.vi	13	0	0	0	0
MaxForth 3.5 VISA.vi	42	0	0	0	0
Upload file.vi	52	0	0	3	0
Strip Command Echo.vi	23	0	0	1	0
Check for error.vi	14	0	0	1	0
GetLine VISA.vi	21	0	0	0	0
Test Line.vi	21	0	0	0	0
SendLine VISA.vi	9	0	0	0	0
Cast Data File.vi	36	0	0	0	0
Check Timeout.vi	16	0	0	1	0
Remove Line.vi	14	0	0	0	0
GetChars VISA.vi	12	0	0	0	0

Рис. 1.8. Количество узлов и использованных ВПП в окне VI Metrics *могут многое сказать о модульности приложения*

Можно ввести индекс модульности по следующей формуле:

Уравнение 1.1 $\text{Индекс модульности} = (\text{№ВПП}/\text{№узлов}) \times 100$

По моему мнению, индекс модульности должен быть около 3. В рассматриваемых примерах: ВП «Дотошный» – 111 пользовательских ВПП, 4947 узлов, индекс модульности 2,2; ВП «Многоуровневый» – 16 пользовательских ВПП, 471 узел, индекс модульности 3,3; ВП «Спагетти» – 56 пользовательских ВПП, 1944 узла, индекс модульности 2,9. Более подробно модульность ВП обсуждается в главе 4.

Обработка ошибок заключается в передаче ошибок по кластеру ошибок и сообщении о них в диалоговом окне или в регистрационном (log) файле. Для надежного приложения обработка ошибок просто необходима.

В ВП «Дотошный» ей уделено очень много внимания: все функции, содержащие терминалы ошибок, соединены в цепочку. Любая ошибка передается выделенному циклу обработки ошибок с помощью очереди, анализируется, отображается и регистрируется. Цикл обработки ошибок приведен на рис. 1.9.

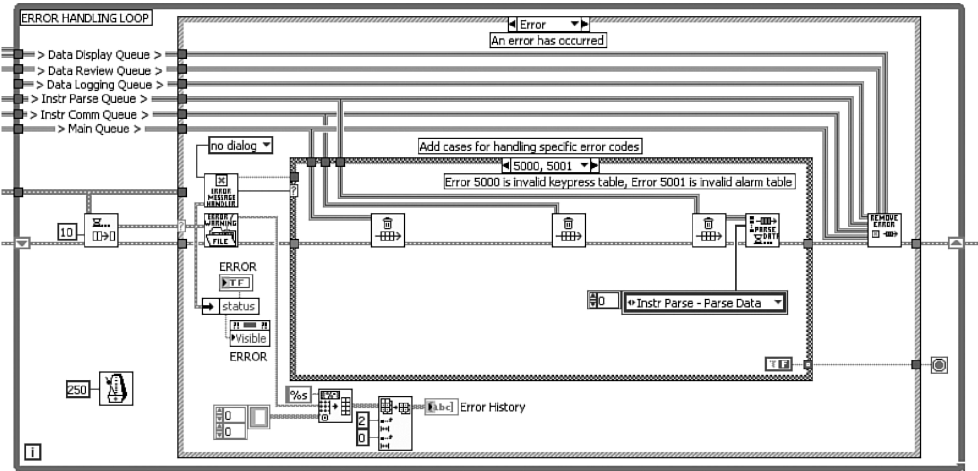
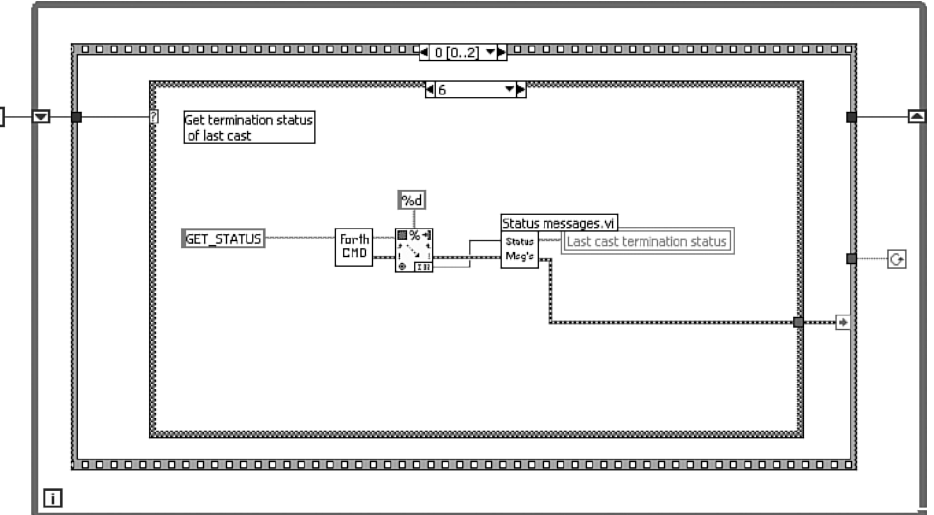


Рис. 1.9. Выделенный цикл обработки ошибок в ВП «Дотошный». Очереди передают информацию об ошибках, потом они обрабатываются, выводятся на экран и регистрируются в файле

Схема обработки ошибок в ВП «Многоуровневый» не такая тщательная и сложная, но в то же время она простая и функциональная. Как видно из рис. 1.10, у первого ВПП, который вызывается в каждом случае внутренней структурой выбора, входной терминал ошибок не используется, а выходной идет на терминал передачи данных структуры последовательности. В ней в первом кадре стоит ВПП обработки ошибок. Схему обработки ошибок можно улучшить, организовав передачу кластера через сдвиговые регистры и задействовав все входные терминалы ошибок. Эффективность немного вырастет, если не вызывать ВП обработки ошибок, если ошибка не произошла.

Пример отсутствия обработки ошибок – ВП «Спагетти». Как видно из рис. 1.3, никакой схемы обработки ошибок нет. В заключение, с точки зрения обработки ошибок, – самый надежный ВП «Дотошный». Корректный анализ ошибок – необходимая составляющая каждого надежного приложения. Очень подробно этот вопрос обсуждается в главе 7 «Обработка ошибок».

Запись в локальные и глобальные переменные – потенциальный источник некорректной работы приложения. Например, если в переменную одновременно за-



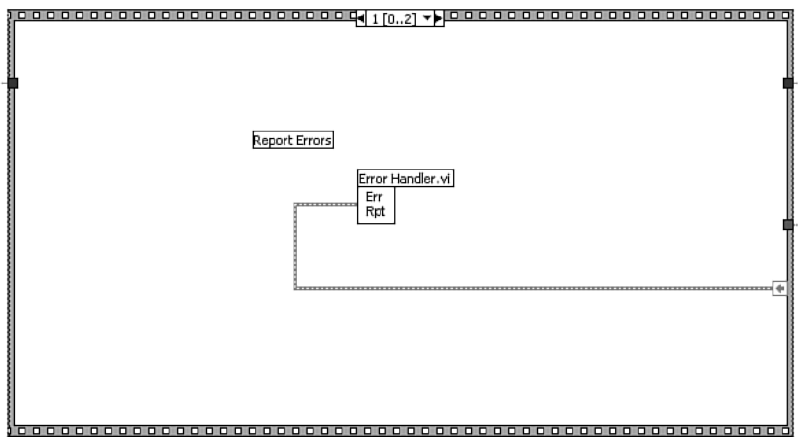


Рис. 1.10. Схема обработки ошибок в ВП «Многоуровневый» работает, но не является оптимальной. Кластер ошибок передается через ВПП, структуру выбора в терминал последовательности. В ее первом кадре он обрабатывается с помощью ВП Error Handler VI

писываются данные из нескольких источников, может произойти так называемая конкуренция (race condition). Общее наблюдение – чем чаще идет запись в переменные, тем больше вероятность нарушения работы программы. Количество операций чтения и записи можно посмотреть в окне **VI Metrics** (Метрики ВП). В ВП «Дотошный» 92 раза идет запись в локальные переменные, в ВП «Многоуровневый» – всего 5 раз, в ВП «Спагетти» – 46 раз. С точки зрения работы с переменными наиболее надежный – ВП «Многоуровневый». Более подробно переменные обсуждаются в главе 4.

Если вы будете следовать советам книги, надежность ваших приложений всегда будет на высоте.

1.1.6. Простота

Простота обратно пропорциональна числу узлов и терминалов приложения. Чем меньше объектов лицевой панели и блок-диаграммы, тем проще приложение. От простоты зависят и читаемость, и производительность приложения. Общее правило советует использовать наиболее простой шаблон из нескольких возможных.

Оценить простоту приложения можно с помощью окна **VI Metrics** (Метрика ВП). Выберите ВП с помощью **Select a VI**, например ВП верхнего уровня. Пункт **totals** (всего) – это общее количество узлов: функций, ВПП, структур, терминалов, констант, переменных и узлов свойств как в ВП, так и во всех его ВПП. Чем меньше это число, тем проще приложение. Обратите внимание, что ВПП, которые вызываются по ссылке, в эту статистику вклада не дают.

В первую очередь простота определяется приложением. Чем больше требуется возможностей, тем больше будет кода, тем сложнее приложение. Во всех рассматриваемых нами примерах ВПП подключаются статически, кроме компонентов лицевой панели ВП «Дотошный». Можно сравнить простоту этих приложений по количеству узлов. Наиболее сложный ВП «Дотошный» – у него 4947 узлов, потом идет ВП «Спагетти» – 1944 узла, самый простой ВП «Многоуровневый» – 471 узел.

Однако стиль также влияет на простоту приложения. Количество элементов для решения одной и той же задачи зависит от выбранного разработчиком подхода. Обычно решения с меньшим числом узлов оказываются более эффективными. Например, OpenG организовала конкурс по программированию в LabVIEW. Необходимо было удалить из строки символ возврата на одну позицию (backspace) и элемент перед ним. ВП оценивались по производительности, стилю диаграммы и стилю иконки. Было подано множество ВП, которые справлялись с задачей. В некоторых ВП использовалось 13 и даже 12 узлов, в других – до 25. Три варианта приведено на рис 1.11. Проще всего первый вариант, в нем меньше всего узлов. Ниже мы сравним их производительность.

1.1.7. Производительность

Это очень широкий термин с множеством определений. Рассмотрим отдельно производительность приложения и ВПП. *Производительность приложения* показывает, насколько хорошо приложение выполняет возложенную на него задачу. Таким образом, мера производительности приложения связана с требованиями. Например, если цель приложения – уменьшить время тестирования, основным характеризующим параметром будет это самое время тестирования. Если цель – повысить качество продукта, то производительность будет определяться количеством бракованных или возвращенных изделий. Обычно приложение решает несколько задач. Чаще всего основная цель – увеличить скорость и простоту работы

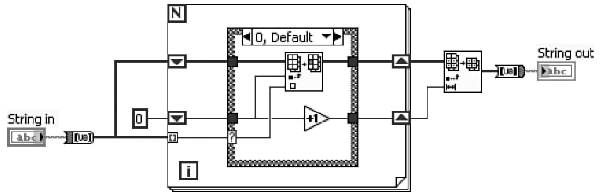


Рис. 1.11а. Решение: 13 узлов

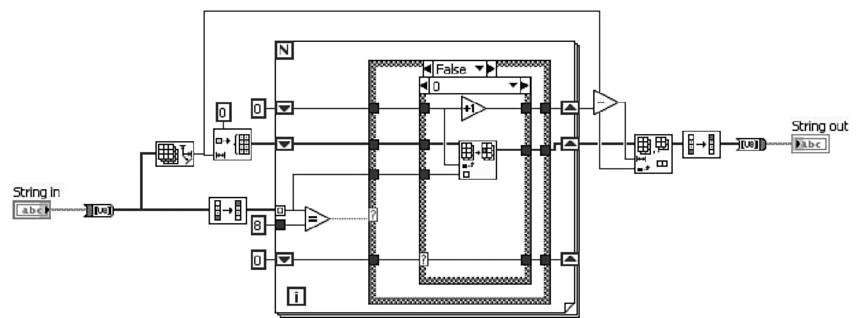


Рис. 1.11б. Решение: 22 узла

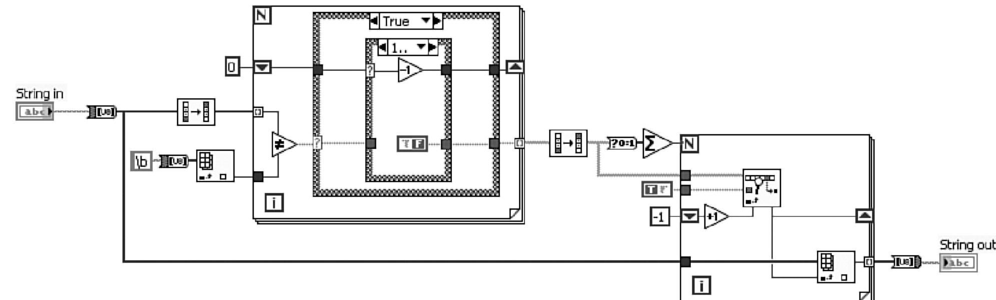


Рис. 1.11в. Решение: 25 узлов

с приложением. Таким образом, производительность приложения – это объединение его эффективности и простоты использования. Надежность также необходима, в противном случае производительность бесполезна.

Посмотрим, как влияет стиль на производительность. В первом примере цель ВП «Дотошный» – надежное измерение, регистрация и отображение физиологических данных. Задача решается на основе шаблона из нескольких параллельных циклов, которые выделяют основные задачи. ВП «Многоуровневый» предназначен для настройки прибора и последующей загрузки данных с удаленных терминалов. В случае отсутствия ошибок простейший интерфейс пользователя успешно решает задачу. В ходе работы ВП «Спагетти» возникают неполадки из-за

недостаточной эффективности архитектуры на основе одного цикла и других факторов, которые мы перечислили выше.

Производительность ВПП – это время его исполнения. В предыдущем разделе мы упомянули о связи производительности с простотой: чем меньше узлов, тем быстрее работает ВПП. Например, ВП анализа строки (рис. 1.11) отличаются по скорости исполнения. На рис. 1.12 приведен пример анализа производительности (инструмент Profile Performance and Memory) при запуске каждого ВПП 1000 раз в цикле для анализа длинной строки. ВП с наименьшим числом узлов (13 – рис. 1.11а) в среднем работал 5,6 мс. Второй пример (22 узла – рис. 1.11б) работал чуть дольше – 5,9 мс. А третий пример (25 узлов – рис. 1.11в) занял 12,8 мс. Разница в несколько миллисекунд может показаться несущественной, но если этот стиль сохранится во всем приложении, производительность может упасть недопустимо низко.

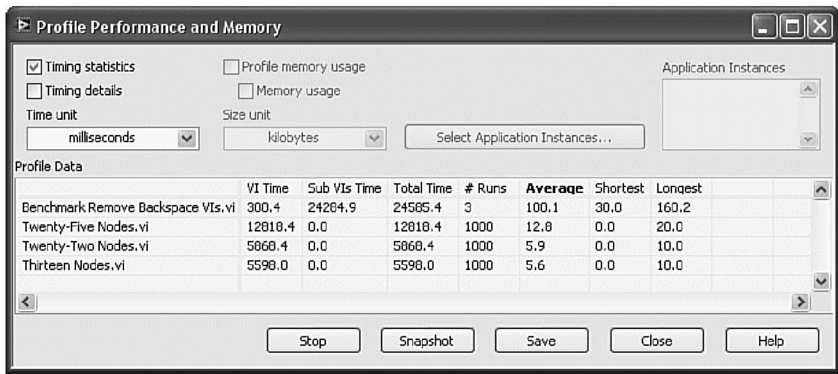


Рис. 1.12. Результаты многократной работы трех ВП – удаление пробелов из строки

1.1.8. Поддержание стиля

Несколько встроенных инструментов LabVIEW помогут вам добиться хорошего стиля программирования. Некоторые из них вы уж знаете – это **Performance and Memory Profiler** для анализа времени исполнения и занимаемой памяти. В закладке свойств ВП **Memory Usage** (Использование памяти) отображается память, которую занимают четыре компонента ВП. Вы можете оценить эффективность и производительность ВП (см. соответствующие разделы главы). Окно **VI Metrics** позволяет увидеть простоту приложения, рассчитать индекс модульности и оценить надежность программы. Есть и другие возможности оценить ваш стиль программирования – это самостоятельный анализ, автоматическое инспектирование и обзор. Сборник правил (приложение 2) поможет вам самостоятельно проверить свой стиль. Многие из приведенных в книге правил учитываются в модуле анализа ВП от NI (LabVIEW VI Analyzer Toolkit). Более подробно они обсуждается в главе 10.

1.2. Стиль или быстрота?

Хорошие разработчики на LabVIEW требуются постоянно. Создаваемые ими приложения определяют успех различных компаний в области исследований, развития и производства. Вместе с тем быстрый ритм требует скорейшей разработки продукта. В результате многие программисты выбирают наиболее быстрый способ решения задачи. В конце концов, это одна из причин, по которой мы выбрали LabVIEW: среда позволяет значительно снизить время создания приложения по сравнению с текстовыми языками программирования.

Определение 1.1 ***Время разработки** – это время, потраченное на разработку, документирование, тестирование, изменение и поддержку приложения **за все время его работы**.*

На первый взгляд кажется, что быстрая разработка приложения и хороший стиль – это взаимоисключающие понятия. Однако это не так: чуть позже вы получите гораздо больше времени, чем сэкономили на стиле сейчас. В компании Bloomy Controls мы несколько раз сталкивались с этим парадоксом. Многие пользователи приходят к нам, потому что у нас – лучшее время разработки. Любые правила программирования, которые увеличивают время разработки, не улучшая эффективности, мы исключаем из рассмотрения. Однако при расчете времени разработки необходимо учитывать все время работы приложения. Многие ошибочно рассчитывают его как время от понимания концепции до выхода первой версии. Они забывают о времени на тестирование, отладку, документирование, проверку и обновления, которые происходят в процессе всего цикла работы приложения. Вот почему мы придерживаемся хорошего стиля.

Теорема 1.2 *Хороший стиль **уменьшает** затраченное время и усилия разработчика.*

Для хорошего стиля может потребоваться больше времени на начальном этапе, но зато с самого начала приложение будет хорошим, более удобным в использовании во время всего цикла работы. Невозможно переоценить это утверждение. Сколько раз Bloomy Controls соглашалась внести минимальные изменения для увеличения возможностей или избавления от ошибок, и каких усилий это требовало. Иногда приходилось разрабатывать приложение с самого начала: из-за плохого стиля мы не могли изменять его. Хороший стиль только снижает время на внесение изменений или модернизацию приложения, а это значительная часть времени разработки.

Необходимо сказать, что в книге приведены только практичные правила, которыми можно пользоваться в реальных ситуациях, в том числе и с жесткими сроками разработки. Другие правила были исключены. Несколько лет назад у нас было правило редактировать историю изменения ВП каждый раз при сохранении любого ВП, вне зависимости от проекта, модели разработки или стадии написания

проекта. В результате все инженеры Bloomy внесли в LabVIEW соответствующую настройку: запрашивать комментирование каждый раз при сохранении прибора. Сразу стало очевидно, что правило непрактично и на первых стадиях проекта требует очень много времени. Примерно через две недели мы от него отказались.

Одно из правил, которым многие пренебрегают, а я считаю необходимым и требую от всего персонала – это описание каждого ВП (VI description). Описание ВП – это одна из необходимых частей документации. Я много раз встречался с коллегами и другими разработчиками, которые откладывали эту небольшую работу на потом, стараясь сэкономить время и уложиться в какой-нибудь срок. Если мы возвращаемся к документированию ВП через какое-то время, то каждый раз приходится вспоминать, что точно делает прибор. Времени тратится гораздо больше. Сразу после создания ВП и его отладки все нюансы свежи, и документирование не отнимает много времени. Описания получаются точными, аккуратными и полезными. Документирование ВП описано в главе 9. После того как вы научитесь пользоваться правилами книги, вы поймете, что это просто практичные советы опытного разработчика.

Подготовка к хорошему стилю

2

Чтобы улучшить стиль приложения, необходима предварительная подготовка. Во-первых, это самое важное – напишите требования к приложению. Создать хороший код, когда известно, что он должен делать, гораздо проще, чем приводить в соответствие код и требования «на лету». Во-вторых, разработайте структуру программы. Воспользуйтесь стандартными шаблонами, например конечным автоматом. После этого настройте среду LabVIEW. Воспользуйтесь преимуществами подходящих настроек и разработанного ранее кода, например шаблонов, драйверов приборов и функций. И наконец, разработайте правила размещения, именования и управления файлами проекта. Только в строго структурированном проекте может быть хороший код.

В этой части описывается подготовка приложения с хорошим стилем. Необходимо отметить, что некоторые из приведенных в главе правил связаны не только с управлением конфигурированием (configuration management – CM), но и напрямую со стилем. Разумеется, многие правила как главы, так и всей книги помогут вам на всех стадиях разработки приложения, в том числе и при управлении конфигурированием, одной из необходимых частей которого является хороший стиль. Приложения на LabVIEW и области их применения слишком разнообразны, чтобы можно было сделать единую систему управления конфигурированием. На основании предлагаемых правил вы сможете разработать систему, которая лучше всего подходит к вашей области.

2.1. Техническое задание

Многие приложения LabVIEW начинаются с необходимости достичь какой-нибудь цели, например, в области промышленного управления или автоматизации. Такой целью может стать увеличение производительности, качества, скорости исследований или разработки, уменьшение ручного труда. Для многих LabVIEW – один из наиболее быстрых методов разработать приложение, которое выполнит поставленную задачу. Кроме этого у LabVIEW есть множество других достоинств: графический язык программирования, открытая поддержка тысяч приборов, переносимость платформы и сетевые возможности. Но наиболее важным для большинства компаний критериев остается экономия времени и средств. И мно-

жество пользователей выбирают LabVIEW как систему, которая отличается минимальным временем разработки готового продукта.

Множество различных особенностей LabVIEW позволяют увеличить скорость разработки приложений. К средствам верхнего уровня относятся Express ВП, драйверы приборов, шаблоны и примеры, которые позволяют создать небольшое приложение с наименьшими затратами на планирование и подготовку. Скорость, с которой вы можете подключить прибор и начать работу с ним, уникальна. Возможностями быстрой разработки кода постоянно пользуются разработчики и менеджеры. В результате многие склоняются к скорейшему завершению кода. Все шаги, которые кажутся необязательными, зачастую отбрасываются.

Разработка требований к приложению – это абсолютно необходимый шаг. Отказ от него будет стоить вам огромных затрат времени и средств. К тому же может произойти такое явление, как сдвиг требований, когда требования к приложению могут измениться во время разработки. Если же все оформлено и документировано с самого начала, вы точно знаете, когда работа будет закончена.

По моим оценкам, менее четверти всех разработчиков записывают формальные требования к программе. Многие проекты начинаются на конференциях или при других личных встречах, наибольшее внимание уделяется оборудованию и общим требованиям к системе. Причем оборудование обычно преобладает, ведь приборы, их характеристики, цены и время поставки – это точно определенные значения. Всем известная гибкость и возможности LabVIEW позволяют считать характеристики, цены и время разработки программ почти виртуальными. На самом деле, у любой программы, как и у прибора, есть свое время создания в человеко-часах. Этот факт часто упускается из вида, что и приводит к непонятым желаниям заказчика и сдвигу требований.

Нельзя сказать, что сдвиг требований – однозначно вредное явление. В области научных исследований, например, требования к системе измерения и автоматизации могут естественным образом измениться вместе с целями проекта или характеристиками всего процесса исследований. В таких областях, с быстро меняющимися условиями, наиболее четко проявляется гибкость и скорость разработки, присущие LabVIEW. Также, если приложение выполнено до окончания остальных работ по проекту, его возможности можно расширить и вывести его на новый уровень. Если же нет точно описанных требований, вы не можете точно сказать, на какой стадии находится проект, успеваете вы его закончить или нет, да и завершить такой проект просто невозможно.

Теорема 2.1 *Сформулированные требования – признак хорошего стиля.*

Документированные требования помогут вам добиться хорошего стиля всего приложения. При записи требований план проекта получается гораздо детальнее и точнее, чем при устном обсуждении. Любой ознакомившийся с этим документом будет четко представлять требования и задачи всего проекта еще до его фактического начала. Стороны могут уточнять и видоизменять требования, пока не придут к полному согласию. После начала работ по проекту вероятность непонимания, избыточной реализации и непредусмотренных деталей падает многократно.

Анализ или дополнение требований – отличное упражнение для разработчика, приступающего к проекту. Во время всех встреч я стараюсь как можно тщательнее понять проект со всех сторон. Каким бы уверенным я ни чувствовал себя в конце совещания, я никогда не завершаю техническое задание без ответа на новые вопросы и идеи, возникающие в процессе. Полученные ответы помогают не только мне лучше понять весь проект, но и заказчику получить более качественное приложение.

Теорема 2.2 *Непредусмотренные изменения проекта отрицательно влияют на его стиль.*

Случайные изменения требований к проекту могут не только снизить уровень его стиля, но и превратить его в спагетти. Разработчик обычно выбирает для проекта самый простой шаблон, который способен реализовать поставленные требования. Например, для ВП верхнего уровня в простом приложении вполне подойдет шаблон непрерывного цикла. Но эта архитектура практически не позволяет вносить дополнительные возможности. Чтобы их добавить, придется расширить границы цикла для новых узлов. При увеличении числа элементов на одной диаграмме сложнее связать их в аккуратную сеть, обойти проводниками уже существующие узлы, другие проводники и структуры. Именно так и мог появиться ВП «Спагетти» из главы 1. Взгляните еще один раз на рис. 1.3: в одном цикле множество функций, проводников и структур выбора. Возможно, сначала приложение было простым и аккуратным, а при расширении требований к проекту превратилось в хаос. Для начальной простой задачи выбранная архитектура была достаточной, но не для последующих дополнений. Если эти предположения верны, разработчику, разумеется, нужно было не только внести изменения, но и изменить архитектуру. Также мне кажется, что сначала не было четкого понимания границ проекта и своего стиля. В любом случае документирование требований значительно снижает вероятность их сдвига, изменений проекта, которые отрицательно сказываются на стиле программы.

Если требований к проекту нет, то либо разработчик их не понимает, либо помнит все наизусть. Хотя одна ситуация от другой почти не отличается. В первом случае со временем программист начнет понимать, что требуется от программы, и каждый раз будет вносить новые изменения в программу. Эта модель называется циклом программирования-отладки. В соответствии с теоремой 2.2, чем больше изменений вносится в программу, в том числе и связанных с изменением понимания целей, тем запутаннее она становится. Это утверждение справедливо для всех языков программирования.

Во второй ситуации, когда разработчик хранит все в своей памяти, понимает требования, но не записывает их, могут возникнуть свои проблемы. Без документации тщательность планирования ПО снижается многократно, в результате чего

первая версия программы будет ужасной и потребует множество изменений. Опять получается цикл программирования-отладки. К тому же вся документация в памяти разработчика недоступна его коллегам. Постепенно все забывается, детали исчезают из памяти, и возможность поддержки приложения со временем падает. Также один разработчик может оказаться недоступным во время всего цикла службы приложения. Ответственность, работа, должность подавляющего большинства работников меняются, разработчики LabVIEW – не исключение.

Как показывает опыт работы Bloomy Controls, многим компаниям требуется изменить, дополнить или отладить различные приложения, написанные на LabVIEW. Без документации и описаний в исходном коде разобраться очень сложно. Такие приложения часто переделываются с самого начала, с документации. И наоборот, если есть подробная документация, с кодом также все в порядке. Так происходит, конечно, не всегда, но очень часто.

2.1.1. Советы по документации

Хорошая документация – это следующий шаг после записей в блокноте. На многих встречах во время обсуждения проекта происходит «мозговой штурм», и необходимо быстро и аккуратно записать основные положения. Поэтому записные книжки входят в обязательный набор любого уважающего себя инженера или ученого. Также они идеально подходят для предварительного оформления требований к проекту.

Правило 2.1 Ведите журнал проекта в LabVIEW

Записная книжка – это предшественник журнала проекта (LabVIEW project journal). Записывайте дату внесенных изменений и ведите этот журнал в хронологическом порядке. Не стоит ограничивать заметки только техническими требованиями, по мере развития проекта они расширяются до полноценных руководств, возможно, с описанием неполадок, перечнем пожеланий, вычислениями по проекту, советами службе поддержки и т.п. Журнал проекта в LabVIEW выполняет те же функции, что и лабораторный журнал для ученого или изобретателя. В лабораторных журналах хранятся данные эксперимента или наблюдения, всегда являющиеся частью научной деятельности. Чуть позже на них основывается анализ, обсуждения, дальнейшая работа и интерпретация данных. Точно так же и журнал проекта – это смесь данных и наблюдений, из которых формируется законченный проект. Очень часто ученые и изобретатели занимаются разработками на LabVIEW и пользуются LabVIEW для проведения экспериментов. И наоборот, разработчики LabVIEW пользуются заимствованными у ученых методами, к таким заимствованиям относится и ведение журнала проекта. В некоторых случаях записанные данные можно даже использовать для подтверждения изобретения, когда компания подает заявку на патент.

Хорошей практикой является запись исходных требований в журнале проекта. В соответствии с теоремой 2.1, сформулированные технические требования положительно сказываются на стиле программирования в LabVIEW. Разработчик может обращаться к своим записям при создании или поддержке приложения до тех пор, пока не потеряет зрение или блокнот. Более того, журналы проекта – это основа формальных требований к разработчику.

Основные достоинства журнала проекта – это простота и краткость, но есть некоторые ограничения:

1. Блокнот не позволяет структурировать данные. Все записывается в хронологическом порядке, и текущие требования, идеи и заметки смешиваются в одну кучу с отложенными. Отличить одни от других бывает затруднительно.
2. Почерк у многих оставляет желать лучшего. Да и во время мозгового штурма трудно вести аккуратные записи, а именно во время них и формируются требования ко многим проектам.
3. Данные из блокнота сложно передать кому-нибудь еще. Попытка пользоваться рукописными данными одного человека в качестве руководства по организации проекта в нескольких командах разработчиков ни к чему хорошему не приведет. Блокнот – это, скорее, личная копия журнала работы по проекту.
4. Если оставить записи в сыром виде, не формализовать и не расширить их, то они не дадут автору возможность взглянуть на проект с другой стороны. Более того, если не обсуждать и не обмениваться требованиями между несколькими командами разработчиков, невозможно прийти к требованиям, которые удовлетворят всех.

Правило 2.2 Напишите техническое задание

Техническое задание, основанное на записях журнала, должно быть для каждого приложения. В грамотном техническом задании содержатся основные цели проекта и список желательных функций в порядке важности. В документе достаточно подробно должны быть описаны все важные требования. Однако конструктивных особенностей приложения нужно избегать, если они не влияют выполнение требований. Нельзя путать техническое задание и описание структуры проекта.

Рекомендации к форме технического задания сформулированы в Руководстве к ТЗ стандарта IEEE 830-1998, а советы по его разработке – в стандарте IEEE 1233-1998. Это отличные руководства, которые помогут вам написать подробное техническое задание к ПО на любом языке программирования, в любой области промышленности и типе приложения. Созданные по этим рекомендациям технические задания – это подробные высококачественные документы.

В некоторых быстроразвивающихся отраслях быстрое изменение требований или необходимость быстро вывести на рынок ограничивают создание хорошей документации. Очень часто желательно разрабатывать программное обеспечение параллельно с оборудованием. Многие характеристики приложения, которые влияют на особенности программы, неизвестны или часто изменяются. Такая си-

туация – не повод, чтобы отказаться от создания технического задания. Наоборот, строго вместе с развитием ПО и оборудования должно идти развитие документации. Например, методы выполнения проектов, принятые в Agile, такие как экстремальное программирование (Extreme Programming), отлично подходят для эволюционирующих проектов. В Bloomy Controls рабочие часы на создание документации и поддержку выделяются отдельно от разработки ПО, такой подход позволяет постоянно обновлять техническое задание. Техническое задание не должно ограничивать изменяющиеся требования. Вместо этого постарайтесь определить наиболее вероятный и самый неблагоприятный сценарий развития и включите его в ТЗ. Если это невозможно, просто поставьте «подлежит определению» и вернитесь к этому пункту позже. Однако в большинстве государственных проектов такая формулировка запрещена. За редкими исключениями, требования должны быть сформулированы до начала разработки ПО. Но, к счастью, федеральные проекты редко относятся к быстро эволюционирующим.

Подведем итоги. Журнал проекта LabVIEW – это неформальная история изменений технической документации проекта, в которой полностью описана цель проекта. Эти документы необходимы для хорошего стиля программирования.

2.1.2. Проектная документация в LabVIEW

Проекты LabVIEW относятся к области действия стандартов IEEE. А именно, приложения на LabVIEW относятся к задачам измерений и автоматизации, в которых обычно ожидается короткий цикл разработки. Большинство задач состоят из сбора, анализа и представления данных. Эти особенности позволяют сформулировать требования к ТЗ, специфичные для проектов LabVIEW. По моему мнению, обязательны следующие пункты:

- формулировка основной задачи;
- бюджет;
- временные рамки;
- подробные описания параметров сбора, анализа и представления данных;
- приоритет каждого требования;
- методика тестирования.

Шаблон ТЗ приведен на рис. 2.1.

При формулировке основной задачи проекта должна быть обоснована необходимость проекта. Очень полезно разъяснить связь проекта с основными продуктами и услугами компании. Этот пункт также называют коммерческой целью проекта. Например, можно сформулировать его в следующем виде:

Разработать систему выхлопа с малым выходом загрязняющих веществ для проектируемых моделей автомобилей 2010 года. Первые 100 прототипов должны быть протестированы в январе 2008 года. Массовый выпуск должен быть налажен к июлю. Потенциальный рынок продукта равен 1 млрд долл. в течение 5 лет.

Бюджет и срок выполнения не только влияют на параметры системы, но и могут определить осуществимость задачи в целом. Абсолютно необходимо представлять расходный бюджет и предполагаемую дату завершения до начала проек-

Automated Test, Data Acquisition, and Control Specialists

LabVIEW Project Specification

[System Name]
[Company]
Revision 1.0
[Date]

- 1 -

Introduction

The following pages contain a functional requirements specification for [system name] for use by [Company] in [City, ST]. The specification was written and reviewed by the following people:
[List the names, titles, and companies of all contributing authors. Describe any other documents referenced by or related to this specification.]

Objective

[Describe Company: What do they provide? To whom? For what? Describe the customer's business objective that's driving this project. Describe current test/measurement/automation/control challenge. Describe the approximate budget and timetable for addressing the challenge.]

System Overview

[Describe the overall system in very high level terms, including both customer-supplied and Bloomy-supplied hardware and software. Describe any subsystems that comprise the system, as well as any systems that are associated but external to the system. Include an overall system block diagram. Clearly differentiate between subsystems and components that are part of the system specified in this document, versus external systems and components. Describe high-level functionality of the specified system. Describe how the system addresses the Company's challenge and business objectives, including budget and timetable.]

Hardware

[Describe the system hardware platform including PC, interface bus, and the primary instruments, modules, or DAQ devices. Describe the purpose of each significant component. Describe any fixtures or equipment racks that are required.]

Input/Output List

[Insert a table containing an itemized list of physical parameters to measure and control, transducers and control devices, DAQ devices and instruments, and PC interface. If the table is lengthy, i.e. more than 25 channels, move it to an appendix.]

Software

[Describe the software platform including PC operating system, LabVIEW, add-on toolkits, TestStand, etc., 3rd party application(s), as well as the custom application(s) to be developed. Describe the purpose of each significant application or tool.]

- 2 -

Acquisition

[Describe the software routines that will acquire data from and/or control the hardware devices. Specify the desired sampling and/or update rates, synchronization, data format, etc.]

Analysis

[Describe any on-line and/or post-acquisition data processing routines that are required. Specify the equations and algorithms to be used. Describe the required throughput for on-line analysis.]

Presentation

User Interface

[Describe the graphical user interface design. Describe the level of operator training and any ease-of-use features. Include a prototype screen shot of one of the primary display screens. Always include the customer's company logo.]

Data Files

[Describe any data files that will be created. Specify the data format, such as binary, ASCII, XML, Microsoft database (mdb), etc. Specify the location, such as local hard drive or remote network server, and how the destination is specified. Specify the required data logging rates and any event or criteria that triggers the data logging. Describe the applications that will be able to read the data, such as MS Word, Excel, Access, SQL Server, Oracle, etc.]

Reports

[Describe any reports that are generated by the system. Specify the format and destination, such as printer, HTML, RTF, or plain text file. List all of the required headings and data fields and the size and format of each. Include a sample report in an appendix if/when possible.]

Connectivity

[Describe any network, intranet, or Internet connectivity required, such as remote access and/or control via web browser, data distribution via FTP, e-mail, etc., or client/server communication via ActiveX, TCP, UDP, DataSocket, or Logos].

- 3 -

Priority Matrix

[Create a table containing an itemized list of software features, and priority level for each. Priorities should include Critical, High, Medium, and Low. This is essentially a subset of the Project Planning Worksheet, without the hours, rates, etc.]

Test Methodology

[Describe how the system will be tested. Will any in-house testing be performed prior to integration at the customer site? Describe any software and/or hardware that will be utilized or developed to simulate and/or test each feature. Specify any use cases that will be applied to test the integrated system. Describe the customer's responsibility for testing the system, if applicable.]

Appendix A: Glossary

[Define all terms, acronyms, and abbreviations used within the specification. List in alphabetical order.]

Appendix B: Input/Output Channel List

[For high channel count DAQ systems (i.e. > 25 channels), place the I/O list in this appendix instead of the hardware section of the main specification body.]

Appendix C: Sample Report

[Create a prototype of any report(s) that must be generated by the system.]

Appendix D: Product Specifications

[Include the manufacturer's specifications of any 3rd party hardware and software products that are discussed within the specification.]

- 4 -

Рис. 2.1. Шаблон технического задания для проекта LabVIEW

та. Могут возникнуть различные препятствия, например, вы можете обнаружить, что период освоения или стоимость комплектующих выходят за рамки требований. Чем раньше вы это выясните, тем лучше.

Я предпочитаю разделять требования на четыре категории: *сбор*, *анализ*, *представление* и *методика тестирования*.

В разделе *«Сбор»* определяются измерения и интерфейсы оборудования. Удобно в одной таблице перечислить все измеряемые параметры и точки ввода/вывода, указав размерность величин, диапазоны, точность и частоту измерения или генерации данных. На этой информации будет основан выбор передатчиков, системы обработки сигнала, ЦАПы, приборы и управляющие блоки. По мере определения эти блоки добавляются в ту же таблицу и постепенно формируют аппаратную часть системы. Обратите внимание, что интерфейсы оборудования и параметры каналов ввода/вывода определяют необходимые библиотеки и элементы LabVIEW: можно воспользоваться доступными драйверами, придется заняться низкоуровневым программированием с помощью VISA или DAQmx или хватит интерактивных экспресс-ВП?

В разделе *«Анализ»* описывается математический аппарат, необходимый для обработки данных «на лету» или после измерений на основе данных, сохраненных в файлы. Иногда требуется предварительная обработка или прореживание данных, фильтрация, поиск экстремумов, анализ ограничений, статистическая обработка и алгоритмы управления. Например, в автомобильной промышленности часто применяется статистическое управление технологическим процессом, для которого требуется расчет средних значений, диапазонов изменения, стандартных отклонений, возможностей технологического процесса, а потом графическое отображение результатов в разных форматах. Все эти вычисления должны быть описаны в разделе *«Анализ»*.

В разделе *«Представление»* описываются все способы отображения данных, с которыми имеет дело пользователь. Сюда входят графический интерфейс, текстовые файлы данных и отчеты. Очень полезно и просто сделать в LabVIEW прототип интерфейса и добавить его внешний вид в техническое задание. Не тратьте много времени на создание работающего интерфейса или подбор цветов и шрифта текста на этой стадии. Достаточно, чтобы было ясно расположение элементов и их масштаб. При такой работе могут выявиться требования, которые вы упустили из виду или недопоняли. Как говорится, лучше один раз увидеть, чем сто раз услышать. Было бы полезно сделать демонстрационный отчет и файл данных в Microsoft Word или Excel.

Обратите внимание, что рисование лицевой панели программы и печать демонстрационных отчетов требуют определенных усилий и, строго говоря, отличаются от чистой формулировки технического задания. В идеальном мире разработка требований и программирование отделены друг от друга. На практике иногда получается так, что программисты могут дополнить интерфейс какими-либо элементами. Лучше рассматривать техническое задание как живой документ, который обновляется на разных стадиях развития проекта. Поэтому детализация технического задания с течением времени изменяется. В результате на основе ТЗ формируется руководство пользователя. Обязательно сохраняйте разные версии проекта, в противном случае вы можете потерять цель развития и скатиться к модели разработки «цикл отладки-программирования». Средства управления исходными файлами проекта обеспечивают сохранение и документацию развивающегося проекта.

В разделе *«Методика тестирования»* описываются правила проверки программы и оборудования. В идеале у каждой функциональной особенности есть свой тест, который позволяет ее локализовать и проверить правильность работы. Иногда формулируется список режимов использования, в которых описываются условия работы программы и оборудования. В задачах измерения и тестирования часто применяют прибор с известными характеристиками, проводят измерения и сравнивают их результаты. Но во многих приложениях такого прибора с известными характеристиками не существует. В этих случаях приходится в методику тестирования включить разработку ПО, которое его эмулирует. В любом случае все эти стадии и проводимые измерения необходимо описать и корректировать по мере развития проекта.

В Bloomy Controls выделяются несколько уровней требований: *критические*, *жесткие*, *рекомендуемые* и *желательные*. Критические требования напрямую следуют из цели проекта. Это вехи. Если какое-либо требование реализовать невозможно, выполнение работ приостанавливается. Жесткие требования очень важны для реализации проекта, но не ведут к его прекращению в случае невыполнения. Рекомендуемые требования – наиболее широкий тип особенностей. Желательные требования выполняются, если позволяют средства, время, а все более высокие требования уже реализованы. Приоритеты расставляются на стадиях разработки и реализации, чтобы определить порядок, сроки и расходы на реализацию требований.

Если возможно, с техническим заданием должны познакомиться все занятые в проекте: как разработчики, так и пользователи. Обязательно проводятся совещания для обсуждения, пересмотра или исправления заявленных требований. Текстовый редактор с возможностью комментирования и регистрации изменений – простой, но полезный инструмент для многопользовательского редактирования. Более специализированные средства редактирования требований, например DOORs от Telelogic и Rational RequisitePro от IBM, позволяют отслеживать изменения более наглядно. Также дополнительный модуль NI Requirements Gateway позволяет связать техническое задание, оформленное в одном из упомянутых пакетов, с исходным кодом LabVIEW. Эти средства позволяют анализировать приложения и проверять его соответствие заявленным требованиям.

2.2. Проектирование

Программное обеспечение проектируется, как обычно, с помощью диаграмм и псевдокода или на универсальном языке моделирования с классами, последовательностями, состояниями и диаграммами компонентов. Они описаны в самых разных источниках, и я здесь не буду на них останавливаться. Просто расширим действие теоремы 2.1 на проектную документацию. Чем более подробным будет план на этой стадии, тем лучше станет стиль вашего программирования.

Многие разработчики после формулировки технического задания переходят сразу к разработке приложения. Если требования оформляют меньше 25% программистов, то проектной документацией не занимается почти никто. Я часто был свидетелем несогласованности разработки оборудования и ПО. Стадия разработки оборудования присутствует всегда: все спаи производятся по начерченной схеме. Блок-схемы и чертежи предшествуют изготовлению деталей, крепежей, стоек. Проектная документация обязательна для оборудования. Чем же программное обеспечение хуже?

Объяснения, нисколько не извиняющие, иногда приводятся. В некоторых приложениях структура определяется после попытки реализации. Для исследования характеристик программы создаются рабочие прототипы ВП. Например, они позволяют определить скорость работы сбора данных и алгоритма на данном компьютере. Принятые прототипы в конце концов превращаются в часть программного обеспечения. Многие разработчики предпочитают развивать прототипы до законченного приложения по модели «цикл разработки-тестирования». К чему это может привести, вы уже видели на примере ВП «Спагетти» в главе 1. Дисциплина требует отложить прототип в сторону и вернуться к разработке проектной документации перед тем, как продолжить создание ПО.

Еще одно объяснение, что простые приложения проектировать не обязательно. Да, приложения, созданные в LabVIEW, могут быть разными: от очень простых до самых сложных. Если задачу можно решить с помощью пары экспресс-ВП, проектирование можно пропустить. Также, если вы можете быстро собрать приложения из ваших любимых шаблонов, драйверов и функций – того, с чем вы очень хорошо знакомы, и программа предназначена для личного пользования, можете не заморачиваться с проектной документацией. Какова же граница сложности приложения, после которой требуется проектная документация? Она даст вам возможность взглянуть на проект изнутри, если вы ее написали.

И наконец, иногда полученную схему сложно воплотить в виде диаграммы LabVIEW и влияние на стиль оказывается отрицательным. Например, простое преобразование схемы производственного процесса в ВП даст вам диаграмму плохого стиля со множеством структур последовательности и циклов. Получится нечто среднее между ВП «Многоуровневый» и «Спагетти» из примеров главы 1, причем, все это будет занимать площадь ВП «Дотошный». Получится самая плохая программа, которая только может быть. Вместо этого нужно понять причины хорошего стиля LabVIEW и применить стандартные принципы проектирования к вашему ВП. Например, на основе шаблона «Конечный автомат» построено большинство других шаблонов программирования.

2.2.1. Поиск полезных источников

Где найти программы, которые облегчат проектирование и разработку одновременно с хорошим стилем? Ищите в сети ссылки, примеры, драйверы, библиотеки, коллег и консультантов. Начните с www.ni.com. Зона разработчиков (NI Developer Zone) – это форум, на котором всем нуждающимся помогут разобраться с

LabVIEW, найти статью, нужный участок кода и поддержку. *LabVIEW Tools Network* – это финансируемый NI раздел сайта внутри зоны разработчиков, на котором можно найти дополнительные библиотеки, книги, руководства для начинающих и многое другое. Также обратитесь на OpenG.org и сообщество LAVA. OpenG – это организованное сообщество, занимающееся разработкой и использованием открытых ресурсов для LabVIEW, адрес их сайта www.openg.org. Сообщество LAVA занимается свободным обменом идей по наиболее животрепещущим темам в LabVIEW, их адрес: www.lavausergroup.org.

Много информации есть и вне сети. Среда LabVIEW содержит сотни примеров ВП, ориентироваться по которым помогает Поисковик примеров NI Example Finder (**Help ⇒ Find Examples**). В справке LabVIEW приведены официальные описания и полезные советы. В вашем любимом книжном магазине (пусть даже сетевом) наверняка найдутся полезные книги. Также NI предлагает пройти курсы в сертифицированных учебных центрах (NI Certified Training Center) и множество полезных разработок по программе взаимопомощи NI Alliance Program. Эта программа объединяет во всемирную сеть более 600 консультантов, системных интеграторов, разработчиков, партнеров и экспертов. Например, Bloomy Controls удостоена звания Select Alliance Partner, на ее территории находятся два учебных центра.

Вернемся к нашему примеру с выхлопом с низким содержанием отравляющих веществ. Необходимо разработать несколько специальных приборов для анализа газа и реализовать статистический анализ данных. Допустим, что в зоне разработчиков NI (NI developer zone) мы не нашли драйверов для приборов. Ничего не дало и обращение к различным производителям. Но в библиотеку LabVIEW Enterprise Connectivity Toolset входит модуль для статистической обработки SPC Toolkit. В него входят все ВП и средства анализа, которые нам нужны. Воспользовавшись этим модулем, можно сэкономить значительные усилия. Также, снизив сложность и объем программы благодаря профессиональным ВП, входящим в этот модуль, мы повысили общий стиль программирования.

2.2.2. Разработка пробной версии

Очень часто стоит создать отдельную программу для испытания определенного прибора, оборудования, компонента или шаблона программирования. Например, для вас очень важна скорость сбора и обработки данных. Вы точно не знаете, что имеющийся компьютер, прибор сбора данных или шина передачи в вашем алгоритме справятся с задачей за нужное время. Необходимо написать отдельную программу, эмулирующую сбор данных на больших скоростях и алгоритм анализа, и проверить скорость ее выполнения. По результатам теста можно будет проверить схему, поискать альтернативные варианты, например более мощный прибор или алгоритм обработки или, в крайнем случае, изменить требования к программе.

В примере с системой выхлопа мы должны передавать данные контроллерам расхода массы, собирать данные нескольких анализаторов, синхронизованных

с точностью до 10 мс и с частотой 10 Гц, проводить статистический анализ и обновлять график. Это критические требования. Не имеет смысла начинать работу над другими функциями, пока мы не убедимся, что эти требования выполнимы. Нашим пробным прибором будет ВП, который несколько раз производит нужные операции, а мы в это время измерим его производительность.

По завершении испытаний обязательно сохраните пробный прибор. Часть его можно будет использовать и в финальной версии программы. Или окажется, что производительность программы заметно отличается от измеренной. Тогда можно будет вернуться к пробному прибору и по шагам проверить работу разных элементов. Если прибор перестал работать, значит, с момента последнего запуска изменилось оборудование, настройки системы или какие-либо другие параметры. Придется вернуться к предыдущей конфигурации, этот шаг называется санитарной проверкой программы. Если изменений нет, но программа не работает, значит, вы что-то пропустили.

Как вы уже поняли, такой пробный прибор положительно влияет на стиль программирования. Он позволяет удовлетворить критические требования, и благодаря ему вы сможете предсказать поведение программы. Использование уже проверенных участков кода гораздо реже требует глобальных изменений. Также он помогает выделить базовые конструктивные элементы программы: шаблоны структуры и данных. После проверки этих элементов количество изменений будет минимальным, а это, в свою очередь, по теореме 2.2 выразится в аккуратной программе.

Правило 2.3 Тестирование блоков программы с помощью отдельных ВП положительно влияет на стиль

Чего я терпеть не могу, так это небрежного обращения с такими пробными приборами. Мы все встречали разработчиков, пользующихся в прототипах именами, которые LabVIEW присваивает по умолчанию: numeric, numeric 2, Untitled 1. И после этого они хотят, чтобы было понятно, что этот прибор делает. На это способны только телепаты. Как мы уже говорили, хороший пробный прибор необходимо поместить в архив и пользоваться им для разрешения неполадок. Это возможно, только если он написан в хорошем стиле, правила которого одинаковы и для законченной программы, и для пробных приборов. Также хороший стиль поможет вам органично встроить его в свое приложение. Хороший стиль всегда пригодится.

2.2.3. Вернитесь к ТЗ

После проектирования программы и уверенности, что критические требования можно выполнить, необходимо вернуться к техническому заданию. Изменения неизбежны. Некоторые требования могут частично противоречить друг другу, придется идти на компромисс. ТЗ – это живой документ. Регулярно его пересматривайте и корректируйте. Избавляйтесь от устаревших требований, добавляйте новые, корректируйте в соответствии с расписанием. Подробнее изучите аспекты, которые были неясны сначала. Храните ТЗ в месте, где все члены команды могут с ним ознакомиться и проконсультироваться. Вносите изменения в соответствии с принятыми в компании правилами.

Вернемся к примеру с системой выхлопа. Нам пришлось изменить требования к сбору и анализу данных после финансовой оценки затрат и тестирования основных модулей. А именно, ВП из библиотеки SPC Toolkit позволили добавить более сложные графики, а не только простейший полосовой. Также пробная программа уточнила допустимую скорость сбора и обработки данных. В результате снизились уровень риска и время на каждую задачу.

2.3. Настройка среды LabVIEW

Пройдите, пожалуйста, небольшой тест:

- Начинаете ли вы большинство приложений с пустого листа?
- Используете ли вы стандартные настройки LabVIEW для цвета, шрифта, размера текста в элементах графического интерфейса?
- Создаете ли вы работающие, но приятные интерфейсы?

Не приходится ли вам тратить время на:

- Изменение привычных шаблонов?
- Поиск предыдущих проектов, чтобы воспользоваться их участками кода?
- Отключение отображения иконок терминалов и автоматической прокладки проводников?
- Выполнение повторяющихся операций в большинстве приложений?

Если вы хотя бы один раз ответили да, вам нужно настроить среду программирования. Если вы начинаете с пустого ВП и свойств элементов, заданных по умолчанию, вряд ли вам кажется, что привлекательный внешний вид лицевых панелей и простота использования не нужны в этом приложении или вы принципиально не хотите воспользоваться чем-то лучшим. Но, скорее всего, вы знаете подходящие друг к другу сочетания цветов, шрифтов и свойств. В этом случае вы начинаете с элементов по умолчанию, потому что ваши работы плохо организованы и вы не знаете, как удобно пользоваться существующими наработками. Ваши любимые шаблоны программирования и схемы пользовательского интерфейса должны быть легко доступны в виде шаблонов. Внешний вид лицевой панели и опции блок-диаграммы должны быть настроены под ваши предпочтения. В этом разделе мы узнаем, как настроить среду программирования и как это повысит стиль приложения.

2.3.1. Диалоговое окно опций LabVIEW

Диалоговое окно опций (LabVIEW Options) открывается из меню **Tools ⇒ Options** (Инструменты ⇒ Опции) и используется для настройки среды программирования. Многие из них напрямую влияют на стиль. Например, в опциях блок-

диаграммы я всегда отключаю **Show dots at wire junctions** (Показывать точки в контактах проводов), **Show subVI names when dropped** (Показывать имена ВПП при размещении) и **Place front panel terminals as icons** (Размещать терминалы как иконки). Без точек, меток и иконок диаграмма выглядит намного аккуратнее. Точки в месте контакта проводника позволяют отличить их от пересечения не связанных данных. Но внимание многих пользователей LabVIEW отвлекается на точки, потому что они могут обозначать приведение типа. При тройном щелчке на проводнике он выделится полностью, и точки в местах контакта излишни. Иконки терминалов и имена ВПП – это просто трата драгоценного места, я всегда отключаю их отображение. На рис. 2.2 приведены мои настройки для блок-диаграммы.

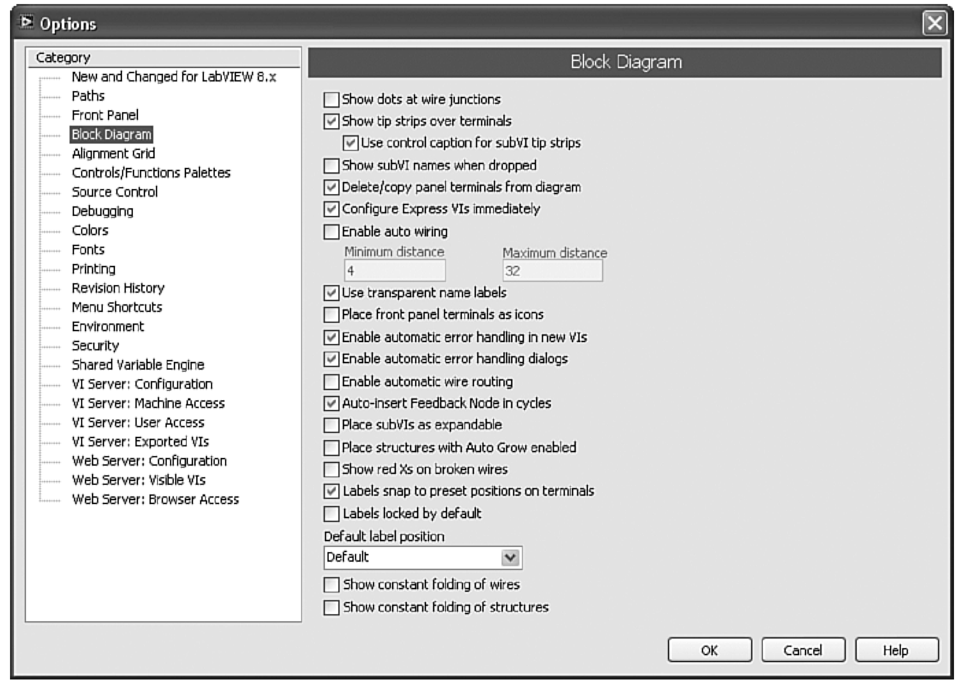


Рис. 2.2. Окно настройки опций LabVIEW, свойства блок-диаграммы

На лицевой панели большинство опций я оставляю по умолчанию. Бывалым разработчикам LabVIEW действительно нравятся прозрачные метки и современный стиль элементов управления. В предыдущих версиях мы тратили много времени, чтобы задать прозрачный фон для меток всех элементов всех лицевых панелей. Прозрачные метки выглядят гораздо лучше, чем выделенные из прошлых версий LabVIEW. Современный или трехмерный стиль элементов управления (Modern style controls) – это также большое достижение, которое впервые появилось в LabVIEW 6.0 и позволило сделать лицевые панели ВП похожими на свои железные аналоги. Также на лицевой панели полезна сетка (Alignment Grid), ее включает пункт **Show front panel grid** (Показать сетку). Сетка видна только при редактировании и помогает ровно разместить различные элементы.

Правило 2.4 Запишите опции LabVIEW и сохраните файл настроек (LabVIEW.ini)

В идеальном мире все разработчики одной компании установили бы себе одинаковые настройки LabVIEW. Это не только обеспечило бы одинаковый стиль, но и позволило бы работникам меняться машинами. Все настройки LabVIEW записываются в конфигурационный файл LabVIEW.ini в родной директории LabVIEW. Я бы посоветовал сохранить этот файл, чтобы можно было быстро перенести любимые настройки на другую машину. Также он позволяет быстро настроить под единый стиль несколько разных машин. Обратите внимание, что в разных версиях LabVIEW, платформах и операционных системах этот файл разный. Также в нем хранится место расположения последних использованных файлов и директорий, которые также уникальны для конкретной машины.

Большинство разработчиков LabVIEW часто меняют внешний вид палитр в зависимости от обстоятельств. Сюда относятся порядок категорий, режим их отображения и скрытия. Пользовательские палитры настраиваются с помощью меню **Tools ⇒ Advanced ⇒ Edit Palette Set** (Инструменты ⇒ Дополнительные ⇒ Редактировать палитры). Собственные палитры – это один из ключевых элементов повторного использования кода.

2.3.2. Повторное использование кода

Повторное использование кода необходимо во всех современных средах программирования. ВПП LabVIEW – это самодостаточный модуль, который можно вызвать из другого ВП статическим включением либо динамически, с помощью настройки вызова (**Call Setup**) или сервера ВП вызовом по ссылке (VI Server Call by Reference Node). ВПП очень удобны для повторного использования в других проектах. Также можно настроить внешний вид элементов управления и сохранить либо в виде отдельных файлов, либо в виде определения типа. Шаблоны программирования можно сохранить в виде ВП или как templates (шаблоны). Эти действия сэкономят вам много времени, в противном случае пришлось бы каждое приложение начинать с пустого листа. У каждого разработчика должна быть библиотека используемого им кода. Более того, в каждой организации, объединяющей группу разработчиков в общем доступе, должна быть такая библиотека. Ее выгоды очевидны: меньшее время разработки, общность работы и высокое качество в результате.

Мы все стремимся сэкономить время! Экономия времени при повторном использовании кода колоссальна. Зачем еще раз изобретать колесо? Попробуйте воспользоваться уже существующими шаблонами, ВП и элементами управления.

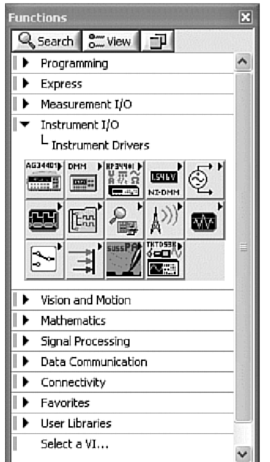


Рис. 2.3. Палитра функций с открытой палитрой драйверов. Каждая иконка соответствует драйверу из папки LabVIEW\instr.lib

Многие пользуются примерами LabVIEW. Они очень хорошо организованы и с удобным поиском. Своим исходным кодом пользуется гораздо меньше разработчиков, возможно, потому, что не очевиден оптимальный путь решения задачи.

Начнем с объекта, который чаще всего используется, но не создается. Это драйвер прибора от LabVIEW. Несмотря на то что в зоне разработчиков есть библиотека из тысяч драйверов приборов (Instrument Driver Network) на www.ni.com/idnet, программируемых приборов десятки тысяч. Большинству из нас приходилось писать драйвер самостоятельно или настраивать один из загруженных драйверов. Много организаций выкладывают свои драйверы для общего доступа. Чтобы ими было действительно просто пользоваться, они должны быть на палитре **Functions** (Функции) на компьютере разработчика. Для этого нужно поместить драйверы в папку LabVIEW\ instr.lib. Каждый раз при запуске LabVIEW считывает ее содержимое и добавляет палитры драйверов. Таким образом, вы можете легко воспользоваться скопированными откуда бы то ни было драйверами в среде LabVIEW. На рис. 2.3 показана палитра функций с открытой палитрой драйверов (**Functions**, **Instrument Drivers**).

Правило 2.5 Создавайте ВПП для повторного использования
- ***Они должны решать отдельные задачи***
- ***Используйте элементы управления, а не константы***
- ***Придерживайтесь хорошего стиля***

Многие из нас разрабатывали ВПП для преобразования строк и массивов, чтения и записи файлов, проведения вычислений и многого другого. Эти ВПП можно будет использовать в другой программе, если они удовлетворяют нескольким критериям. Во-первых, они должны решать отдельную, но общую задачу с разными параметрами. Иногда это означает, что придется заменить константу на терминал. Во-вторых, стиль ВПП должен быть на высоте, в соответствии со всеми рекомендациями этой книги.

Предположим, что мы разрабатываем приложение, в котором нужно обрабатывать пакеты данных от прибора, разделенных символами возврата каретки и перехода строки. Можно сделать ВПП, вызывающий в цикле функцию поиска по шаблону (Match Pattern). Если шаблон поиска для этой функции – строка из двух символов, ВПП будет не очень гибким и воспользоваться им еще раз будет сложно. Лучше заменить константу на управляющий элемент с именем «шаблон поиска» и назначить ему терминал. ВПП выполняет отдельную, но более общую задачу: обработку пакета данных с произвольным разделителем.

Правило 2.6 Библиотеки ваших приборов должны быть на палитре функций
- ***Скопируйте библиотеки приборов в папку LabVIEW\user.lib***
- ***Настройте палитры функций и элементов управления***

Последний шаг в настройке ВПП для повторного использования – сделать их легко-доступными разработчику. Поиск в архивах старых проектов – долгое и неблагодарное занятие. Прямо во время программирования можно предусмотреть, что данный ВПП понадобится вам в будущем, и сохранять такие программы отдельно. После этого можно скопировать библиотеку в папку LabVIEW, откуда они автоматически загрузятся в палитру. Папка LabVIEW\user.lib работает аналогично LabVIEW\instr.lib. Каждому файлу, папке, библиотеке LabVIEW ставит в соответствие функцию Пользовательской палитры (**Functions ⇒ User Libraries**).

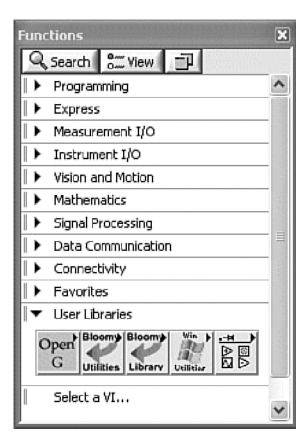
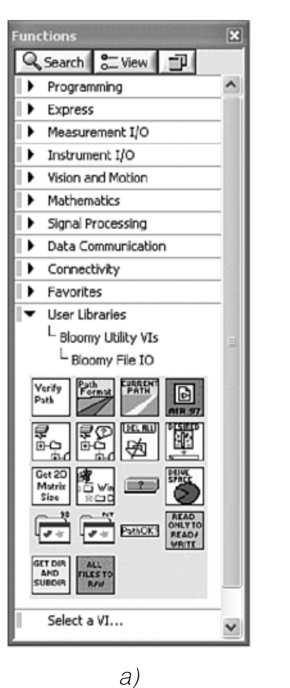


Рис. 2.4. Палитра функций с открытой пользовательской палитрой из пяти библиотек

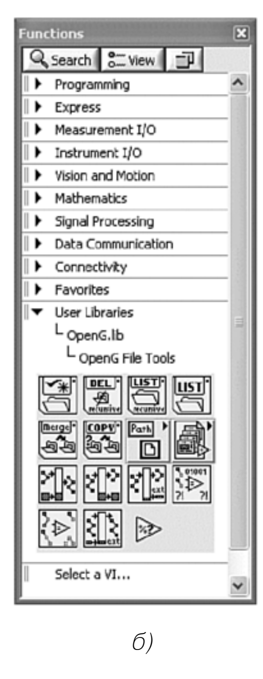
На рис. 2.4 приведен пример пользовательской палитры функций. В папке есть пять библиотек: OpenG, Bloomy Utilities, Bloomy Library, Win Utilities и библиотека без определенной иконки. Библиотекам и папкам без настроечного файла LabVIEW присваивает иконку по умолчанию. Свойства палитры, включая отображение папок и библиотек, а также расположение палитр низших уровней, меню, иконок и ВП настраиваются с помощью пунктов меню **Tools ⇒ Advanced ⇒ Edit Palette Set** (Инструменты ⇒ Дополнительные ⇒ Редактировать палитры).

Некоторые приборы в LabVIEW выполняют простые операции и расширяют возможности встроенных виртуальных приборов, они называются **utility VI** (служебные ВП). Полезно поместить их в палитру с аналогичными родными ВП, а не только в пользовательскую палитру. Например, в библиотеках OpenG и Bloomy есть палитра работы с файлами. Ее ВП показаны на рис. 2.5а и 2.5б. Однако удобно, если бы эти палитры были расположены в палитре чтения и записи файлов (**File I/O**). К счастью, LabVIEW позволяет это сделать. Как показано на рис. 2.5в, в палитре работы с файлами есть копии функций из пользовательских библиотек. Это сделано с помощью редактора палитр, который позволяет добавить новые подпалитры, связанные с файлами MNU, папками или библиотеками служебных ВП. Отметим, что сами файлы есть только в папке LabVIEW\user.lib. Файлы MNU, определяющие содержание палитры, находятся в следующей папке: <папка данных по умолчанию>/<LabVIEW <номер версии>>/palettes).

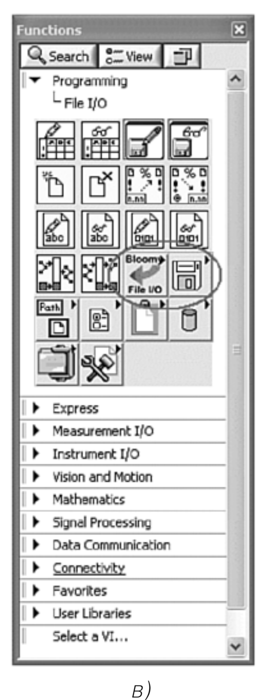
Достаточно часто приходится изменять элементы управления и индикаторы LabVIEW. Чаще всего настраивается тип данных элемента, изменяется размер, цвета и шрифт меток и внедренного текста. С помощью редактора элементов управле-



а)



б)



в)

Рис. 2.5. В библиотеках OpenG и Bloomy есть служебные ВП для работы с файлами, они находятся в папке LabVIEW\user.lib. После настройки палитр эти функции становятся доступными и из палитры ***File I/O***

ния можно кардинально изменить внешний вид и создать совершенно новый элемент. Результат работы можно сохранить в виде файла элемента управления, определения типа или строгого определения типа и пользоваться ими в других проектах. Чтобы они появились на палитре элементов управления, скопируйте их в папку LabVIEW\user.lib. По расширению файла LabVIEW поймет, куда его добавить: на палитру функций или элементов управления. На палитре функций будут файлы с расширением .vi и .vit, а на палитре элементов управления – файлы .ctl.

Шаблоны – это ВП с часто используемыми комбинациями структур, сохраненные с расширением .vit. К стандартным шаблонам относятся **SubVI with Error Handling** (ВПП с обработкой ошибок), **Dialog Using Events** (Диалог с событиями) и **Standard State Machine** (Конечный автомат). Все шаблоны можно открыть командой **New** (Создать), когда вы выбираете команду **VI or Project from Template** (ВП или проект по шаблону) в окне приветствия LabVIEW или в меню **File** (Файл). Это диалоговое окно показано на рис. 2.6.

Шаблон **SubVI with Error Handling** (ВПП с обработкой ошибок) состоит из лицевой панели с кластерами ввода и вывода ошибки, соединенными соответственно с нижним левым и правым контактом терминальной панели. Разработчик может добавлять новые элементы управления, присваивать им контакты и управлять блок-диаграммой в кадре **No Error** (Нет ошибки) структуры выбора. Этот шаблон экономит разработчику пару минут, но эти действия настолько стандартны и скучны, что его выгода очевидна.

ВП **Dialog Using Events** (Диалог с событиями) – это ВП, лицевая панель которого открывается при вызове и ждет действий пользователя с различными элементами лицевой панели. Поведение лицевой панели соответствует стандартным настройкам диалогового окна, есть две кнопки: **OK** и **Cancel**. Блок-диаграмма состоит из Цикла по условию со структурой событий, которая ожидает события **Value Change** (Изменение значения) с любым из элементов, после этого ВП останавливается. Если ваши диалоговые окна созданы на основе этого шаблона, графический интерфейс пользователя будет действовать и выглядеть аналогично системным окнам.

ВП **Standard State Machine** (Конечный автомат) – это стандартный шаблон из цикла со структурой выбора, сдвиговым регистром для передачи значений состояния и определения типа с названиями состояний. Конечный автомат используется очень широко и подробно обсуждается в главе 8 «Шаблоны».

Правило 2.7 Скопируйте свои шаблоны в папку LabVIEW\templates

Как показано на рис. 2.6а, при открытии нового файла предлагаются вышеупомянутые стандартные шаблоны LabVIEW. Вы можете добавить свои шаблоны в папку LabVIEW\templates. Они не только повысят производительность вашей работы, но и повысят стиль программирования. На рис. 2.6б приведены шаблоны Bloomy Controls.

2.4. Структура проекта, именование файлов и управление

В LabVIEW 8.0 появился Project Explorer (Проводник проекта) – интерфейс в виде древовидной структуры для управления файлами проекта. Свойства проекта хранятся в XML-файле проекта LabVIEW с расширением .lvproj. Файлы проекта могут быть где угодно, называться любым именем, работать со все расширяющимся набором целевых платформ. С этим связаны большие перспективы и не менее большие сложности для отдельных разработчиков и целых команд. Желательно договориться о правилах размещения, именования и управления файлами проекта.

Логичная структура, имена и средства управления исходным кодом для многих привычны. Но это не всегда верно. Например, кажется, что разработчики стараются сократить имена файлов. Я видел имена, состоящие из аббревиатур с подчеркиванием и номером, как будто ограниченные форматом 8.3 из DOS. Как ни удивительно, такая ситуация встречается достаточно часто. Мне приходилось работать с проектами из сотен подобных ВП в одной библиотеке, без ВП верхнего уровня. Причем создателя этого под рукой не было, документация отвратительная, хотелось застрелиться.

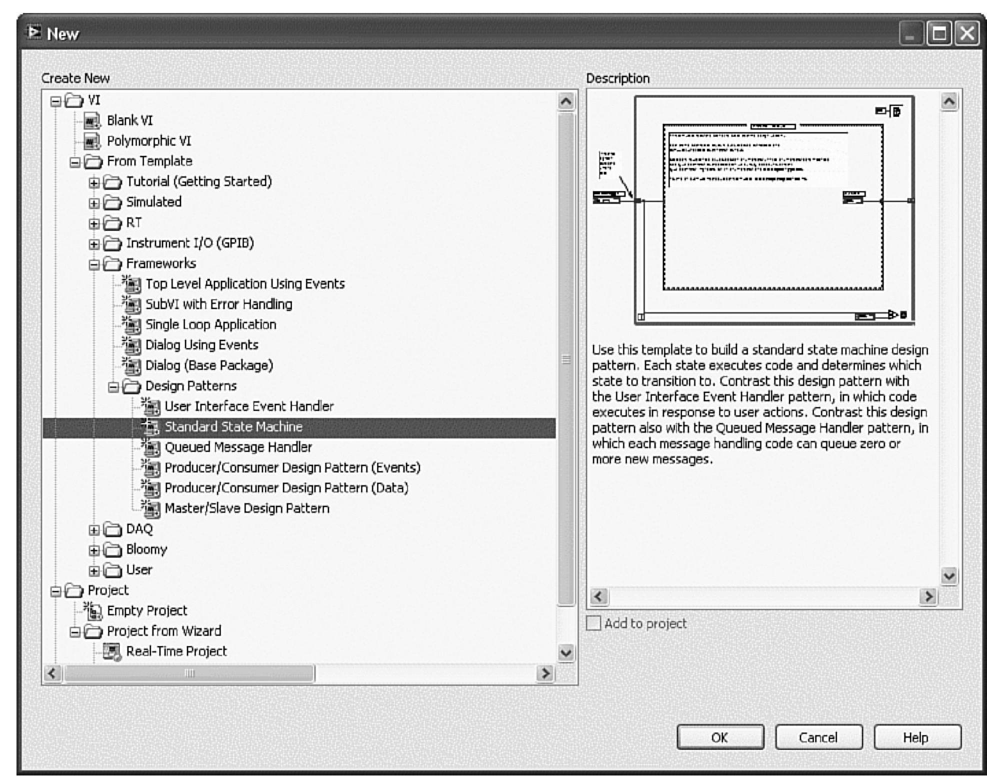


Рис. 2.6а. При открытии нового файла предлагаются стандартные шаблоны LabVIEW, в том числе и конечный автомат

Более того, я встречался с организациями с десятками ПК, на каждом были дюжины копий одного и того же приложения. Разработчики меняли константу (коэффициент усиления) на блок-диаграмме, диапазоны элементов управления или индикаторов и сохраняли ВП под новым именем. Некоторые, не все, из этих копий хранились на сервере, компакт-дисках и других носителях. При любом существенном изменении приходилось изучать каждую копию, искать отличия и, возможно, редактировать все варианты. Вместо этого нужно было один раз договориться о структуре хранения проекта, именовании файлов и управлении ими.

2.4.1. Расположение файлов

Правило 2.8 Структурируйте место расположения файлов проекта

Файлы проекта LabVIEW могут храниться где угодно, поэтому на первый взгляд хорошо организованный проект (в окне **Project explorer**) может запутать вас при попытке найти его файлы на диске, в сетевых папках, на съемных дисках и во всех

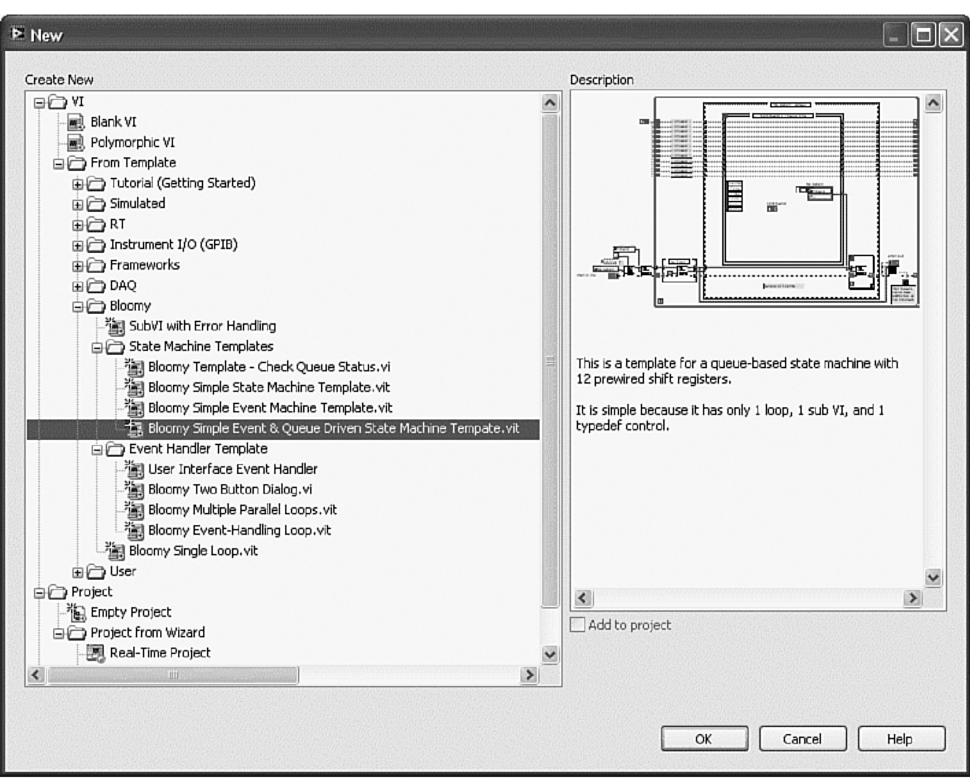


Рис. 2.6б. Шаблонами компании также удобно пользоваться, если они размещены в папке LabVIEW\templates

остальных местах. Без строгой организации хранения файлов остальные члены команды могут на вас сильно обидеться, пытаясь отыскать перемещенный файл. Также вам самим будет сложно управлять рассеянными файлами проекта вне LabVIEW, например, с помощью проводника. Поэтому организация файлов каждого проекта должна быть строго структурирована. Разработайте иерархию папок для размещения файлов, например, как показано на рис. 2.7а: у проекта есть 5 папок: Application Build, Data, Documentation, Graphics и LV Source. В первой, Application Build, расположены запускаемые файлы и средства установки. В папке Data хранятся примеры данных, необходимые разработчику. В папке Documentation находится документация проекта: технические описания, руководства к приборам и другие файлы, поставляемые вместе с ПО, например файлы справки. В папке Graphics расположены необходимые графические файлы: снимки экрана, логотипы компании и иконки. Папка LV Source предназначена для исходных файлов проекта. Исходные файлы, которые используются в проекте, но не изменяются специально для него, хранятся отдельно и используются в нескольких проектах. К ним относятся служебные ВП и компоненты, например драйверы приборов.

Правило 2.9 Структура папок проекта должна соответствовать иерархии приложения

Пять папок проекта разделяют файлы на основные группы. Если файлы проекта разделены по этим группам, а папки расположены в одной директории проекта, то можно сказать, что ваш проект хоть как-то структурирован. Однако для лучшей организации файловая структура может быть сложнее. Очень полезно поддерживать структуру папки исходных файлов проекта соответствующей архитектуре приложения. Например, чаще всего она содержит папки Analysis (Анализ), Configuration (Настройка), DAQ (Сбор данных), Data Manipulation (Управление данными), File IO (Запись и чтение файлов) и User Interface (Пользовательский интерфейс). Обычно ВП верхнего уровня проекта находится в корне папки LV Source, а все ВПП разделены по вышеперечисленным папкам. Пример структуры папок сложного приложения приведен на рис. 2.7б.

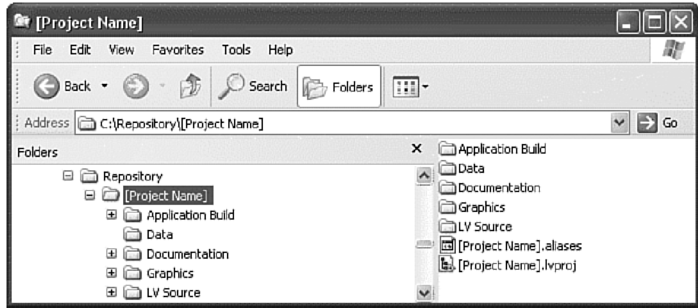


Рис. 2.7а. Структура проекта с основными папками

С пользой организованного хранения файлов соглашаются многие разработчики. Однако, как это бывает со многими правилами, благие намерения распадаются под давлением внешних обстоятельств. Не приучив себя к порядку, знание многих правил окажется для вас бесполезным, включая и правила файловой структуры проекта. В проекте LabVIEW, как и во многих других средствах управления проектами, при перемещении файлов на диске могут возникнуть проблемы обращения к ним, то есть линковки. Секрет поддержания строгой организации файлов следующий:

Правило 2.10 Иерархия папок должна быть создана перед написанием программы

Если вы соблюдаете это правило, проблем с организацией файлов у вас не будет: просто сохраняйте файл в нужную папку, открывайте их и редактируйте. Если структура папок уже существует, эти действия тривиальны. Более того, вам

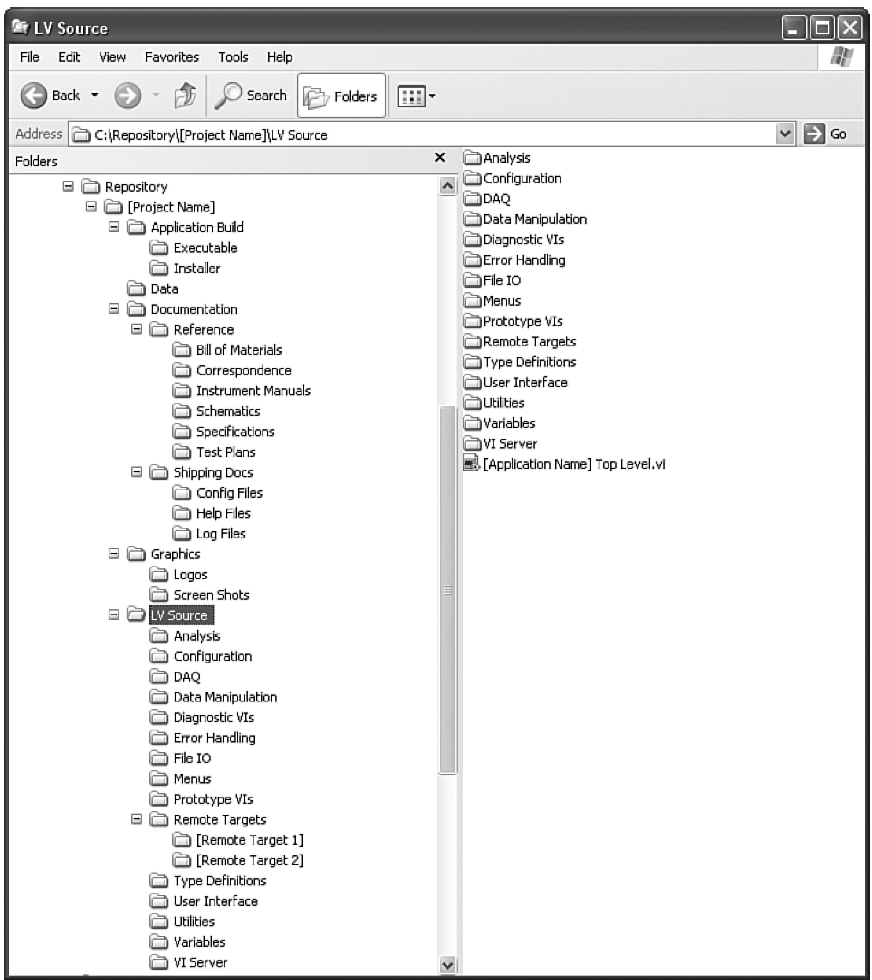


Рис. 2.7б. Вложенные папки позволяют легче структурировать проект и помогают при работе с крупномасштабными приложениями

не придется создавать структуру папок с нуля для каждого проекта. В структуре большинства проектов есть много общего: одинаковые категории файлов (большинство из них приведено на рис. 2.7б). Вам понадобится один шаблон со структурой папок, который можно скопировать в начале нового проекта.

В этом шаблоне могут быть не только папки, но и шаблон файла проекта, ВП верхнего уровня, документация и другие объекты. Такая организованная структура папок и файлов проекта – многообещающая основа для приложения с хорошим стилем.

Наконец, заметим: самое важное – осознать, что шаблон структуры – это только предпосылка для хорошего проекта. Нельзя распределить все файлы любого

приложения в раз и навсегда заданный шаблон, его придется настраивать: добавлять и удалять папки в соответствии с требованиями приложения. Насколько это возможно, сделайте все изменения до программирования, чтобы не перемещать файлы потом.

2.4.2. Проект LabVIEW

После создания строгой файловой структуры перейдем к проекту LabVIEW. Окно **Project Explorer** помогает управлять файлами проекта и их иерархией для отражения структуры приложения. Если у вас уже есть структура папок, работа по организации проекта не займет много времени. Откройте новый проект или шаблон, в контекстном меню объекта My Computer выберите пункт **Add Folder** (Добавить папку), найдите место расположения проекта и выберите существующую папку. Повторите этот процесс для каждой папки, которая понадобится в проекте. К ним обычно относятся папки с исходными файлами, технические описания, руководства к приборам и другие служебные данные. Пример структуры проекта приведен на рис. 2.8. В отличие от объектов на диске, вы можете перемещать элементы проекта, когда вам захочется. Размещение файлов в дереве проекта не влияет на их место на диске.

Правило 2.11 Если это возможно, объединяйте исходные файлы проекта LabVIEW в библиотеки проекта

Библиотека проекта LabVIEW – это файл с расширением .lvlib, в котором описываются свойства взаимосвязанных файлов LabVIEW. К таким свойствам относятся префикс имени, версия, права доступа, пароль, иконки палитры и элементы меню. Библиотеки проекта предназначены для объединения исходных файлов, выполняющих взаимосвязанные операции. Библиотеки проекта предотвращают конфликт при открытии файлов с одним именем. Перед именем всех исходных файлов проекта появляется имя библиотеки, это позволяет открыть файлы с одним именем. Например, у драйверов разных приборов может быть ВП инициализации Init VI или закрытия Close VI, их можно открыть одновременно. Также библиотека позволяет разделить ВП на общие (public), которые доступны всем пользователям, и защищенные (private) – специально для разработчика. Обратите внимание, что драйверы приборов, библиотеки и модули до версии 8.0 обычно представляют собой архивированные файлы с расширением .llb. Этот тип файлов отличается от библиотек проекта: он хранит все файлы внутри себя, а не только описания к ним.

По сравнению с папками и библиотеками проекта у файлов llb есть определенные недостатки. Это слабое внутреннее структурирование, отсутствие совместимости с системными средствами управления файлами, большее время загрузки и сохранения, риск повреждения всех файлов. Файлы внутри них могут быть либо верхнего уровня, либо нет. Внутренние файлы нельзя найти средствами Windows.

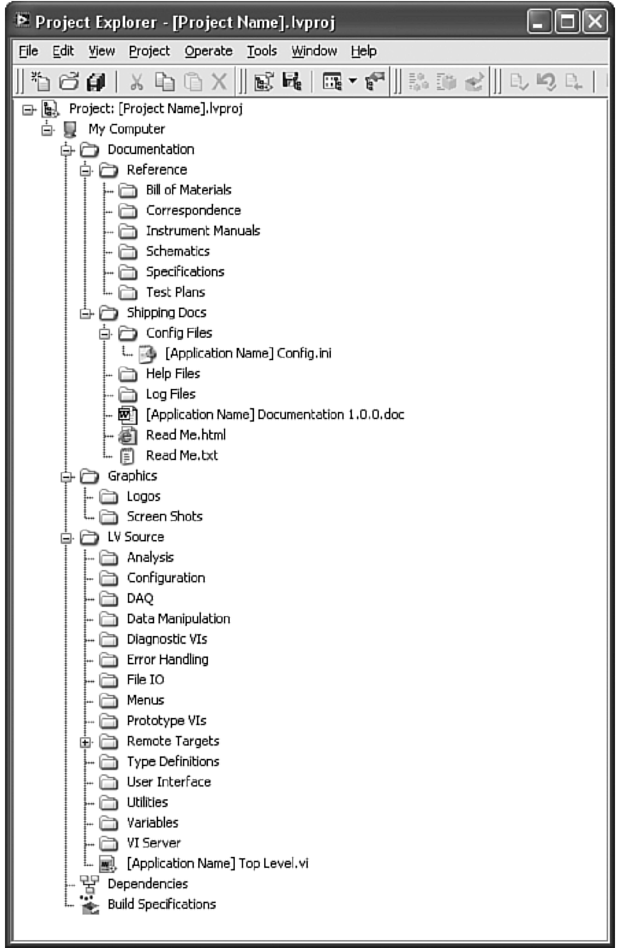


Рис. 2.8. Иерархия папок в проекте LabVIEW соответствует структуре приложения, аналогично организации папок

При открытии и закрытии файлов определенное время тратится на его архивирование. И наконец, при повреждении одного файла llb вы потеряете все содержащиеся в нем файлы.

К счастью, эти файлы можно преобразовать в папки с файлом библиотеки проекта. Во-первых, с помощью меню **Tools ⇒ LLB Manager** (Инструменты ⇒ Менеджер библиотек) преобразуйте файл llb в папку с ВП. После этого добавьте эту папку в проект в Project Explorer, в проекте появится папка со всеми содержащимися в ней ВП. Преобразуйте ее в библиотеку проекта с помощью пункта контекстного меню **Convert to library**. Задайте имя библиотеки и сохраните ее.

2.4.3. Именование файлов

В этом разделе перечислены правила именования исходных файлов LabVIEW.

Правило 2.12 Имена исходных файлов должны быть описательными и уникальными

Имя файла должно кратко описывать его назначение. Как правило, имя файла вместе с именем библиотеки должно быть уникальным для каждого файла. Представьте себе, что вам потребуется отлучиться в середине важного проекта. У ваших коллег должна быть возможность найти исходные файлы, понять их стиль и продолжить работу над проектом даже в ваше отсутствие. Если они смогут это сделать, ваш стиль действительно хороший.

Правило 2.13 Не используйте аббревиатур

Избегайте правил именования, основывающихся на использовании аббревиатур, акронимов и чисел. Например, имя исходного файла IPS_Osc_H_to_0.vi, может быть, и значило что-нибудь для его создателя, но для всех остальных это бессмыслица. Напишите столько полных слов, сколько нужно, чтобы описать файл. Избегайте использовать символы, которые не воспринимаются некоторыми файловыми системами: \, /, :, ~. Менеджер llb (LLB Manager) и Анализатор ВП (VI Analyzer) помогут проверить межплатформенную совместимость имен файлов.

Правило 2.14 Не пользуйтесь именами, которые LabVIEW дает по умолчанию

Никогда не называйте файлы так, как предлагает LabVIEW при создании нового файла: Untitled или Control с номером. Избегайте их любой ценой. По моему мнению, это одно из самых серьезных нарушений хорошего стиля.

Многие разработчики к имени файла добавляют номер версии. Для файлов внутри библиотек проекта или при использовании средств управления исходниками это излишне. Также встроенная функция LabVIEW **VI revision history** (История изменения ВП) в меню **Tool** ⇒ **Source Control** ⇒ **Show History** (Инструменты ⇒ Управление исходниками ⇒ Показать историю) хранит номер версии вашего ВП. В настройках можно указать автоматическое увеличение номера версии и предложение внести комментарии каждый раз при сохранении или закрытии измененного ВП. Это единственный метод управления номером версии прямо из исходного файла.

У фирменных драйверов приборов, выпущенных до версии 8.0, имеется специальное соглашение именования файлов. У имени каждого файла есть префикс с аббревиатурой производителя и номером модели прибора. Например, у цифрового осциллографа Keithley 2000 будет префикс ke2000. Также все требуемые файлы явно перечислены в стандарте функционального тела драйвера прибора VXI (VXIplug&play Instrument Driver Functional Body). Этот стандарт можно загрузить с www.vxipnp.org. Но у современных драйверов описание прибора хранится в библиотеке проекта. Поэтому префикс прибора в имени больше не требуется.

Правило 2.15 Укажите ВП верхнего уровня

В каждом проекте есть ВП с особым статусом, это ВП верхнего уровня. Всегда выделяйте их, в том числе и местоположением: файл в структуре папок и в Project explorer должен лежать близко к корневой директории проекта. Имя файла также должно быть специальным, например имя_проекта_main.vi. Этот ВП не включается в библиотеку, поэтому название проекта перенесено в имя файла, это обеспечивает его уникальность.

2.4.4. Управление исходниками

Управление исходниками – это процесс организации, хранения, совместного использования исходных файлов в команде разработчиков. Для этих действий предназначены программы сторонних производителей, многие из них интегрируются в LabVIEW и позволяют осуществлять основные действия прямо из среды проекта. LabVIEW версии 8.2 поддерживает Microsoft Visual SourceSafe, Perforce, MKS Source Integrity, IBM Rational ClearCase, Serena Version Manager (PVCS), Seapine Surround SCM, Borland StarTeam, Telelogic Synergy, PushOK (с дополнениями CVS и SVN) и ionForge Evolution. В Windows пользователям предлагается Microsoft Source Code Control Interface (Интерфейс управления исходным кодом Microsoft). На других платформах поддерживается только Perforce с интерфейсом в виде командной строки.

Особенно необходимо управление исходниками при командной работе. Благодаря этим инструментам разработчики могут работать над одним проектом с регулируемым доступом к файлам и папкам. Разумеется, невозможно одновременно редактировать один и тот же файл. Также большинство программ позволяют настроить права доступа для каждого пользователя. Например, разрешение на изменение документации проекта могут иметь только менеджеры, а редактировать файлы исходного кода только разработчики. Также некоторые программы не ограничивают доступ к файлам и занимаются решением конфликтов при сохранении различных изменений.

Средства управления исходниками регистрируют изменения всех типов файлов и хранят резервную копию. Эти функции полезны не только команде, но и отдельным разработчикам. Также умение работать со средствами управления исходниками необходимо для получения многих сертификатов, например ISO 9000.

Правило 2.16 Следуйте правилам внесения изменений, принятым в вашей компании

Если в вашей компании внесение изменений (CM – change management) организовано, соблюдайте установленные рекомендации, включая и работу с исходниками. Правила могут описывать определенные программные средства, действия и настройки среды разработки. При работе под контролем средств управления исходниками файлы проекта обычно находятся в определенном месте и защищены правами доступа. При создании новых файлов могут быть действия, связанные с внесением файлов в эту систему. Более подробная информация приведена в системе помощи LabVIEW.

Правило 2.17 Не стоит изменять расположение файлов

Необходимо подчеркнуть, что после начала работы файлы проекта можно перемещать в Project explorer, но не на диске. В противном случае разорвутся связи файлов в проекте и в средствах управления исходниками.

Вот теперь мы готовы приступить к программированию

Ссылки

1. Howard M. Kanare. Writing the Laboratory Notebook. Washington D.C.: American Chemical Society, 1985. (Ведение лабораторных записей).
2. Ресурсы для общего пользования находятся по адресу www.bloomy.com/resources.
3. Ссылки на руководства по программированию:

- Larman, Craig. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, second edition. Upper Saddle River, NJ: Prentice Hall PTR, 2001. (Использование UML и шаблонов. Введение в объектно-ориентированный анализ и разработку и стадии приложения);
- McConnell, Steven. Code Complete, second edition. Redmond, WA: Microsoft Press, 2004. (Законченный код);
- McConnell, Steven. Software Project Survival Guide. Redmond, WA: Microsoft Press, 1997. (Руководство по выживанию для программистов).

4. Bloomy Controls – партнер-интегратор NI (NI Select Integration Partner), предлагающий разработку систем и обучающие курсы в северо-восточной части США, офисы расположены в Виндзоре, Коннектикуте, Массачусетсе и Нью-Джерси.

3 Стиль лицевой панели

Эта часть программы LabVIEW называется лицевой панелью, потому что она немного похожа на лицевую панель «железных» приборов. Когда LabVIEW впервые увидела свет в 1986 году, основной ее целью было управление приборами по КОП или через последовательный порт. Как много воды утекло! Сегодня LabVIEW используется в широчайшем спектре приложений, отраслей промышленности и приборостроения. Сложность приложений несоизмерима со старыми задачами. Эволюция не могла не затронуть и лицевую панель. Такие современные средства, как трехмерные элементы управления, различные типы панелей, оболочки, узлы свойств, сервер ВП, структура событий и субпанели, вывели программы LabVIEW на новый уровень.

Несмотря на множество нововведений, многие из приемов, знакомых по ранним версиям LabVIEW, применимы и до сих пор. Новые средства часто предназначены для организации и облегчения создания графического пользовательского интерфейса. В этой главе приведены советы, которые помогут вам сделать лицевую панель ВП более наглядной, удобной и соответствующей промышленным стандартам. Начнем с некоторых полезных терминов, без которых невозможно двигаться дальше.

ВП верхнего уровня (**Top level VI**), как следует из названия, находятся на вершине иерархической лестницы всех ВП приложения, их лицевая панель – основная видимая пользователю часть программы. У некоторых приложений ВП верхнего уровня только один, обычно его называют main.vi. В других приложениях ВП верхнего уровня запускает другие высокоуровневые ВП с пользовательскими панелями, эти ВП также называются ВП верхнего уровня.

Диалоговые ВП (**Dialog VI**) – это ВП, связанные с пользовательским интерфейсом, которые требуют ввода определенных данных. У них гораздо меньше вложенных ВПП и более слабое функциональное назначение, чем у ВП верхнего уровня. Их основная цель – взаимодействие с пользователем. Также эти ВП обычно делают модальными (modal), чтобы пользователь не мог перейти к другому окну приложения, пока не введет нужные данные. При выборе типа ВП диалог (dialog) в окне свойств ВП на закладке внешнего вида (**File ⇒ VI Properties ⇒ Window Appearance**) LabVIEW автоматически устанавливает свойство Модальный. Но в данной книге мы не раз встретимся с окнами диалога, настройки кото-

рых отличаются от устанавливаемых по умолчанию, в частности, некоторые будут немодальными.

Все ВП, лицевая панель которых отображается пользователю, включая диалоговые ВП и ВП верхнего уровня, называются ВП пользовательского интерфейса (**GUI VIs**). ВП, лицевая панель которых скрыта от пользователя во время работы программы, – это виртуальные подприборы (ВПП). Обратите внимание, что эти определения не запрещают вызывать ВП верхнего уровня или диалоговые окна из других ВП. В рамках этой книги ВП пользовательского интерфейса (иногда просто ВП) – единственные ВП, с которыми работает пользователь, ВПП видны только разработчику по определению. Поэтому многие свойства и рекомендации для лицевых панелей ВП и ВПП будут отличаться. Если такие отличия есть, в каждой части главы будет специально оговорено, к какому типу ВП относится данный совет.

ВП, предназначенные для персональных компьютеров в офисах, лабораториях и других личных вычислений, если это важно подчеркнуть, мы будем называть ВП **desktop**. Программы, предназначенные для управления оборудованием, – это промышленные ВП (**Industrial GUI VIs**). Внешний вид обычных ВП выполнен в стиле других программ системы, а промышленные ВП предназначены для управления оборудованием вне зависимости от платформы компьютера, на который они установлены. Это накладывает свои ограничения, и опять же советы для этих двух групп ВП могут различаться.

Многие ВПП разработаны специально для конкретного приложения, другие выполняют общие функции и используются многократно в разных задачах. Теоретически затрачиваемые усилия можно ограничивать в зависимости от такого разделения ВПП. Но на практике оказывается выгоднее считать, что все ВП будут использоваться многократно, качество и стиль ВПП будут заметно выше. Иногда я разделяю правила создания коммерческих (**commercial**) ВПП и стандартных (**standard**). Первые – это составные части проектов, таких как библиотеки разработки или драйверы приборов. Стандартные ВПП предназначены для многократного использования, но их цель – улучшение стиля программы. Если это не оговорено особо, предлагаемые советы не будут различать эти типы ВПП.

В главе представлены примеры – как подтверждающие или нарушающие правила. Тем не менее я не собираюсь описывать, как создавать графический интерфейс; кому это нужно, обращайтесь к книге Дэвида Риттера[1] (David Ritter) «LabVIEW GUI Essential Techniques» (Техника создания пользовательского интерфейса), это достаточно полное руководство. Также, если вы создаете приложение для конкретной операционной системы, прочитайте рекомендации по стилю приложений этой системы. Цель этой главы – сделать лицевую панель ваших ВП наглядной, удобной и соответствующей промышленным стандартам.

3.1. Расположение

Расположение (Layout) – это организация элементов на лицевой панели, которое может облегчить работу как пользователю, так и программисту. В разделе приведены правила расположения элементов лицевой панели: общие правила и отдельно для ВП и ВПП.

3.1.1. Общие правила

Эти правила относятся к лицевым панелям всех ВП – как пользовательского интерфейса, так и подприборов.

Правило 3.1 Разделяйте логически не связанные элементы с помощью закладок, кластеров, элементов оформления и промежутков

Выделите логически связанные элементы управления и поместите их в одну группу, например, как показано на рис. 3.1. Это приложение предназначено для управления двумя одинаковыми тестовыми стендами: Station 1 и Station 2. Все элементы управления и отображения размещены на одной лицевой панели. Можно выделить несколько логических групп. Во-первых, вся панель разделена на две половины, обозначенные **Station One** и **Station Two** слева и справа соответственно. Дополнительно элементы разных стендов и внутри каждой группы разделены элементами оформления (decorations). Внутри каждой группы есть две основные

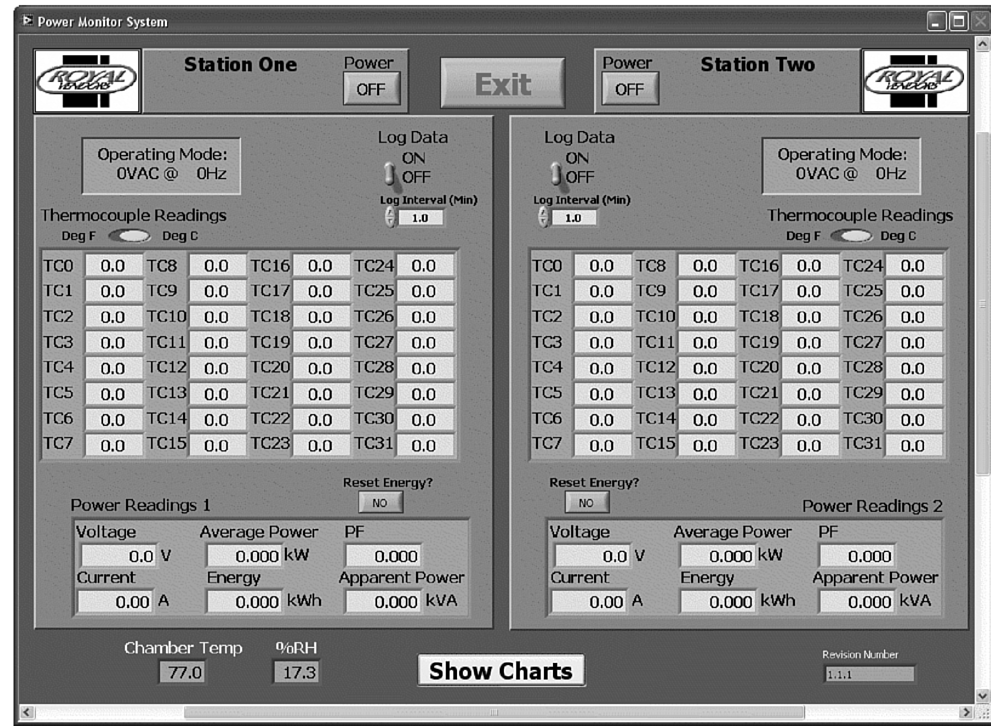


Рис. 3.1. На лицевой панели ВП элементы, относящиеся к разным измерительным стендам, разделены с помощью кластеров, элементов оформления и промежутков

подгруппы: температура и мощность. В центре расположено 32 цифровых элемента в кластере, отображающие температуру; внизу расположено 6 данных мощности также в виде цифровых элементов в кластере. Границы и более светлый фон еще лучше выделяют содержимое каждого кластера.

Правило 3.2 Придерживайтесь симметричного расположения элементов

Мои любимые средства редактирования в LabVIEW – выровнять и расположить объекты (**Align and Distribute Objects**), вынесенные на инструментальную панель. Я ими пользовался при редактировании каждой лицевой панели, будь то пользовательский ВП или подприбор. Они позволяют создавать эффективные приложения на высоком уровне. Достаточно один раз взглянуть на лицевую панель, и станет понятно, пользовался ими разработчик или нет. Панели с ровно расположенными элементами выглядят симметрично и аккуратно, в противном случае может получиться мешанина. Причин отказываться от этих средств я не вижу.

Лицевая панель на рис. 3.1 симметрична, кажется, что правая часть – зеркальное отражение левой. Индикаторы в кластерах также расположены ровно и равномерно: 32 индикатора температуры разделены на 4 колонки из 8 элементов, находящихся максимально близко друг к другу, но нигде не перекрываясь. Чтобы это сделать, выделите несколько индикаторов и воспользуйтесь инструментом вертикального сжатия (**Vertical Compress**) из меню расположения (**Distribute Objects**). После этого выровняйте по правому краю все объекты (меню **Align Objects**). Добившись идеальной колонки индикаторов, сгруппируйте элементы с помощью команды группировать (**Group**) из меню управления порядком расположения (**Reorder**) и сделайте еще 3 копии. Выделите получившиеся группы и аккуратно разместите их с помощью команд меню выравнивания объектов. С помощью аналогичных действий сделаны три колонки из двух индикаторов в кластере отображения мощности. Обратите внимание, что индикаторы температуры расположены более компактно, их метки находятся слева, и элементы могут примыкать друг к другу.

Группировка используется для объединения объектов. Чтобы разделить логически несвязанные объекты, добавьте между ними промежутки. На рис. 3.1 две половины лицевой панели разделяются полосой фона. Аналогично между кластерами температур и мощности также есть небольшой промежуток. Пустые места сверху и снизу панели выделяют окно приложения и позволяют пользователю сосредоточить внимание в центре.

Правила симметричного и группированного размещения элементов применимы не только к ВП пользовательского интерфейса. Эти советы помогут вам сделать ваши ВПП на высоком уровне, некоторые примеры и дополнительные правила приведены в разделе «Текст панели ВПП».

Правило 3.3 Размер объектов одного назначения должен быть одинаковым

У элементов похожего назначения и важности размер должен быть одинаковым. Иногда с самого начала сложно сказать, какого размера должны быть элементы группы. Например, на рис. 3.1, индикаторы мощности тесно связаны логически, но требуют разной точности: от одного до трех знаков. Казалось бы, что менее точные элементы должны быть уже, но так будет плохо. Сделайте ширину этих индикаторов достаточной для самого широкого числа. Управлять размером нескольких элементов позволяют команды меню **Resize Objects**. Выберите все необходимые элементы и сделайте их ширину равной максимальному значению среди них (команда **Maximum Width**). Также можно задать размеры объектов минимальному среди выбранных с помощью других команд этого меню.

3.1.2. Панель пользовательского ВП

В этом разделе перечислены советы, касающиеся только ВП пользовательского интерфейса.

Правило 3.4 Разверните окно ВП верхнего уровня промышленного приложения на весь экран

Большинство промышленных приложений – это отдельный интерфейс к управляемому ими прибору или оборудованию. Другие системные объекты оператору видеть совсем не обязательно, поэтому часто при запуске системы управляющее приложение запускается автоматически, а его лицевая панель занимает весь экран. Для этого установите свойство ВП разворачивать панель на весь экран во время работы (**File ⇒ VI Properties ⇒ Run Time Position ⇒ Maximized Window** из выпадающего меню). Обратите внимание, что при изменении разрешения ранее невидимая часть лицевой панели появится на экране и оператор доберется до специально скрытых, расположенных ниже границы элементов. Лучше сделать эти элементы невидимыми.

Правило 3.5 Диалоговые окна должны быть небольшими

Правило 3.6 Располагайте диалоговые окна в центре экрана

Диалоговые окна должны быть небольшими и располагаться в центре экрана. Если в свойствах соответствующего ВП указать расположение в центре во время работы (**Centered** Window Run Time Position), то вне зависимости от разрешения экрана окно появится в центре. Без этого расположение окна определяется координатами во время последнего сохранения.

Разрешение экрана влияет на внешний вид и размещение лицевой панели. Если приложение будет работать при разных разрешениях, то укажите свойства **Maintain proportions for different monitor resolutions** (Сохранять пропорции для различных разрешений экрана) и **Scale all objects on the front panel as the panel**

resizes (Масштабировать объекты при изменении размера лицевой панели). Эти свойства находятся в категории **Window size** (Размер окна), также ими можно программно управлять с помощью свойств класса ВП сервера ВП. Если ваше приложение состоит из нескольких окон (одно рядом с другим), создайте служебную программу, которая будет изменять их местоположение в зависимости от разрешения. Чтобы получить разрешение монитора, воспользуйтесь функциями сервера ВП для обращения к свойству **Display.All Monitors** (Дисплей.Все мониторы).

Правило 3.7 Пользуйтесь диалоговыми окнами LabVIEW для настольных приложений, избегайте их в промышленных задачах

Диалоговые окна LabVIEW запускаются с помощью функций палитры **Dialog & User Interface** (Диалог и пользовательский интерфейс). LabVIEW предлагает диалоговые окна с одной, двумя и тремя кнопками, а также экспресс ВП **Prompt for User Input and Display Message to User** (Окно ввода данных с сообщением). Эти функции просты в использовании и отображают диалоговые окна в стиле операционной системы компьютера. Это очень удобно для переноса приложения с одной платформы на другую, однако внешний вид окон менять нельзя. Также эти окна и их элементы обычно меньше объектов в промышленных ВП. Если оператор находится не прямо за столом с монитором, ему будет трудно воспринять данные. Это основная причина, по которой стандартные диалоговые окна не используются в промышленных приложениях. Во многих промышленных приложениях с несколькими задачами на стандартное диалоговое окно можно не обратить внимания. Однако диалоговые окна полезны и в промышленных приложениях, например для вывода сообщения об ошибке. К счастью, диалоговое окно несложно создать самостоятельно. Два одинаковых окна с разным назначением показаны на рис. 3.2.

Правило 3.8 Системные элементы управления используются для настольных приложений, трехмерные элементы – для промышленных

Диалоговые окна специального вида используются как в настольных, так и в промышленных приложениях. В первом случае просто выберите **Dialog** (Диалог) из вариантов внешнего вида окна (**Window Appearance**) и пользуйтесь элементами с палитры **System** (Системные). Цвет фона, поведение и другие свойства приложения будут соответствовать свойствам операционной системы. Аналогично стиль элементов управления, отображения и оформления также соответствует системным настройкам.

Оформление диалоговых окон в стиле операционной системы не рекомендуется для промышленных приложений. Пользовательские ВП этого типа, включая диалоговые окна, должны соответствовать решаемой задаче. Как уже было сказано, обычно это означает больший размер панелей, объектов, текста и объемных элементов управления.

Рис. 3.2. Стандартный диалог с двумя кнопками (слева) хорошо подходит для настольных приложений. Аналогичное диалоговое окно для промышленных приложений (справа) выделяется гораздо лучше

Правило 3.9 Более важные элементы управления в промышленных приложениях должны быть больше по размеру и располагаться ближе к центру окна

Размер по умолчанию индикаторов и элементов управления подходит для большинства настольных приложений, но не для промышленных задач. Важные объекты должны быть хорошо видны с нескольких метров от монитора. Чем важнее объект, тем больше он должен быть. Расположение и размер объекта подчеркивают его важность. Большие элементы в центре сразу бросаются в глаза. Увеличьте размер важных индикаторов и элементов управления и переместите их ближе к центру.

На рис. 3.3 приведено два диалоговых ВП для тестирования стереофонического радиоприемника. Эти ВП предлагают пользователю настроить параметры измерения и громкость приемника. При измерении силы звука ВП генерирует нужный сигнал. Версия приложения для настольного тестирования приведена на рис. 3.3а. Лицевая панель – это окно с набором свойств диалога и состоит только из элементов управления и отображения с системным стилем. Размер элементов и шрифта – стандартный для операционной системы. Пользователь сидит рядом с монитором и задает все параметры. Это приложение удобно, например, для лаборатории, когда все элементы системы расположены недалеко один от другого.

Промышленный аналог этого приложения приведен на рис. 3.3б, лицевая панель состоит из трехмерных элементов управления, с одной кнопкой **OK**. Индикаторы левого и правого уровня сигнала (**Left**, **Right Audio Output**) большого размера и расположены по середине окна, они сразу привлекают внимание. Перечисленные особенности помогают оператору оперативнее выполнить настройку. Достаточно взглянуть на панель, сразу становятся понятны наиболее важные элементы, с которыми удобно работать. Промышленные ВП используются при серийном производстве и измерениях.

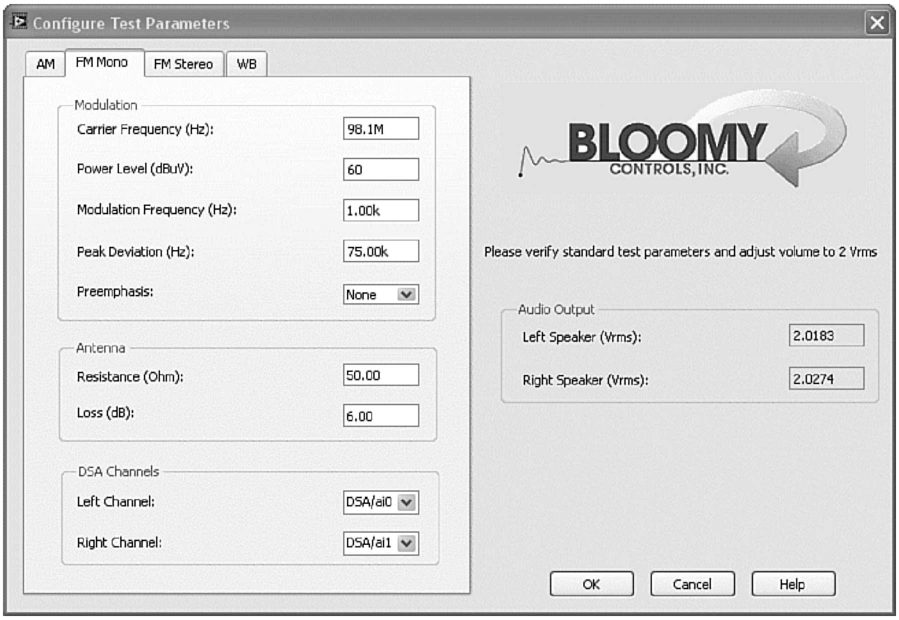


Рис. 3.3а. Это диалоговое окно предназначено для настольного приложения, например, исследовательской лаборатории

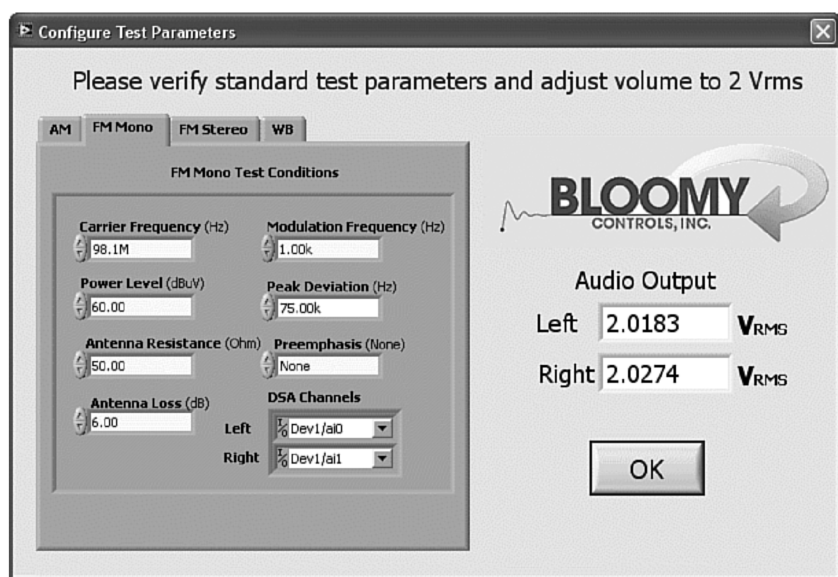


Рис. 3.3б. Это диалоговое окно функционально мало отличается от представленного на рис. 3.3а, но предназначено для промышленного серийного тестирования

Правило 3.10 Ограничивайте количество информации на лицевой панели
- ***На панели не должно быть больше 7 групп из 7 различных элементов.***
- ***Следите за наличием пустого места между группами.***

Встречаются приложения с сотней или даже тысячами измеренных или рассчитанных параметров, которые нужно отобразить пользователю. Один путь – уменьшать их размер и пытаться вместить все индикаторы на одной лицевой панели. Второй, предпочтительный – объединять их в логические легко читаемые группы, которые можно разместить на различных закладках, панелях или субпанелях с продуманной системой навигации по ним. Размер и количество объектов зависят от типа индикатора, его метки, выравнивания, разрешения монитора и других параметров. Одно из общих правил: на панели не должно быть больше 7 групп из 7 различных элементов.

Как показано на рис. 3.4, у ВП мониторинга мощности Power Monitor System (см. рис. 3.1) на лицевой панели 7 групп элементов. Это две группы для каждого измерительного стенда, выделенные элементами оформления (1, 4), кластеры индикации температуры (2, 5) и мощности (3, 6) и общие для двух стендов элементы

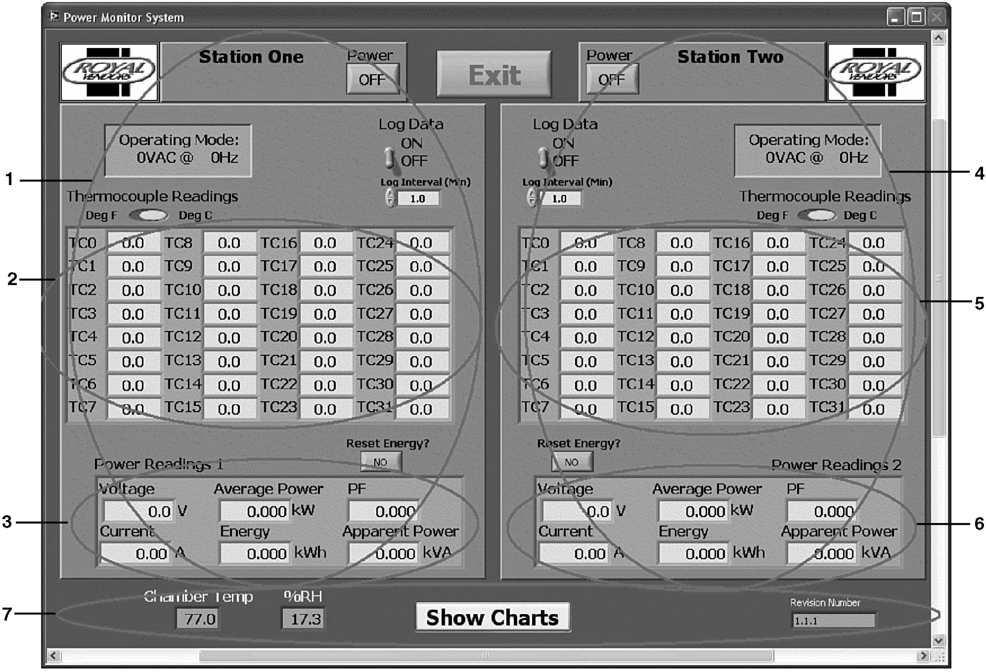


Рис. 3.4. Лицевая панель ВП Power Monitor System состоит из 7 групп объектов. Порядок и расположение 32 элементов в кластере индикации температуры продумано для удобства восприятия

(7). Кластер мощности состоит из 6 элементов, кластер температуры – из 32 тесно сгруппированных и пронумерованных. Но пользователю не требуется изучать все 32 индикатора, чтобы понять назначение каждого. Прочитав 3 или 4 метки, сразу становится понятно назначение и нумерация каждого. С точки зрения отображения эти 32 индикатора – это группа однотипных индикаторов. Но все-таки они разные, поэтому, строго говоря, в этом ВП не соблюдается ограничение числа элементов.

Плотность размещения элементов на ВП Power Monitor System достаточно высокая. Он состоит из 91 элемента на панели, оптимизированной для разрешения 1024×768. Плотность элементов легко уменьшить с помощью закладок (tab control). Пример – рис. 3.5, на котором каждому стенду соответствует своя закладка. Панель закладок занимает все окно, одновременно видны элементы только одной закладки. Количество индикаторов на экране уменьшилось до 48. Каждая закладка разделена на 5 логических групп, выделенных 3 элементами оформления и 2 кластерами индикации температуры и мощности. С пониженной плотностью расположения элементов данные лучше разделены и легче воспринимаются.

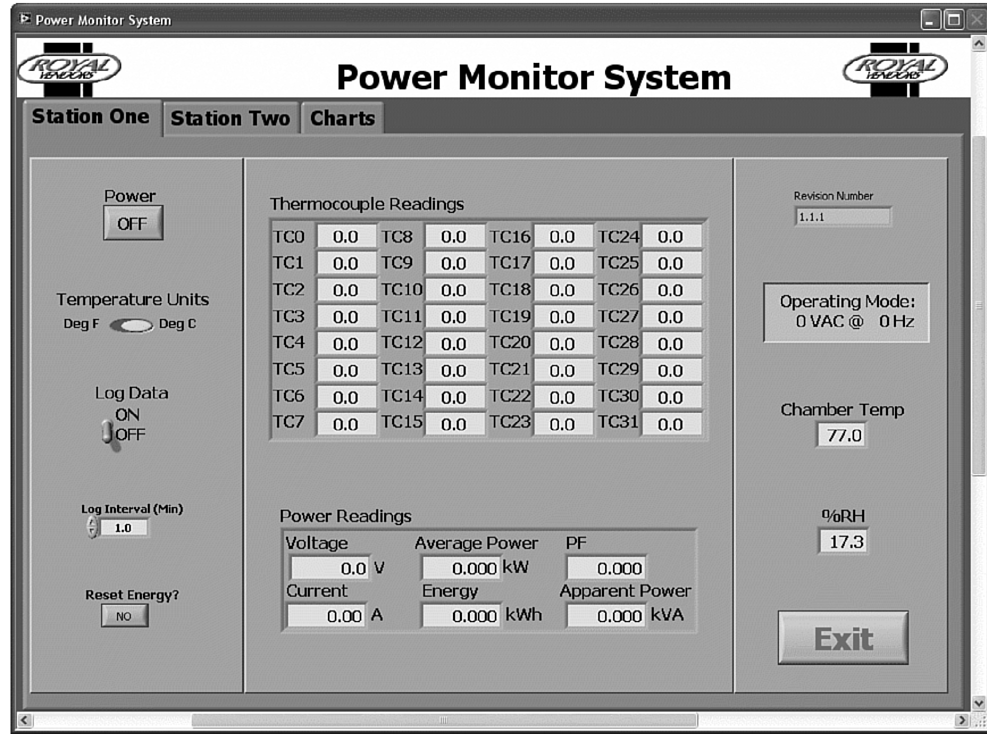


Рис. 3.5. Отдельные закладки панели соответствуют разным стендам, количество индикаторов уменьшено до 48 единиц на экран

При расположении элементов, показанном на рис. 3.5, пользователю видны данные только одного стенда. Они работают одновременно, поэтому приходится постоянно переключать закладки. Чтобы этого избежать, можно добавить еще одну закладку с отображением общего состояния системы с небольшим числом элементов отображения. Например, закладка **Overview** (Обзор) на рис. 3.6 отображает для каждого стенда максимальную и минимальную температуру, среднюю мощность, график температур и кнопку питания (**Power ON/OFF**). Для отображения большого числа данных полосовой график очень удобен. Такая структура позволяет лучше заметить изменение средней температуры каждого канала, а также закономерности и критические моменты, чем численные кластеры. Это иллюстрация влияния типа отображения на удобство восприятия данных. Итак, общая информация отображается на закладке **Overview**. Чтобы посмотреть более подробно, пользователь может переключиться на нужную закладку. Таким образом, снижена плотность расположения элементов и количество действия для навигации по приложению.

Правило 3.11 Избегайте перекрытия объектов

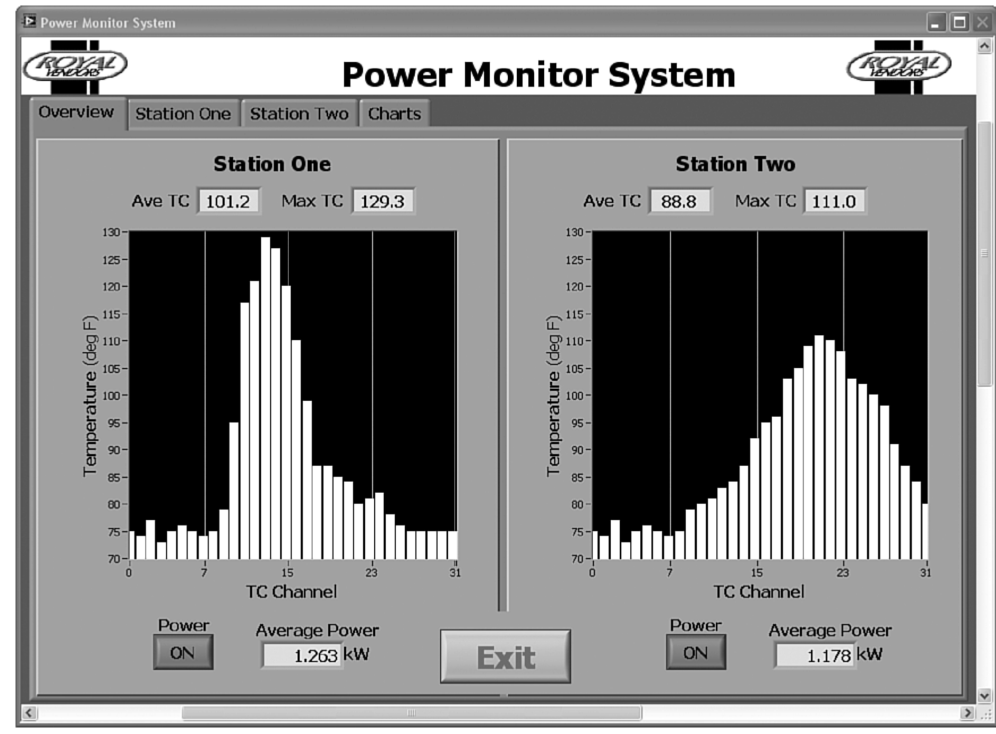


Рис. 3.6. На закладке обзора **Overview** приведена суммарная информация по двум стендам с помощью небольшого числа элементов

Если различные элементы лицевой панели перекрываются, то эффективность LabVIEW снижается. В частности, никогда не накладывайте индикатор на график. Каждый раз при изменении значения индикатора LabVIEW придется перерисовывать весь график, а это одна из наиболее трудоемких операций. Также у графиков LabVIEW есть много полезных функций комментирования зависимостей, поэтому накладывать дополнительные элементы обычно не требуется.

Иногда наложение элементов не влияет на производительность и допускается, например, если одновременно виден только один объект. Невидимые объекты LabVIEW не перерисовывает. Еще один пример – статический импорт изображения и наложение на него различных элементов. Например, это может быть схема промышленного процесса или тестируемого прибора с цифровыми индикаторами, которые отображают данные в данной точке в реальном времени. Если фон индикатора не прозрачный, то нижележащий объект не перерисовывается. Обновляются только данные индикатора. Пример такой схемы приведен на рис. 3.7.

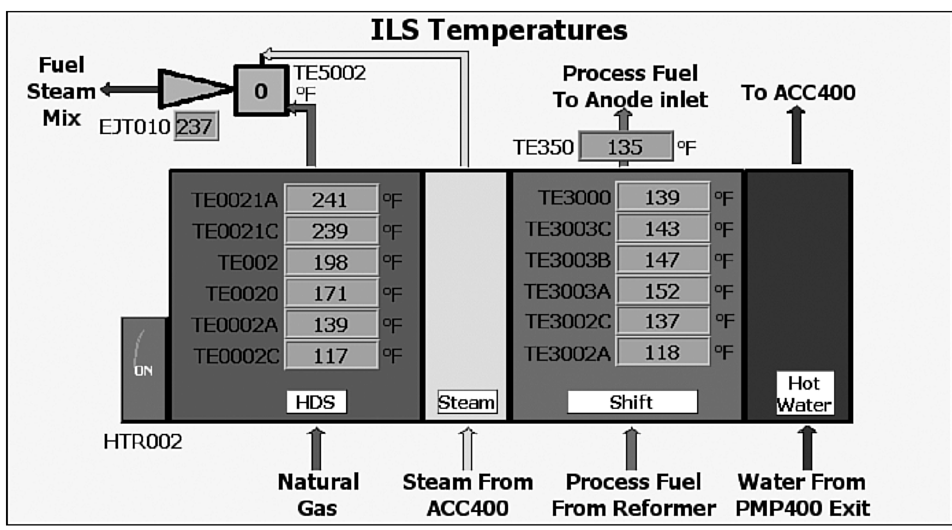


Рис. 3.7. Статическое изображение технологического процесса с наложенными цифровыми индикаторами работает достаточно эффективно, потому что фоновый рисунок не перерисовывается

Для пользовательского ВП необходимо задать несколько свойств окна лицевой панели. Большинство из них совпадают с параметрами окна приложения верхнего уровня (**Top level application window**) или диалога (**Dialog**), стандартные свойства которых применяются выбором соответствующего пункта. Один из наиболее важных пунктов – отключить кнопку прерывания исполнения (**Abort**). Во-первых, эта кнопка даст пользователю возможность прервать исполнение программы вне зависимости от состояния программы и без уведомления исполняющегося кода. Метода отследить нажатие кнопки и предпринять какие-либо действия не существует. Это очень опасное свойство. Большинство приложений LabVIEW управляют различными приборами, многие работают с сетевыми устройствами или целевыми платформами. При нажатии этой кнопки LabVIEW неожиданно останавливается, вне зависимости от состояния этих приборов и устройств. Внешние ресурсы могут оказаться в неопределенном состоянии, для сброса которого может понадобиться перезапуск. Также можно фрагментировать буферы памяти или испортить открытый файл. Поэтому всегда скрывайте кнопку **Abort** от глаз пользователя и обеспечьте грамотное завершение приложения.

Правило 3.12 Скрывайте панель инструментов

Еще лучше скрыть всю панель инструментов. Когда кнопка прерывания исполнения скрыта, остаются кнопки запуска (**Run**), непрерывного запуска (**Run Continuously**) и паузы (**Pause**). Эти инструменты удобны для обучения новичков. Если вы дочитали досюда, то, наверное, знаете, что такое цикл и средства отладки. Также у большинства ВП верхнего уровня включено свойство **Run when opened** (Запускать при открытии), поэтому панель инструментов LabVIEW обычно не нужна.

Правило 3.13 Во всех профессиональных приложениях есть логотип компании.

Отличный завершающий штрих – добавить логотип компании на лицевую панель ВП верхнего уровня. LabVIEW позволяет импортировать графические файлы всех стандартных форматов: выберите пункт меню **Edit** ⇒ **Import Picture to Clipboard** (Редактировать ⇒ Импортировать изображение в буфер обмена) и вставьте его на лицевую панель. Мой логотип обычно расположен на верхнем поле приложения.

3.1.3. Лицевая панель подприбора

В этом разделе приведены правила расположения элементов специально для лицевой панели ВПП, которая скрыта во время его работы.

Правило 3.14 Для лицевой панели ВПП, ее объектов и большей части текста пользуйтесь стандартным стилем

Панель подприбора открывают только разработчики, поэтому не стоит тратить время на ее разукрашивание. Наоборот, она должна оставаться строгой – так сразу становится понятно, что это именно подприбор. Когда разработчик открывает лицевую панель и видит стандартные цвета, шрифты, элементы, то сразу становится понятно, что это ВПП. Я предлагают потратить немного времени на то, чтобы сделать стиль ваших ВПП профессиональным и удобным. Пользуйтесь стандартным внешним видом для панелей, объектов и большей части текста. Это

обычно означает серый цвет, элементы с системной палитры, размер окна меньше полного экрана и системный шрифт размера 13.

Правило 3.15 Расположение элементов должно соответствовать разъемам соединительной панели

Правило 3.16 Размер панели должен быть удобным для расположения всех элементов

Правила расположения индикаторов и элементов управления в ВПП перекликаются с советами главы 5 «Иконка и контакты» по их подключению к разъемам. Располагайте управляющие элементы слева, индикаторы справа, ссылки каналов ввода-вывода (I/O refnum) сверху, кластеры ошибок снизу. Если элементов больше 8, то наиболее важные должны быть снаружи. Если элемент редко используется, ему самое место в центре по горизонтали (внутри). Примерно разместите элементы, подберите размер лицевой панели и воспользуйтесь средствами выравнивания и размещения элементов, чтобы сделать профессиональную лицевую панель. После этого назначьте все элементы разъемам.

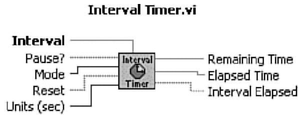
На рис. 3.8 показано два кандидата на роль служебного ВПП измерения интервалов времени Interval Timer VI. Наиболее важный параметр – **Interval** (Интервал времени), ему нужно назначить разъем соответствующей важности (**required**). Элементы **Pause?** (Приостановить?) и **Mode** (Режим) используются реже всего. Лицевая панель ВПП на рис. 3.8а стандартного размера, элементы расположены хаотично. Управляющий элемент **Interval** расположен в центре и соединен со средним разъемом. Центр панели подходит для размещения важных элементов пользовательского ВП, но не подприбора. Лицевая панель на рис. 3.8б гораздо аккуратнее, ее размер соответствует количеству элементов. Наиболее важные параметры находятся с краю, самый важный – интервал слева сверху, дополнительные – в центре. Для обработки ошибок добавлены соответствующие кластеры. Назначение разъемов соответствует расположению элементов. Обработка ошибок обсуждается в главе 7 «Обработка ошибок», а в главе 5 приведены советы по назначению разъемов.

3.2. Текст

Внешний вид текста оказывает существенное влияние на удобство работы, в том числе и в приложениях LabVIEW, которые мы учимся создавать. В этом разделе приведены советы, касающиеся текста лицевой панели. Начнем с общих правил, подходящих как для пользовательских ВП, так и подприборов.

3.2.1. Общие правила

Сколько раз вы полностью читали лицензионное соглашение при установке программы? Или вы автоматически нажимаете кнопку, подтверждающую согласие?



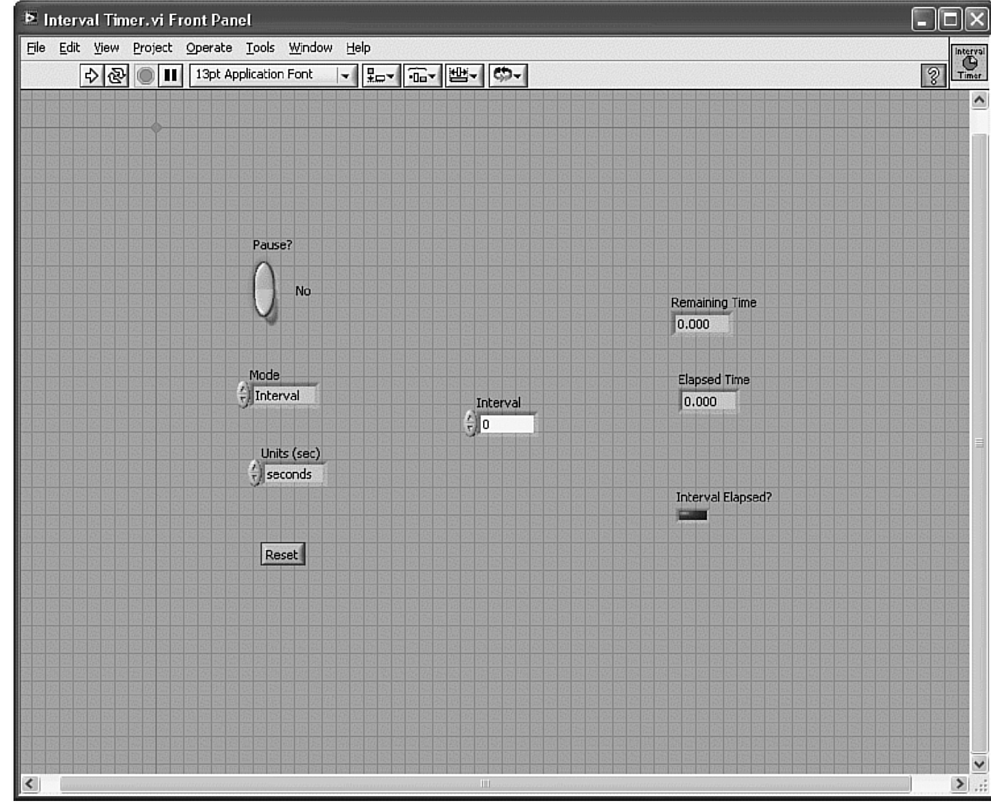


*Рис. 3.8а. Элементы лицевой панели ВП **Interval Timer VI** расположены хаотически, размер панели неподходящий*

Приступая к новому программному обеспечению, вы начинаете с чтения документации, справочной информации или сразу приступаете к работе, разбираясь с кнопками и командами? Не вызывал ли у вас раздражения длинный текст в окне программы?

Правило 3.17 Количество текста на лицевой панели должно быть минимальным

Понятные лицевые панели состоят из графических элементов, а не текстовых. Пользователи и разработчики не любят читать длинные описания, они предпочи-

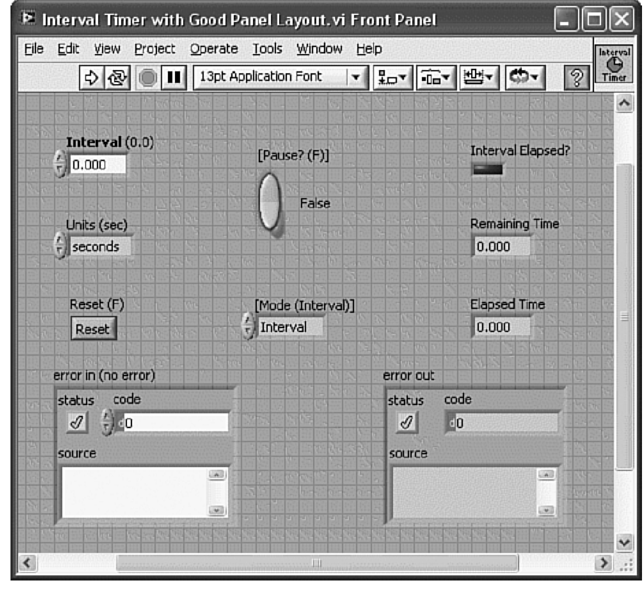


Рис. 3.8б. Элементы аккуратно расположены, размещены в порядке приоритета, который подчеркивается шрифтом. Размер панели удобен для работы

тают кнопки, элементы управления, объекты, иконки и картинки, при взгляде на которые становится понятно их назначение. Не следует размещать на кнопках или их метках предложений, длинных фраз и параграфов с описанием. Они либо затормозят понимание программы, либо пользователи их просто проигнорируют. Чем больше текста на лицевой панели, тем сложнее и запутаннее выглядят программы. Для подробных описаний есть гораздо более подходящие места.

Иногда при удаленной работе над приложением разработчики переносят рекомендации на лицевую панель ВП, чтобы пользователь мог разобраться с программой. Однако этот текст в лучшем случае окажется полезным только новичку. Через некоторое время текст будет полностью игнорироваться и занимать лишнее место. Можно задаться вопросом, каково соотношение опытных пользователей и новичков, разбирающихся с приложением? Если только программа не предназначена для новичков или редкого пользования, например это может быть программа обучения или установки, то текст лучше убрать.

Вместо длинных описаний на лицевой панели можно воспользоваться следующими средствами:

- создавайте интуитивно понятный пользовательский интерфейс, которому вообще не требуются текстовые пояснения;
- добавьте кнопку и пункт меню **Help** (Справка), который по запросу открывает справочную информацию: документ с описанием, справочное окно LabVIEW или диалоговое окно с описанием;
- напишите руководство к ПО в виде текстового документа или файла форматированной справки.

Правило 3.18 Удаляйте временные замечания сразу после выполнения их инструкций

Еще одна ситуация, после которой остаются лишние текстовые описания, – это привычка разработчиков оставлять заметки и указания на лицевой панели шаблона или ВП, предназначенного для других разработчиков. Например, в шаблонах драйверов приборов есть свободный текст с описанием требуемых изменений каждого ВП. Всегда удаляйте такие описания сразу после выполнения инструкций. В противном случае приложение кажется незавершенным, очень неприятно, открыв готовое приложение, обнаружить там инструкции по его редактированию.

Правило 3.19 Не злоупотребляйте форматированием текста

Чтобы различить различные типы данных или событий, в большинстве приложений потребуется не один стиль текста. Например, в промышленных ВП наиболее важные данные отображаются полужирным шрифтом большого размера. Также контрастный текст может пригодиться для заголовков, важных сообщений: ошибок или предупреждений. Но если на лицевой панели слишком много стилей, она станет слишком яркой и даже более запутанной. Для большинства приложений хватит 3 стилей: стандартного для большинства элементов, большего размера или полужирного для выделения важных данных и контрастного стиля для заголовка и других специальных элементов. Очень важно выбрать эти стили и пользоваться ими на протяжении всего приложения.

LabVIEW поддерживает самые разные стили, но не все дают один результат. Шрифты Application (Приложения), System (Системный) , Dialog (Диалог) – это не сами шрифты, а ссылки на шрифты используемой операционной системы. Шрифт Приложение – стандартный, он предназначен для большинства элементов. Шрифт Диалог используется в сообщениях и элементах диалоговых окон. Системный шрифт предназначен для заголовков окон и важных сообщений, привлекающих внимание пользователя. Благодаря этим шрифтам внешний вид приложения похож на стандартное окно системы. Но шрифт, который будет использоваться на самом деле, зависит от операционной системы и настроек пользователя. В Windows используется только один шрифт: в Windows XP – Tahoma, в Windows Vista – Segoe UI. Преимущества отсутствия жесткого задания шрифта – лучшая

переносимость и совместимость программ для разных операционных систем. Благодаря этим средствам, если вы переносите программу на другую платформу или обновляете свою ОС, внешний вид приложения все равно будет соответствовать новой системе. Мы не знаем, какие шрифты будут использоваться в следующих версиях Windows, но отображение шрифтов позволит изменить вид приложений без каких-либо действий.

В качестве альтернативы можно пользоваться шрифтами, которые поддерживаются большинством операционных систем: Arial, Times New Roman и Tahoma. Если нужно жестко задать внешний вид текста, выберите стандартный шрифт, обычно они выглядят одинаково на всех платформах. Иногда вам придется выбирать: придерживаться стиля ОС или сохранять неизменным внешний вид окна.

Для достижения особых эффектов вам может потребоваться нестандартный стиль. Например, для обозначения различных величин используются греческие буквы, например из шрифта Symbol. Чтобы сохранить ширину строковых элементов, пригодится шрифт с одинаковым размером символов: monospace Courier New или другие. Учтите, что на разных платформах эти шрифты могут выглядеть поразному. Иногда придется произвести определенные операции, чтобы восстановить расположение меток. Часто используемая греческая буква µ на некоторых машинах может превратиться в m. Некоторые специальные шрифты вообще преобразуются по неизвестным правилам, которые могут различаться даже для лицевой панели и блок-диаграммы одного ВП. Поэтому для многоплатформенных приложений специальные шрифты не рекомендуются. В крайнем случае, можно создать нужную метку и вставить ее как рисунок.

Правило 3.20 Выберите один шрифт и подчеркивайте разный стиль его размером, толщиной и цветом

Шрифты приложения, диалога и системный используются в разных ситуациях, однако, в наиболее популярных операционных системах Windows XP и Vista для них используется один шрифт. Я также советую пользоваться одним шрифтом для лучшей совместимости: либо системным, либо универсальным, например из группы **sans serif**: **Arial** или аналогичным. Чтобы получить разный стиль, изменяйте размер и цвет текста, это спасет вас от путаницы при изменении шрифта.

Как показано на рис. 3.9а, на ВП Power Monitor System используется 8 различных стилей, а именно шрифт Application: полужирный размера 42 – для заголовка **Power Monitor System**, полужирный размера 24 – для меток закладок, полужирный размера 20 – для всех меток индикаторов и данных, полужирный размера 13 – для меток **Log Interval** (Интервал регистрации) и текста кнопки **Reset Energy**, обычный размера 13 – для метки **Revision Number** (Номер версии) и обычный размера 37 красного цвета – для кнопки **Exit**. Различные стили подчеркивают приоритет разных элементов, но лучше для этого пользоваться расположением элементов. Более приятная лицевая панель показана на рис. 3.9б, на ней используется только 3 стиля. Полужирный шрифт размера 20 используется для всех ключевых меток,

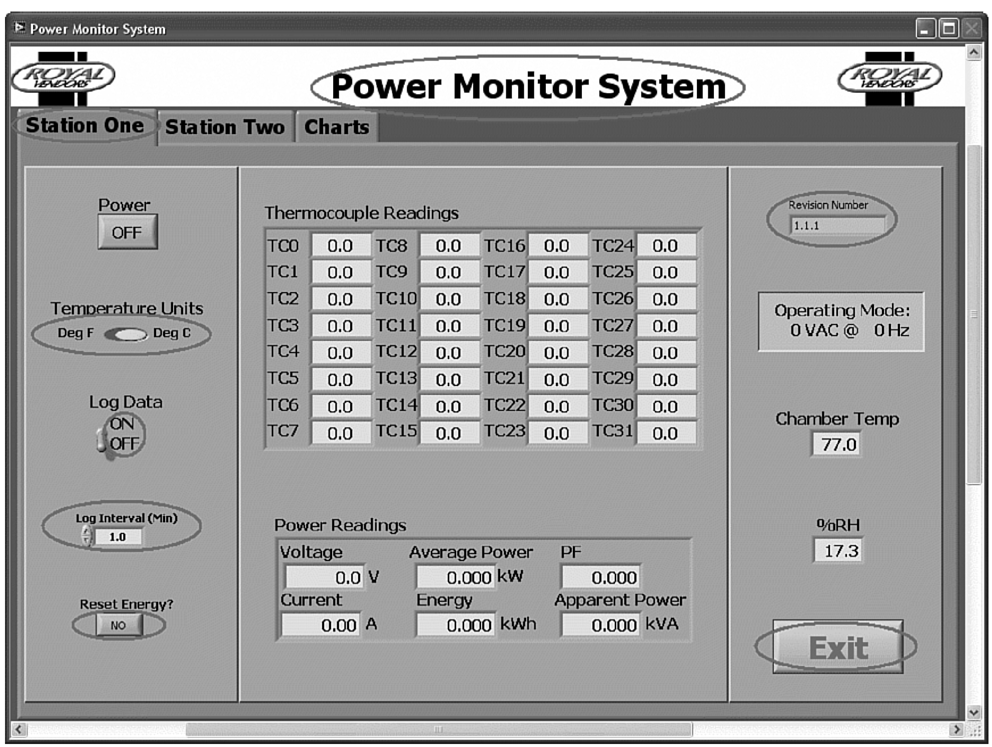


Рис. 3.9а. На лицевой панели ВП Power Monitor System используется 8 стилей текста, различающихся размером и цветом

данных и текста кнопок, обычный шрифт размера 20 – для дополнительных меток, включая размерность и положения переключателя, шрифт заголовка не изменился: 42 размера, полужирный. Лицевая панель стала понятнее и аккуратнее.

Также следите за использованием заглавных букв. Например, если у вас есть 12 индикаторов одной важности с похожими данными, с большой буквы должны быть написаны все, но не несколько. Казалось бы, простое правило, но его стоит подчеркнуть, потому что нарушается оно достаточно часто. Наилучший способ использовать подходящие стили текста – разработать их до программирования, лучше, если они будут одинаковыми во всех ваших работах. Это позволит вам сохранить время при оформлении нового приложения.

3.2.2. Метки элементов

Правила этого раздела описывают текст у меток элементов.

Правило 3.21 Пользуйтесь интуитивно понятными метками и внедренным текстом

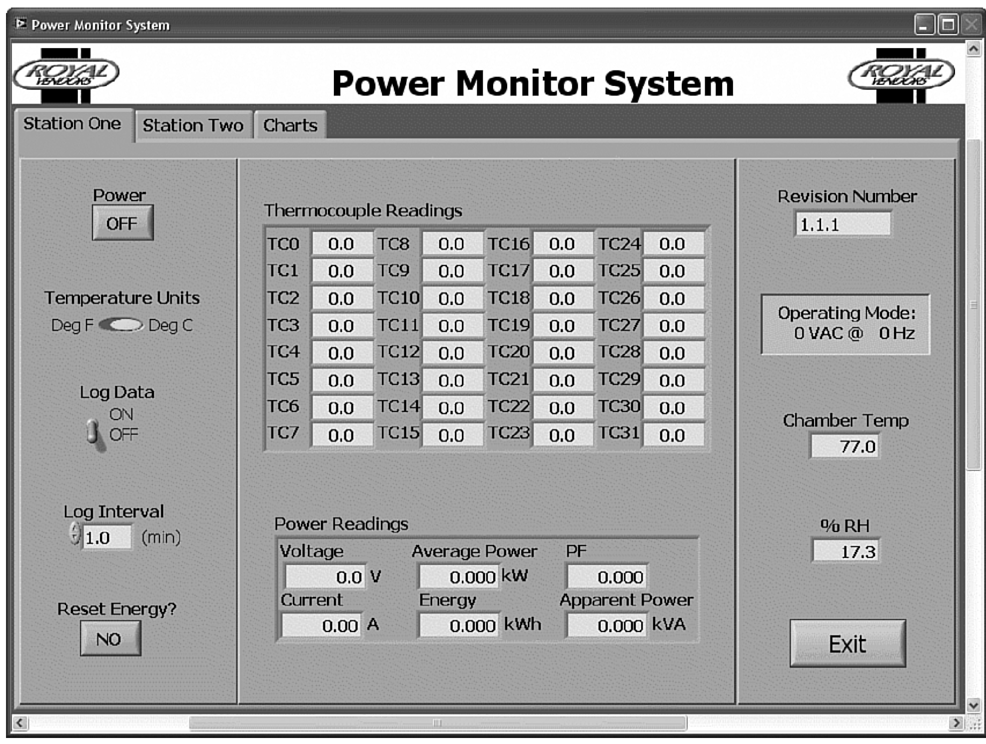


Рис. 3.9б. Лицевая панель ВП Power Monitor System переработана, на ней используется только 3 стиля. Программа стала понятнее и аккуратнее

Это правило – аналог совета 3.17, но для элементов управления. Метки индикаторов и элементов управления, а также их внедренный текст должны быть интуитивно понятными. Поставьте себе задачу описать действие каждой кнопки, индикатора и элемента управления одним или двумя словами. Пример показан на рис. 3.10. Эти два меню функционально одинаковые. В первом случае кластер с меткой **COLLECT MENU** состоит из кнопок с тремя строками внедренного текста: заглавными буквами – имя кнопки, потом описание действия и горячая клавиша. На рис. 3.10б те же самые кнопки обозначены максимально кратко: одним понятным словом, причем оно лучше описывает назначение кнопки, чем целая фраза. Название кластера перенесено в заголовок окна, а строка меню скрыта, так количество текста на экране стало минимальным. В результате, стиль ВП на рис. 3.10б лучше, чем ВП на рис. 3.10а.

Что делать, если пользователю нужна информация о назначении и действии каждой кнопки? Лучше всего ее разместить во всплывающей подсказке и описании элемента. Когда пользователь наводит мышку на элемент, описание отображается в окне контекстной справки. Оно может быть любого нужного вам размера и позволяет описать элемент гораздо подробнее, чем встроенный текст. Если окно контекстной помощи мешает, пользователь может его закрыть.

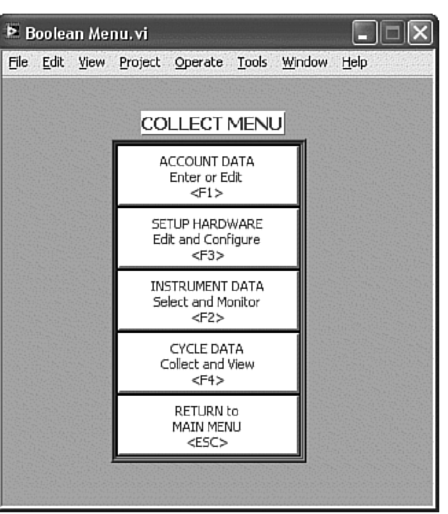


Рис. 3.10а. В меню элементов слишком много лишнего текста

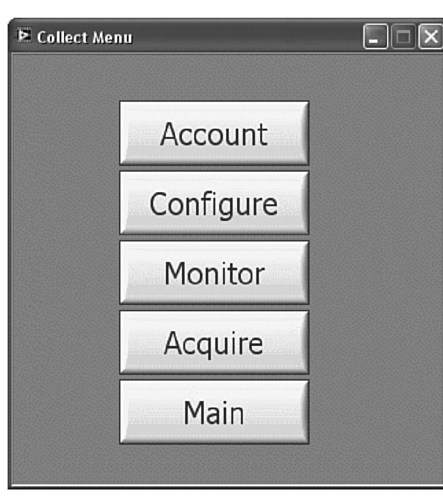
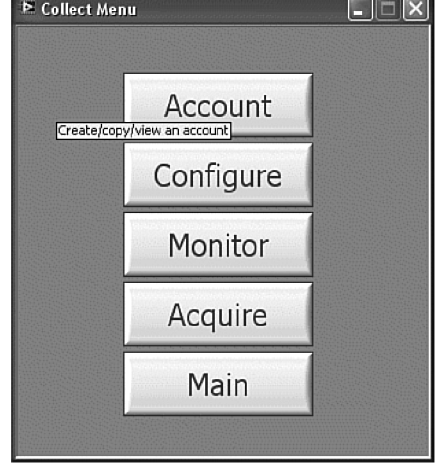


Рис. 3.10б. Текст кнопок максимально краткий, строка меню скрыта, название перенесено в заголовок окна



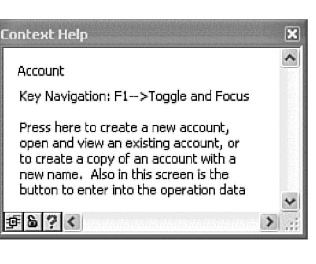


Рис. 3.10в. Полное описание элементов и всплывающая строка для кнопок меню

Всплывающая подсказка – это строка текста, которая появляется на экране, когда пользователь наводит курсор на элемент, через несколько секунд подсказка исчезает. Она идеально подходит для одного предложения с кратким описанием элемента. На рис. 3.10в приведена всплывающая подсказка и описание в окне контекстной справки для кнопки **Account**. Таким образом, меню кратких логических элементов эффективнее описывает действие элементов, чем текстовое меню на рис. 3.10а.

Сделаем еще одно замечание, касающееся меток элементов. В большинстве случаев лучше оставлять их фон прозрачным. Это свойство используется по умолчанию: **Use transparent name labels** в меню **Tools** ⇒ **Options** ⇒ **Front Panel** (Инструменты ⇒ Опции ⇒ Лицевая панель). В более ранних версиях LabVIEW метки были выделены, приходилось устанавливать фон каждой. Эти времена остались в прошлом, если только у вас установлено упомянутое выше свойство.

3.2.3. Текст в подприборах

В этом разделе приведены правила текста ВПП.

Правило 3.22 Используйте стандартный шрифт (Application размера 13, черный) для большинства элементов ВПП

Как мы уже говорили, украшать ВПП даже вредно, поэтому для оформления ВПП чаще всего используется стандартный шрифт черного цвета размера 13 на серых лицевых панелях со стандартными элементами.

Правило 3.23 В конце меток элементов указывайте значение по умолчанию

Правило 3.24 В метках элементов используйте метки с полужирным шрифтом и пояснения стандартным шрифтом

Если возможно, в конце меток указывайте значение по умолчанию. Они полезны при работе с ВПП: как с лицевой панелью, так и благодаря окну контекстной помощи. Коммерческие ВПП, например, драйверы и библиотеки разработчика требуют дополнительных усилий. В метках элементов используется полужирный шрифт, а значения по умолчанию задаются в скобках стандартным шрифтом. Это соглашение принято в руководстве к разработке драйверов (Instrument Driver Guidelines)[2] от NI. Панели ВПП выглядят аккуратнее. На рис. 3.11 показан ВПП Interval Timer, соблюдающий эту конвенцию.

3.2.4. Текст промышленных ВП

Удобство чтения – очень важное свойство промышленных ВП. Следующие правила помогут улучшить стиль лицевой панели специально для многопользовательских промышленных приложений.

Правило 3.25 Увеличьте контраст между шрифтом и фоном

Правило 3.26 Увеличьте размер шрифта кнопок и важных данных

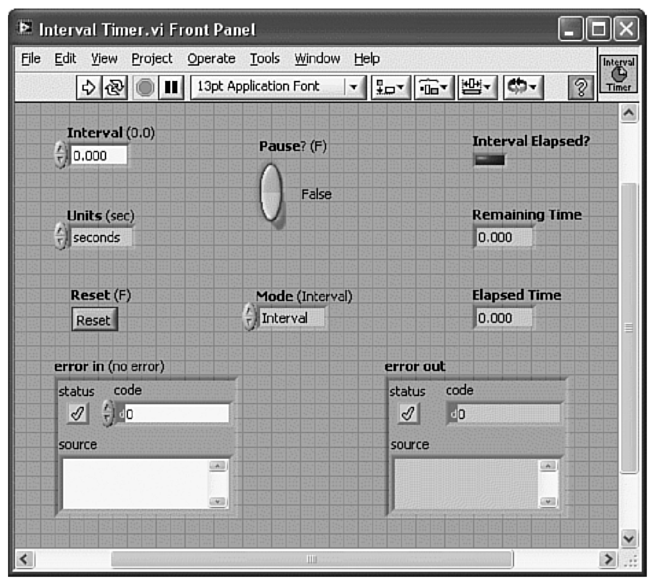


Рис. 3.11. Стиль меток в ВПП Interval Timer был изменен для коммерческого приложения. Названия элементов в метках обозначены полужирным шрифтом, а значения по умолчанию – стандартным

Важные данные и объекты в промышленных приложениях должны быть четко видны с нескольких метров. Увеличьте размер объектов и шрифта, а также контраст между текстом и фоном: при использовании черного текста сделайте фон более светлым. Размер шрифта можно увеличить с помощью диалогового окна Font Style (Стиль шрифта) или комбинацией клавиш **Ctrl++**. Каждое нажатие увеличивает размер шрифта на 1. На темном фоне пользуйтесь белым или другим ярким цветом. У людей могут быть проблемы с различением оттенков, поэтому контраст важнее самого цвета.

Правило 3.27 Увеличьте промежутки между метками и объектами многоплатформенных приложений

Как уже обсуждалось выше, внешний вид текста зависит от платформы и операционной системы. Настройки разрешения, видеоадаптера, драйвер, параметры монитора и установки пользователя в ОС могут влиять на большинство шрифтов. Если ваше приложение предназначено для переноса на другие платформы, предусмотрите дополнительные промежутки, чтобы текст не перекрывался.

3.3. Цвет

Многие экспериментировали с цветовой гаммой лицевых панелей, но не всегда удачно. Также часто встречаются черно-белые приложения. Выбор цвета – это искусство. Некоторые разработчики успешно занимаются программированием, но не оформлением своих результатов. Более того, примерно 10% людей страдают нарушениями зрения, которые мешают им различать основные цветовые оттенки[3]. Даже для людей с нормальным зрением цвет – самое неопределенное свойство лицевой панели. Одни могут прийти в восторг, другие плеваться от отвращения. Но все-таки общие правила существуют.

Правило 3.28 Не злоупотребляйте цветами
• Выберите три основных цвета: яркий, приглушенный и оттенок серого

У ВП, которые вызвали наибольшее отвращение, была одна общая черта: слишком много ярких оттенков. Выберите 3, максимум 4, основных цвета и на протяжении всего приложения пользуйтесь только ими. Очень простой совет: один из этих цветов должен быть ярким, другой приглушенным, третий – оттенком серого. С помощью инструмента выбора цветов в LabVIEW это делается очень просто: 3 указанных категории уже вынесены в полоски – рис. 3.12. Оттенки серого и приглушенные цвета предназначены для больших объектов и фона лицевой панели. Они находятся в двух верхних строчках палитры. Полоса ярких цветов находится ниже. Расчетливо воспользуйтесь одним или двумя для привлечения внимания к небольшим объектам или анимации.

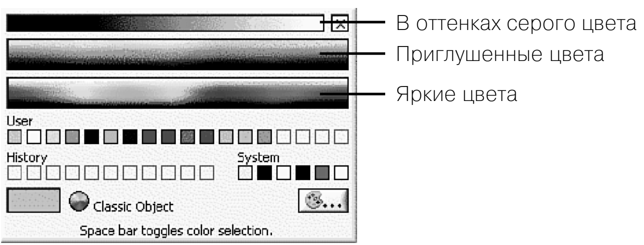


Рис. 3.12. Инструмент выбора цвета LabVIEW с полосками приглушенных, ярких цветов и оттенков серого

Правило 3.29 Разработайте цветовую схему приложения
• Логически связанные элементы окрашивайте одним цветом
• Придерживайтесь одного цветового набора на протяжении всего приложения

В разделе 3.3. «Расположение» мы обсудили, как подчеркнуть логические отношения между элементами с помощью групп и промежутков. Цвет помогает усилить визуальное восприятие соотношений между объектами. Ваша цветовая схема должна объединять логически связанные элементы между собой, причем для разных окон приложения схема должна быть одинаковой. Легенда цвета поможет формально описать цветовую схему.

Правило 3.30 Следуйте общепринятым соотношениям при использовании желтого, зеленого и красного цветов

У красного, желтого и зеленого цветов в промышленных приложениях есть определенное назначение. Зеленый соответствует нормальным условиям работы, желтый – предупреждение, красный – ошибка. Для промышленного ВП соблюдение этих требований обязательно.

Правило 3.31 Оставьте лицевую панель и объекты ВПП серыми

Панели ВПП разукрашивать не нужно. Как мы уже говорили, стандартный цвет помогает подчеркнуть их роль. Если вы работаете над коммерческим продуктом, например драйвером или библиотекой приборов, скромная цветовая схема поможет подчеркнуть, что ВПП относятся к данной группе. В этом случае, по моему мнению, стоит использовать один цвет для ограниченного количества выделений. Но функциональные и ясные иконки с этой точки зрения гораздо полезнее хотя бы потому, что они находятся на блок-диаграмме пользователя. Поэтому у меня на иконки уходит гораздо больше времени, чем на цветовое оформление лицевой панели.

Правило 3.32 Оставьте цветовую схему простой, не тратьте слишком много времени

Некоторые разработчики тратят слишком много времени на эксперименты с цветовой гаммой в поисках оптимальной для ВП. Самое важное в работе – расставить приоритеты выполнения: работу приложения, интерфейс и другие задачи, соответственно планируя время. Для коммерческих приложений профессиональный интерфейс очень важен, и приоритет у этой работы достаточно высок. Однако основное назначение приложений LabVIEW – тестирование, измерение и управление. Убедитесь, что ваше приложение работает как часы, а потом можете потратить оставшееся время на доведение интерфейса до идеала. Обычно я анализирую, выполняю и тестирую требования в порядке их важности. К тому же у меня есть несколько заготовленных цветовых схем, которые мне понравились раньше, а на эксперименты можно потратить время и после решения критических задач.

Ниже, в разделе 3.5 «Примеры», приведено несколько лицевых панелей пользовательских ВП и подприборов. Книга черно-белая, и всех особенностей цветового оформления вы не увидите. Цветные рисунки можно скачать с сайта издателя www.prenhallprofessional.com/title/0131458353. Цветную электронную копию книги можно купить на www.prenhallprofessional.com/title/0132414813.

3.4. Навигация по приложению

Теорема 3.1 *Надежность определяется разработчиком.*

Навигация по приложению – это последовательность действий с помощью элементов управления и команд меню для перехода между различными частями программы. Если приложением будете пользоваться не только вы, надежность и простота использования – это одни из ключевых свойств программы. Более того, важно понимать, что надежность приложения определяется целиком разработчиком. Никогда не предполагайте, что пользователь будет выполнять действия в задуманном вами порядке, и не допускайте последовательность, которая может привести к сбою. Например, на лицевой панели может быть кнопка возврата приборов в начальное состояние. Что случится, если пользователь нажмет ее во время проведения измерений? Прибор может повиснуть, нужно скрыть кнопку, когда это опасно. У теоремы есть два доказательства: одно теоретическое, другое практическое. Теоретическое: программист отвечает за разработку надежного приложения, это на нем лежит ответственность сделать приложение, которое будет работать без сбоев. В LabVIEW есть все средства для создания надежных интерфейсов.

3.4.1. Элементы управления

Правило 3.33 Ограничьте количество активных элементов управления

Общее правило следующее: активными должны быть элементы управления, которые не вызовут нежелательных действий. Рассмотрите различные режимы работы приложения: настройка, сбор данных, анализ и отображение. Многие из элементов управления, необходимые в первом режиме, в остальных не используются. Попробуйте использовать различные панели или закладки для отображения только тех элементов, которые нужны в данном режиме. Изменение активной закладки должно происходить программно, выбор «опасных» закладок нужно запретить. Для программного управления видимыми или активными в данный момент элементами пользуйтесь узлами свойств для изменения свойств Visible и Disabled. Таким образом, после запуска сбора данных закладка настройки должна быть отключена. Пример такого ВП приведен в разделе 3.5.2. «Служебный диалог».

Также у интерфейса могут быть параметры, соответствующие определенной конфигурации. Например, рассмотрим задачу измерения тока, температуры, напряжения или сопротивления с помощью цифрового осциллографа. В настройках можно выбрать схему измерений: с 2 или 4 проводами. Этот управляющий элемент относится только к измерениям сопротивления, в других случаях он должен быть отключен.

В то же самое время избегайте лишнего изменения положения, скрытия или вывода на экран элементов, в противном случае ваше приложение будет напоминать калейдоскоп. С программой приятно работать, если все элементы расположены в логичном, отведенном им месте. Я предпочитаю не скрывать элементы из виду, а отключать их, изменяя это свойство (disabled) со значения 0 (включено) на значение 2 (отключено и затенено). В этом случае оператор всегда знает, где находятся параметры, и может легко их найти. В рассмотренном примере управляющий элемент **Resistance Type** (Тип сопротивления) всегда остается на экране, но может быть затенен в зависимости от типа измерений.

Правило 3.34 Ограничьте диапазон изменения всех управляющих элементов

Правило 3.35 Пользуйтесь свойством Data Range (Диапазон данных) для численных элементов управления

Правило 3.36 Пользуйтесь списками выбора, а не строками там, где это возможно

Еще один важный вопрос: что случится, если пользователь введет любое значение в строковый или числовой элемент управления? Есть ли бессмысленные значения, возможность ошибки или другие проблемы? С численными элементами управления эту проблему решить достаточно просто. Просто задайте диапазон данных (**Data Range**) в свойствах элемента или программно с помощью узлов свойств. Для строкового элемента вам, возможно, придется создать ВП поиска недопустимых символов или их комбинаций. В LabVIEW есть несколько удобных функций для решения этой задачи: Match regular expression (Найти регулярное выражение) и Search and replace string (Поиск и замена строки). Если можно вводить только несколько строк, лучше воспользоваться текстовым списком (text ring или list box), числовым списком (enum) или комбинированным окном (combo box). Введите допустимые строки, пользователю придется выбрать один вариант из этого набора. Первые три из указанных элементов каждой строке ставят в соответствие целое число. Комбинированное окно преобразует строку в другую строку. В соответствии с рекомендациями NI (*Instrument Driver Guidelines*[2]), для драйверов приборов лучше всего подходят списки (text ring). Я предпочитаю пользоваться списками enum, потому что у них есть очень полезное свойство: при соединении их с терминалом выбора структуры варианта (Case structure), все строки списка автоматически назначаются кадрам структуры. Эта конструкция называется **enumerated Case structure** (Структура варианта со списком). Более подробна она обсуждается в главе 4 «Блок-диаграмма» и главе 9 «Документация».

При использовании списков различного вида выбранную строку можно получить из массива строк элемента (свойство Strings[]), как показано на рис. 3.13.

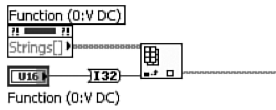


Рис. 3.13. Чтобы получить выбранную строку списка, выделите из массива строк элемент, соответствующий номеру выбранной строки

Правило 3.37 Задайте порядок элементов (tabbing order) и пользуйтесь свойством Key focus (Фокус ввода)

Если вы хотите помочь пользователю ориентироваться в элементах управления, пользуйтесь такими инструментами LabVIEW, как tabbing order (порядок элементов) и Key focus (Фокус ввода). Порядок элементов задается с помощью меню **Edit** ⇒ **Set Tabbing Order** (Редактировать ⇒ Задать порядок элементов), фокус ввода – это свойство элемента, которое показывает, выделен ли элемент и позволяет пользователю вводить данные, не пользуясь мышью для выбора элемента. Например, фокусом ввода пользуются при открытии панели, чтобы указать первый элемент для ввода данных, после этого пользователь может переходить между элементами с помощью клавиши **Tab**. Если некоторые элементы скрыты от глаз пользователя за границами панели (например, кластеры ошибок), не забудьте исключить их из списка элементов. Для этого откройте окно **Properties** (Свойства элемента) с помощью контекстного меню, перейдите на закладку **Key Navigation** (Навигация) и выберите **Skip this control when tabbing** (Исключать элемент из выбора). Наконец, назначьте горячие клавиши, чтобы пользователь мог выполнить основные действия без помощи мыши. Клавиши назначаются на закладке навигации в свойствах элемента.

Правило 3.38 Настройте меню ВП верхнего уровня

Правило 3.39 На ВП должна быть справочная кнопка или пункт меню

Стандартное меню LabVIEW для пользователя практически бесполезно. Большинство ее команд предназначено разработчику для редактирования исходного кода в среде LabVIEW. Эти пункты только путают пользователей, незнакомых со средой. Замените меню ВП верхнего уровня на более соответствующее решаемой задаче. Сотрите все относящиеся к программированию пункты и оставьте только те, которые нужны пользователю. Как минимум должна быть команда **Help** (Справка), в этот пункт можно добавить и пункт **Show Context Help** (Показать окно контекстной помощи) и открытие описания. По команде **Show Context Help** откроется окно с описанием элементов лицевой панели под курсором. Удобство файлов компилированной справки (CHM файлов) мы обсудили выше, в разделе 3.2.2 «Метки элементов». Настроить пункты меню можно с помощью команды **Edit** ⇒ **Run-Time Menu** (Редактировать ⇒ Меню во время работы).

В диалоговых ВП меню не используются. В случае необходимости, добавьте кнопки нужных пунктов, в частности справки, чтобы открыть контекстную помощь или документацию. Также для определенных элементов управления можно создать контекстные меню: **Advanced** ⇒ **Run-Time Shortcut Menu** ⇒ **Edit** (Дополнительно ⇒ Контекстные меню во время работы ⇒ Редактировать).

3.4.2. Ваш стиль

Вы можете не соглашаться со всеми правилами, которые я предлагаю, и это здорово. Стиль пользовательских элементов – понятие субъективное. Поэтому самое важное правило – делать все последовательно.

Правило 3.40 Будьте последовательным!
• Продумайте расположение и внешний вид элементов
• Сохраняйте элементы с измененным видом как строгие определения типа

Если вы разработали набор шрифтов, цветов и типов элементов управления, пользуйтесь ими во всем приложении или даже в разных приложениях. Пользователям станет легче ориентироваться в ваших работах: поняв принцип работы одного приложения, станет понятно, где расположены специфические элементы управления, как управлять интерфейсом. Одинаковый стиль приложений убедит пользователя в том, что ведут себя они также одинаково. Также важно придерживаться одного стиля в рамках одной организации. Это, в частности, означает, что вам и вашим коллегам нужно разработать этот стиль вместе. Надеемся, что эта книга поможет вам. В наборе ВП, созданных для одной общей задачи, пользуйтесь одинаковыми элементами в одном месте каждого ВП. Например, если вы делаете обучающую программу из нескольких шагов с кнопками навигации «Вперед», «Назад» и «Выход», эти кнопки должны быть одинаковыми и расположенными в одной и той же части окна.

Для настройки внешнего вида и свойств элементов лицевой панели предназначен специальный control editor (редактор элементов) LabVIEW, он запускается из контекстного меню: команды **Advanced** ⇒ **Customize** (Дополнительно ⇒ Редактировать). Сохраненный элемент управления или определение типа позволит вам пользоваться элементами в разных приложениях и поддерживать свой стиль элементов. Просто добавьте определение типа в проект и перетаскивайте его на лицевые панели приборов. Элемент динамически загружается из файла и содержит все его настройки.

Статус определения типа (**Type Definition Status**) выбирается в списке при редактировании элемента. Это может быть **Control** (Элемент), **Type Def** (Опреде-

ление типа) и **Strict Type Def** (Строгое определение типа). Элемент позволяет просто сохранить в файле выбранный индикатор или элемент управления со всеми настройками. Разместив его на лицевой панели нового ВП, вы можете дополнительно редактировать его, причем изменения не затронут исходный файл. Каждую копию определения типа также можно настраивать, не меняя тип данных. Все параметры строгого определения типа заданы в его файле, они одинаковы для всех копий.

В моих проектах встречается много копий одного элемента, который приходится редактировать. В большинстве случае нужно, чтобы изменения затронули все копии элемента, поэтому я предпочитаю строгие определения типа. Мне не требуется искать каждую копию, если нужно внести изменения: редактируется один файл элемента, все его копии обновляются автоматически. Также строгие определения типа помогают поддерживать одинаковым стиль всех ВП проекта. Более подробно определения типа обсуждаются в главе 6 «Структуры данных».

3.5. Примеры

В этом разделе приведены примеры пользовательских ВП и подприборов. Цветные рисунки можно скачать с сайта издателя www.prenhallprofessional.com/title/0131458353. Цветную электронную копию книги можно купить на www.prenhallprofessional.com/title/0132414813.

3.5.1. ВПП из блок-диаграммы

Один из методов создания ВПП – выделить участок блок-диаграммы и выбрать пункт меню **Edit ⇒ Create SubVI** (Редактировать ⇒ Создать ВПП). Как по волшебству, выделенный фрагмент заменится на ВПП. Это очень удобно, но одновременно это самый простой способ разрушения стиля. Лицевую панель, блок-диаграмму, иконку и соединительную панель придется долго редактировать. Пример нарушения множества правил – рис. 3.14а. Управляющие элементы расположены по горизонтали и не соответствуют разъемам (Правило 3.15). Стандартные элементы, такие как **task ID** (ИН задачи) и кластеры ошибок, соединены с нестандартными разъемами (Правило 3.40), размер панели не соответствует количеству элементов (Правило 3.16). Более того, назначение элементов не понятно (Правило 3.21). У всех трех входных данных в имени содержится **out** (вывод), для различения одинаковых элементов используются номера. Результат редактирования приведен на рис. 3.14б. Блок-диаграмма, иконка и соединительная панель этого прибора подробно обсуждаются в главах 4, 5 и 9.

Приложение: ВП генерации аудиосигнала с помощью звуковой карты
Условия: Создан из фрагмента блок-диаграммы, нарушено множество правил
Цветовая схема: Стандартный вид панели и элементов
Расположение: Элементы расположены горизонтально, не согласованно с разъемами

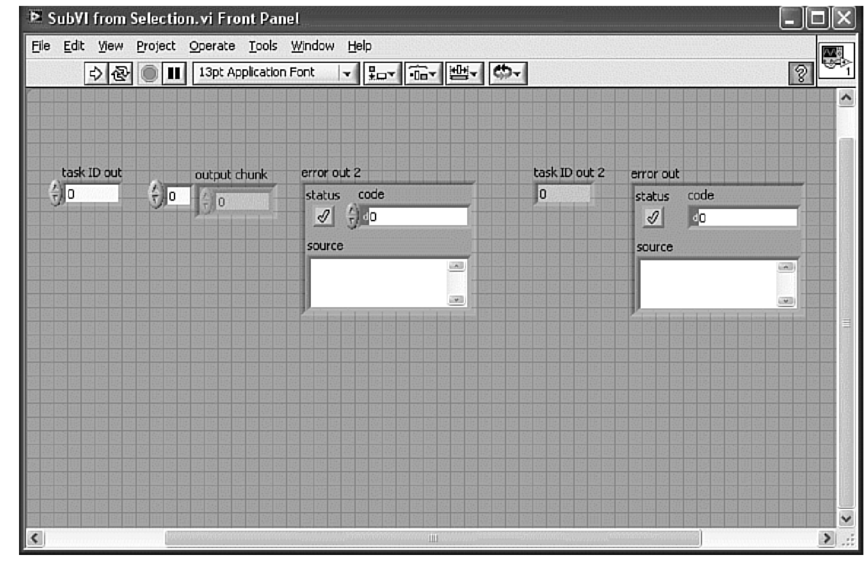


Рис. 3.14а. ВП, созданный из фрагмента блок-диаграммы (SubVI from Selection VI), нарушает Правила 3.15, 3.16, 3.21 и 3.40

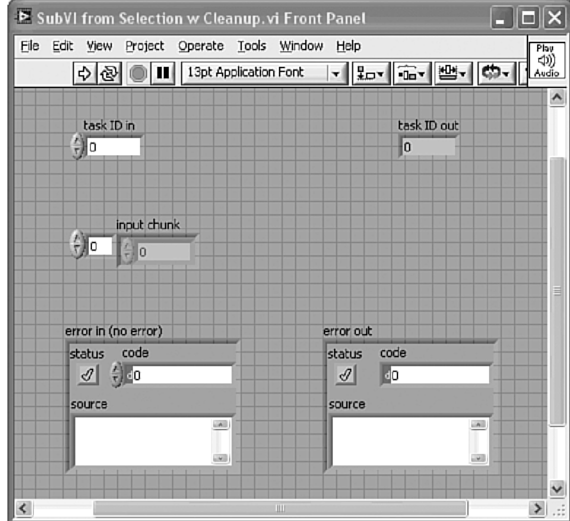


Рис. 3.14б. ВП, созданный из фрагмента блок-диаграммы после редактирования (SubVI from Selection w Cleanup VI), правила оформления ВПП не нарушены

Заголовок: Нет
Текст кнопок: Нет
Шрифт элементов: Стандартный шрифт приложения размера 13
Дополнительные шрифты: Нет
Навигация: Отсутствует в ВПП

3.5.2. Служебный диалог

ВП для обработки и передачи RS-232 Parse and Transmit Utility VI – это служебный диалоговый ВП, предлагающий пользователю настроить протокол сообщений. Свойства ВП на рис. 3.15а соответствуют свойствам диалога: внешний вид окна, расположение элементов, типы данных. Однако стиль элементов, шрифтов, элементов оформления не соответствует стандартам. А именно: для данных используется шрифт приложения размера 13, а для меток – Trebuchet MS размера 16, лучше назначить шрифт диалога в обоих случаях. Цвет панели темно-серый, размер соответствует диалогу. Метки элементов управления и текст логических элементов хорошо различимы на светло-сером и белом фоне, чего нельзя сказать о черных данных на темном фоне. Элементы можно выделять с помощью клавиши **Tab**.

Приложение: Служебное приложение настройки формата сообщений для прибора
Условия: Не соответствует правилам оформления диалога
Цветовая схема: Светло-серое обрамление элементов на темно-серой панели, элементы темно-серые с черным текстом
Расположение: Логически связанные элементы объединены с помощью элементов оформления

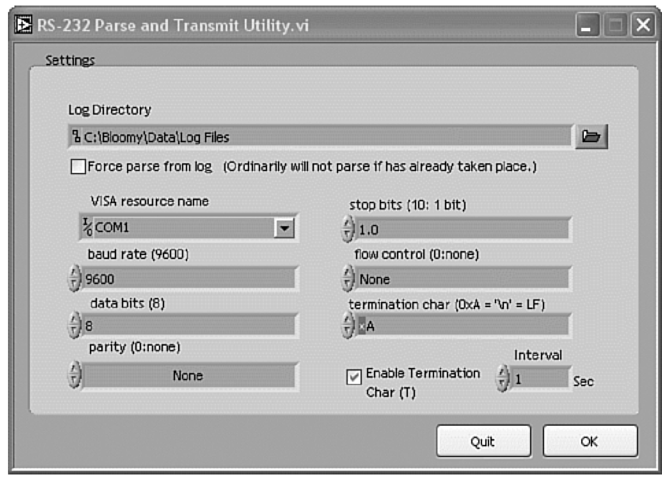


Рис. 3.15а. Внешний вид RS-232 Parse and Transmit Utility VI не соответствует стилю системы

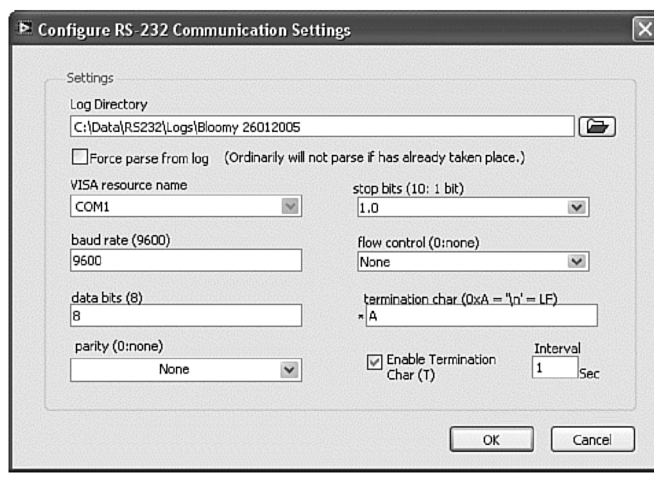


Рис. 3.15б. Внешний вид лицевой панели соответствует стилю Диалог, шрифты, элементы управления выполнены в стандартном стиле диалога

Заголовок: Trebuchet MS, размер 16
Текст логических элементов: Черный шрифт Приложение размера 13 на белом фоне
Шрифт элементов: Черный шрифт Приложение размера 13 на белом фоне
Дополнительные шрифты: Нет
Навигация: Последовательное выделение с помощью Tab

На рис. 3.15б все недостатки исправлены. Установлен стиль окна **Dialog** (Диалог), все элементы управления и оформления выбраны с системной палитры. Название ВП заменено на краткое описание назначения окна. Управляющий элемент VISA Resource Name (Имя ресурса VISA) заменен на комбинированное окно, чтобы сохранить стиль ОС. Текст кнопок **Quit** (Выход) и **OK** заменен на **Cancel** (Отмена) и **OK** соответственно, но в общепринятом порядке.

3.5.3. Тестирование и сортировка конденсаторов

На рис. 3.16а показан ВП верхнего уровня приложения тестирования и сортировки конденсаторов (Capacitor Test & Sort). Эта промышленная задача работает в производственном цикле под управлением операторов среднего уровня. Расположение элементов логичное и понятное: предварительные данные сверху, основной график в центре, кластеры статистических данных в нижней части, большие кнопки навигации вдоль нижней границы. Контраст между текстом и фоном отличный: белый на черном или синем. Однако количество шрифтов избыточное: у каждой кнопки свой. Интуитивно понятное **STOP** (Стоп) написано заглавными буквами полужирным шрифтом Диалог размера 29; **Change work order** (Изменить задание) – полужирным шрифтом размера 16; **Print** (Печать) – полужирным

шрифтом Диалога размера 29; **Exit system** (Выйти из системы) – полужирным шрифтом размера 20. То есть используется 4 разных стиля, с текстом разной длины для кнопок одного назначения. Таким образом, в оформлении кнопок нарушено сразу 4 правила: 3.17, 3.19, 3.21 и 3.40. Также у приложения есть стандартная строка меню, не допустимая для промышленных приложений, а численный индикатор со светодиодом наложены на график, нарушая правило 3.11.

Приложение: ВП верхнего уровня в задаче тестирования и сортировки конденсаторов

Условия: Разрешение 1024×768, средняя плотность элементов

Цветовая схема: Синий, черный, белый

Расположение: График достаточного размера, расположен в центре, численный индикатор и светодиод наложены на график, нарушая правило 3.11

Заголовок: Белый полужирный шрифт Приложение размера 16 на черном фоне

Текст кнопок: 4 разных шрифта разного стиля. Белый текст на черном фоне достаточно контрастный

Шрифт элементов: Черный шрифт Приложение размера 16 на белом фоне

Дополнительные шрифты: Черный шрифт Приложение размера 13. Полужирный для меток, нормальный для данных

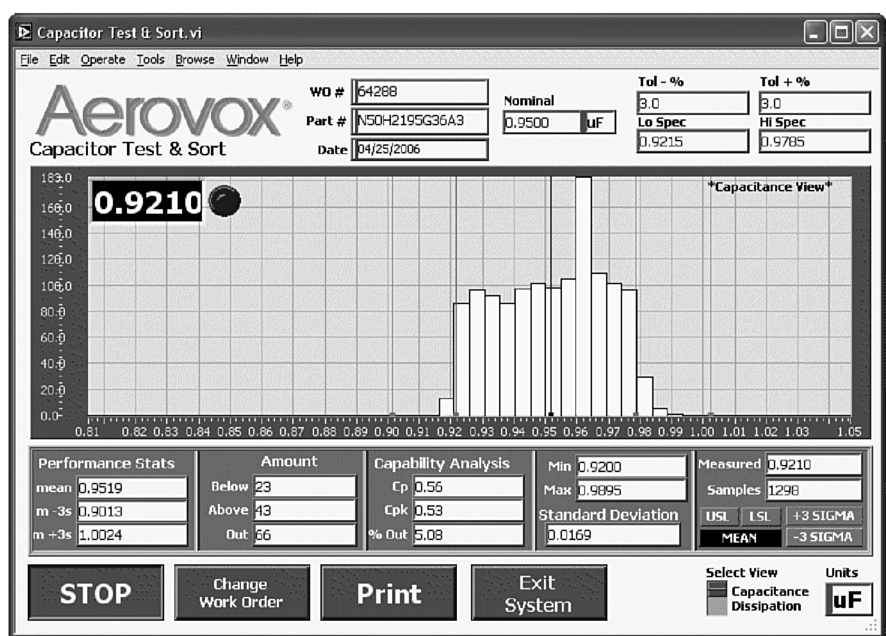


Рис. 3.16а. Расположение элементов ВП верхнего уровня в задаче тестирования конденсаторов (Capacitor Test & Sort) логично и интуитивно понятно. Стиль кнопок, меню и расположение индикаторов нарушает правила оформления

Навигация: 5 больших кнопок внизу и стандартное меню затрудняют навигацию. Стандартное меню не требуется пользователям

На рис. 3.16б исправлены недостатки текста кнопок, меню и наложенных индикаторов. Все кнопки обозначены коротким словом с первой заглавной буквой, полужирным шрифтом диалога. Команды меню: **File** (Работа с файлами), **Work Orders** (Задания), **Print** (Печать) и **Help** (Справка) помогают навигации в приложении. Индикаторы с графика перемещены наверх, увеличивая эффективность работы.

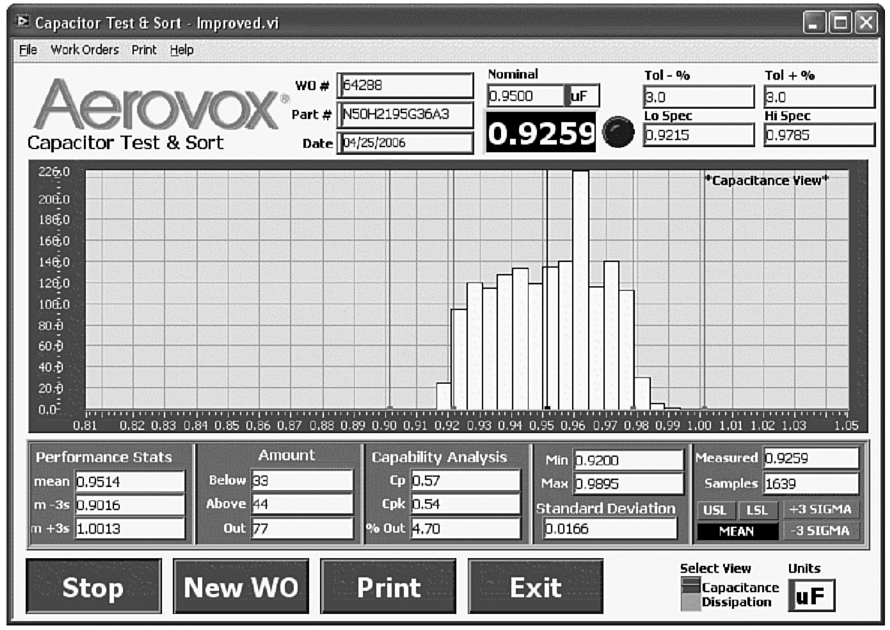


Рис. 3.16б. Исправлены недостатки текста кнопок, меню и наложенных индикаторов

3.5.4. Центрифуга

На рис. 3.17 приведен ВП для сбора данных и управления центрифугой. Приложение предназначено для проведения экспериментов над почвой при исследовании землетрясений инженерами и учеными и содержит сотни параметров. Соответствующие элементы организованы на вложенных закладках. Внешними закладками может управлять пользователь: она разделяет приложение на задачи: настройки (**Configure**), мониторинга (**Monitor**), сбора данных (**Acquire**) и анализа (**Analyze**). Управление второй панелью закладок на странице настройки пользователю напрямую недоступно: настройки каналов сбора данных зависят от типа

сенсора (элемент **Sensor selection**). Использование вложенных закладок и соответствующих данных описано в главе 6. За исключением логотипов, на лицевой панели используется шрифт Приложение трех разных стилей.

Приложение: ВП управления центрифугой
Условия: Разрешение 1024×768, высокая плотность объектов
Цветовая схема: Красно-коричневый, серый, белый
Расположение: Вложенные панели закладок, внешняя соответствует этапам работы. Сотни параметров задаются с помощью прозрачной панели закладки на странице настройки
Заголовок: Заголовки закладок черным полужирным шрифтом Приложение размера 18 на стандартном сером фоне
Текст кнопок: Черным полужирным шрифтом Приложение размера 13 на стандартном сером фоне
Шрифт элементов: Черным шрифтом Приложение размера 13 на белом фоне
Дополнительные шрифты: Черный шрифт Приложение размера 13 на красно-коричневом фоне
Навигация: Комбинация меню, закладок, списков и кнопок

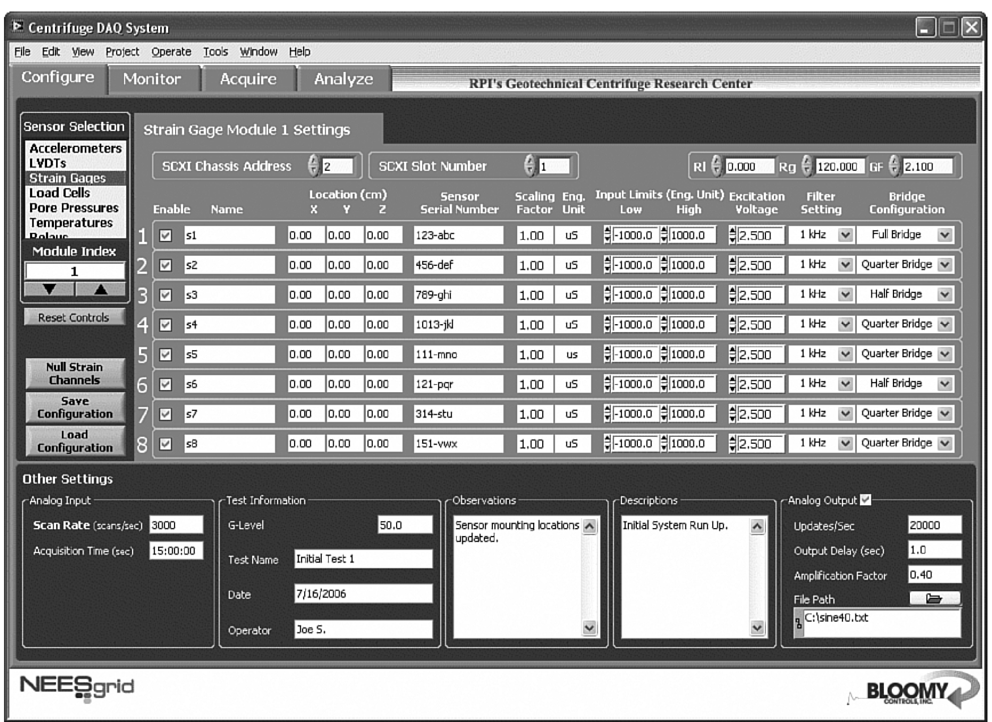


Рис. 3.17. Исследовательское приложение анализа центрифуги для симуляции сейсмической активности

3.5.5. Спектральный анализатор

На рис. 3.18 приведена лицевая панель коммерческого ВП для диагностики механического износа промышленного оборудования. Программой пользуется технический обслуживающий персонал. Логически связанные элементы объединены на панели закладок. Основной график спектра отображается в центре экрана. Обратите внимание, что кнопки в верхней части экрана видимы всегда, в зависимости от режима работы они отключаются программно и затеняются. Логотип компании размещен в свободном месте справа от графика. Правила оформления лицевой панели практически не нарушены, и она интуитивно понятна.

Приложение: Виртуальный спектральный анализатор
Условия: Различные разрешения экрана, средняя плотность объектов
Цветовая схема: Синий, белый, стандартный серый
Расположение: График большого размера в центре. Кнопки навигации программно отключаются
Заголовок: Черный шрифт Приложение размера 18 на светло-сером фоне

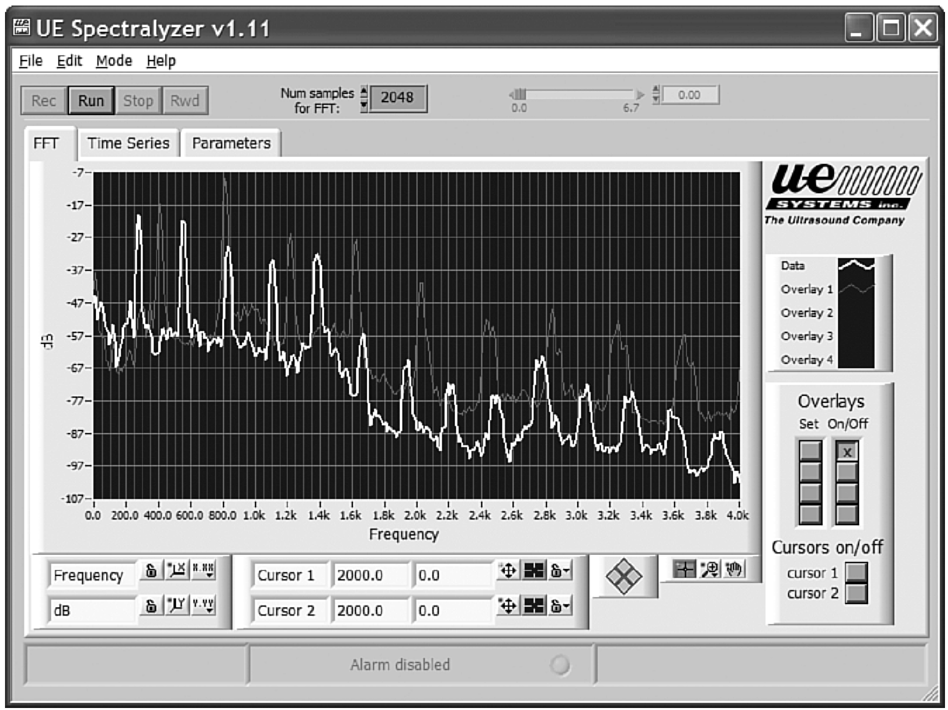


Рис. 3.18. Коммерческий ВП для диагностики механического износа промышленного оборудования. Правила оформления лицевой панели практически не нарушены, и она интуитивно понятна

Текст кнопок: Черный шрифт Приложение размера 13 на стандартном сером фоне

Шрифт элементов: Черный шрифт Приложение размера 13 на белом фоне

Дополнительные шрифты: Черный шрифт Приложение размера 12 на белом и сером фоне

Навигация: С помощью клавиши **Tab** и кнопок управления

3.5.6. Интерфейс управления парапланом

ВП управления парапланом (рис. 3.19) – это симулятор полета с регистрацией технических параметров параплана. Интерфейс представляет собой виртуальную кабину с 4 группами индикаторов: вертикальной и горизонтальной ситуации (**VSI**, **HSI**), статуса (**Status**) и управления (**Control**). Каждая группа разделена промежутками и элементами оформления. Простые числовые и строковые индикаторы с тонкими границами позволяют увеличить плотность расположения данных без потери читаемости. Стандартный шрифт **Arial** всего приложения обеспечивает одинаковый внешний вид панели вне зависимости от компьютера, ОС и

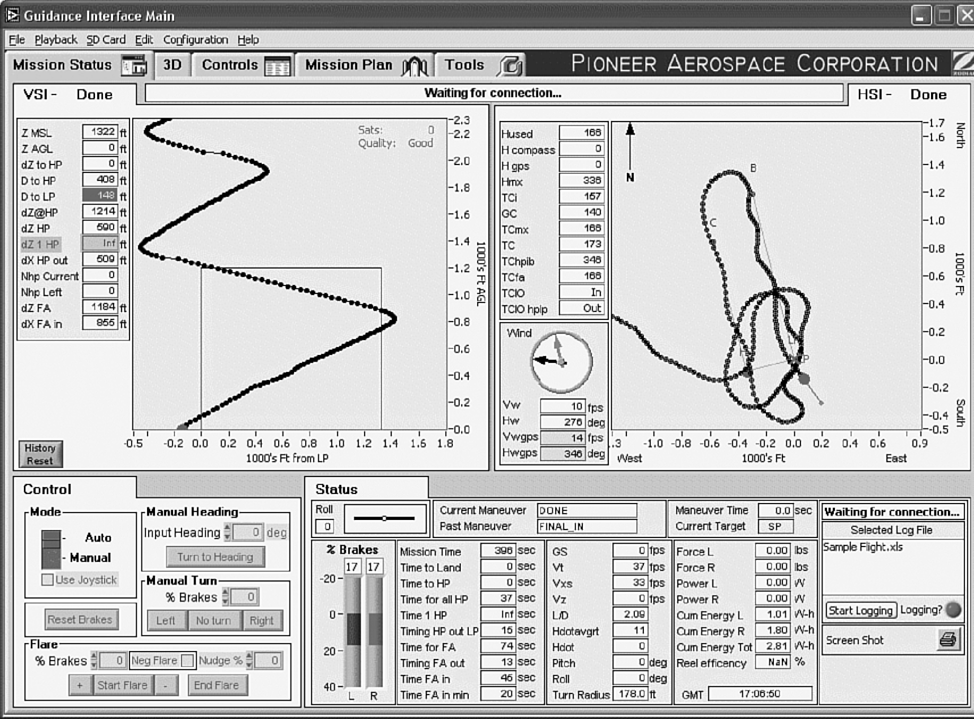


Рис. 3.19. ВП-симулятор кабины параплана.
Высокая плотность отображения данных, но без потери удобства работы

настроек пользователя. Для повышения используется черный цвет на белом или светло-сером фоне. Иконки помогают работе с программой.

Приложение: Симулятор параплана

Условия: Виртуальная кабина с максимальной плотностью расположения данных

Цветовая схема: Белый или светло-серый фон с черным текстом

Расположение: Графики достаточного размера, элементы ручного управления программно отключаются и затеняются в автоматическом режиме

Заголовок: Полужирный черный шрифт MS Sans Serif размера 15 и 18 на белом или светло-сером фоне

Текст кнопок: Черный шрифт Arial размера 13 на сером фоне

Шрифт элементов: Черный шрифт Arial размера 13 на белом фоне

Дополнительные шрифты: Черный шрифт Arial размера 14 на белом или сером фоне

Навигация: Панели закладок и меню

Ссылки

1. Ritter, David. LabVIEW GUI Essential Techniques (Руководство по интерфейсу пользователя). New York, NY: McGraw-Hill, 2002.
2. Instrument Driver Guidelines online tutorial (Руководства к драйверам приборов), NI Developer Zone (Зона разработчиков NI).
3. Rosenthal, Odeda, and Robert H. Phillips. Coping with Color-Blindness (Привычный дальтонизм). New York, NY: Avery Publishing Group, 1997.

Блок-диаграмма LabVIEW – это больше, чем средство представления исходного кода, скорее, это вид искусства. Отличная работа вызывает восхищение, плохая – удручает. Мы уже видели примеры обоих крайностей: это ВП «Дотошный» и «Спагетти» в главе 1 «Зачем нужен стиль». Где-то посередине между этими двумя типами и находятся большинство работ. У некоторых разработчиков диаграммы аккуратные, но слишком большие и простые. Другие излишне загромождают работу модульными структурами, и общий принцип приложения понять сложно. Другие пользуются локальными переменными, а не проводниками для передачи данных. Многие, слишком многие, не обращают внимание на документацию, экономя время. В результате многие диаграммы – это компромисс между реализацией хорошего стиля и попыткой сэкономить время. Получившаяся смесь в какой-то степени привлекательна и как-то работает.

Теорема 4.1 *Отличная блок-диаграмма требует минимального времени.*

Многие разработчики заблуждаются, полагая, что для создания привлекательной диаграммы потребуется время, которого не хватит для завершения проекта. Вот и жертвуют привлекательностью в попытках сделать работу быстрее. Разумеется, на то, чтобы оптимизировать внешний вид сложной диаграммы, потребуется потратить много времени, и многие признаются, что занимались этим время от времени. Однако всегда оказывается, что отладка запутанной диаграммы занимает больше времени. В соответствии с теоремой 4.1, если взять все время жизни приложения, хороший стиль отличается минимальными затратами времени и усилий. К тому же аккуратность не потребует много времени: если вы знаете правила стиля и знаете, как их применять, время только экономится.

В этой главе приведены правила, которые помогают сделать аккуратную блок-диаграмму в реальных задачах с жесткими сроками. Совместно с советами других глав они гарантируют вам удобный и понятный исходный код. Более того, они помогут сделать вашу работу красиво.

4.1. Расположение

В этом разделе приведены правила расположения основных частей диаграммы и принципы создания ВПП.

4.1.1. Основные части

Следующие правила касаются общих свойств блок-диаграммы.

Правило 4.1 Установите разрешение 1280×1024

Разрешение экрана влияет на то, какая часть блок-диаграммы видна во время работы. Разумно выбрать подходящее разрешение, чтобы окно диаграммы оставалось одинаковым во время работы на разных компьютерах. Чем больше разрешение, тем меньше по сравнению с размером экрана объекты, тем больше кода влезает на один экран. Разрешение должно быть достаточно большим, чтобы была видна большая часть программы, но без лишней нагрузки на зрение. Минимальное разрешение, с которым рекомендуется работать, – 1024×768, но современные компьютеры позволяют достаточно комфортно работать и с разрешением 1280×1024. Больше разрешение увеличивать не стоит, глаза начнут напрягаться. У меня отличное зрение, но несколько лет назад во время работы в LabVIEW мне пришлось носить очки. Выбор оптимального разрешения экрана и современный монитор позволили снять эту проблему.

Большинство современных компьютеров позволяют подключить несколько мониторов, при работе в LabVIEW это очень удобно: на одном может быть блок-диаграмма, на другом – лицевая панель, не придется постоянно переключать их окна.

Правило 4.2 Оставьте фон белым

Правило 4.3 Располагайте объекты плотно

Правило 4.4 Ограничьте размер блок-диаграммы одним экраном, по крайней мере, по ширине или высоте

Не окрашивайте блок-диаграмму. Оставьте ее фон и фон всех структур белым. Поток данных должен быть виден отчетливо. Желательно размещать объекты достаточно плотно, но так, чтобы они и проводники не перекрывались. Общее правило: ограничивайте размер блок-диаграммы одним экраном. Иногда, в больших приложениях, с несколькими параллельными циклами это правило соблюдать трудно. В этом случае расположите циклы так, чтобы диаграмма была видна при прокрутке только в одном направлении или преобразуйте циклы в ВПП. После-

дний метод подробно обсуждается в главе 8 «Шаблоны программирования». Избегайте больших диаграмм, выходящих за границы экрана, в них сложно ориентироваться.

На рис. 4.1 приведено два аналогичных калибровочных ВП. Первый, на рис. 4.1а, слишком насыщенный и сложный. Как видно, в некоторых случаях проводники и объекты перекрываются, а это запутывает. На рис. 4.1б показан исправленный ВП. Функции расположены равномерно, проводники проложены аккуратно, добавлены комментарии, через всю диаграмму проходит кластер ошибок. Код получился гораздо приятнее. В скором времени мы вернемся к этому примеру.

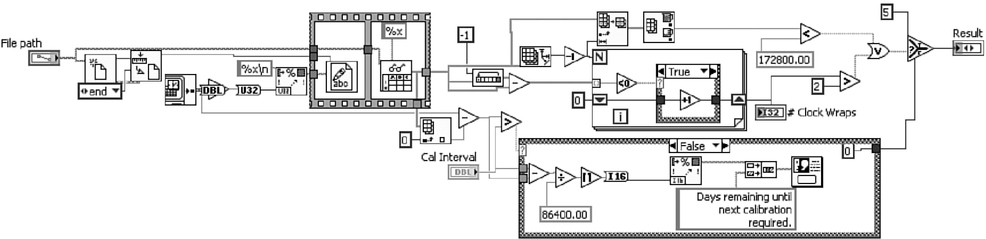


Рис. 4.1а. Блок-диаграмма калибровочного ВП слишком насыщенная и запутанная

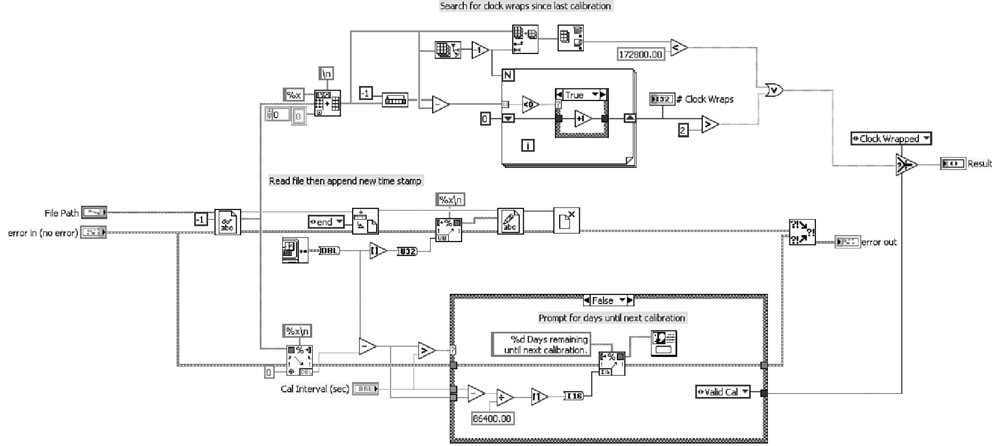


Рис. 4.1б. Исправленная блок-диаграмма, с достаточными промежутками, аккуратными проводниками и комментариями

4.1.2. Блоки в ВПП

Если ваша блок-диаграмма выходит за границы одного экрана при разрешении 1280×1024, то ваш код недостаточно модульный. Нужно заменить окно навигации (Navigation window) на ВПП.

Правило 4.5 Создайте многоуровневую архитектуру ВПП
• Повысьте индекс модульности до 3,0

Структура приложения должна представлять многоуровневую архитектуру ВПП только с горизонтальной или вертикальной прокруткой. Свойства иерархии ВП (VI Hierarchy) можно посмотреть с помощью соответствующей команды меню: **View ⇒ VI Hierarchy**. Исключите из анализа стандартные ВП (VI Lib), глобальные переменные и определения типов. Оставьте только объекты и подприборы. Структуру можно представить в виде пирамиды, алмаза или овала. За исключением самых простых задач она должна представлять собой один ВП верхнего уровня и строки подприборов. В главе 1 мы определили индекс модульности как отношение количества ВП к полному числу узлов, умноженному на 100. Эти данные приводятся в окне метрики ВП (**Tools ⇒ Profile ⇒ VI Metrics**). Для большинства приложений оптимальный индекс модульности равен 3.

Правило 4.6 Создавайте модульные программы с помощью ВПП
• Разрабатывайте ВП-компоненты высокого уровня
• Замените множество узлов свойств на ссылки элементов в ВПП

В зависимости от выбранного шаблона, диаграмма ВП верхнего уровня должна состоять исключительно из структур, проводников, ВП-компонентов и подприборов. ВП-компоненты (Component VIs) – это ВП высокого уровня или динамически загружаемая библиотека ВП, содержащая в себе законченную подсистему приложения. Например, система сбора данных приложения или графический интерфейс, которые вызываются через один ВПП, относятся к таким элементам. На диаграмме ВП верхнего уровня практически не должно быть функций обработки данных: математических, форматирования строк, обработки массивов и др.

В некоторых приложениях для управления интерфейсом используется множество узлов свойств. Большинство их действий зависят от событий интерфейса, поэтому для их обработки идеально подходит структура событий (Event Structure). Каждому событию соответствует определенный кадр, поэтому код не занимает много места на экране. Но использовать разные кадры для изменения свойств одинакового набора элементов неудобно. Лучше передать массив ссылок (Control References) и значений свойств в ВПП. Этот прием позволяет сэкономить память и уменьшить сложность блок-диаграммы.

Правило 4.7 Используйте ВПП в ВПП
• Преобразуйте обособленные участки кода в ВПП
• Разработайте и/или используйте драйвера приборов и служебные ВПП

По аналогиии необходимо преобразовать ВПП высокого уровня: замените обособленные участки кода на ВПП. Также замените на ВПП все группы связанных функций. Признак обособленного участка, который необходимо преобразовать в ВПП: его назначение можно полностью описать 2–3 предложениями. Получившиеся ВПП легко «склеить» в программу более высокого уровня.

Проверка обособленности: *операцию можно полностью описать 2–3 предложениями.*

Также модульность программы повышают драйверы приборов и служебные ВП. В драйверах приборов содержится набор низкоуровневых функций управления прибором, включая передачу строковых данных, функции VISA и преобразование возвращенной строки. Служебные ВПП расширяют возможности функций LabVIEW. Тысячи драйверов и служебных ВПП можно скачать в сети[1]. Повторное использование кода мы обсудили в главе 2 «Подготовка к хорошему стилю».

Вернемся к ВП калибровки (с рис. 4.1) и рассмотрим рис. 4.2. Это ВП высокого уровня. Внимательно посмотрев на блок-диаграмму, можно выделить 3 основных операции: чтение и добавление интервалов времени в файл, поиск и подсчет дат и запрос: когда проводить следующую калибровку. Как показано на рис. 4.2б, каждую операцию можно преобразовать в отдельный ВПП, расположенный на месте соответствующего участка кода. На рис. 4.2в ВПП расположены один за другим и связаны кластером ошибок. В результате сложный ВПП превратился в вызов трех функций. Вся структура приложения стала гораздо аккуратнее.

Правило 4.8 Не добавляйте ВПП просто для экономии места

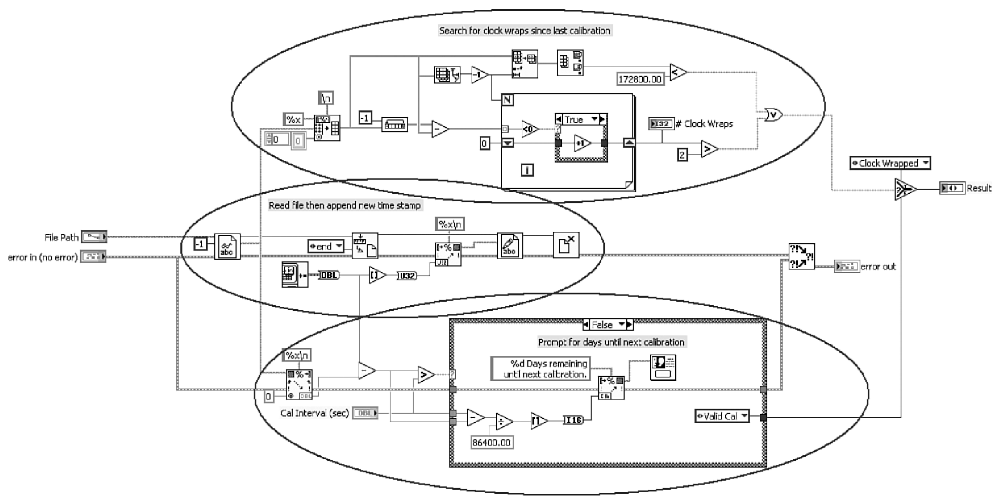


Рис. 4.2а. Калибровочный ВП состоит из трех частей

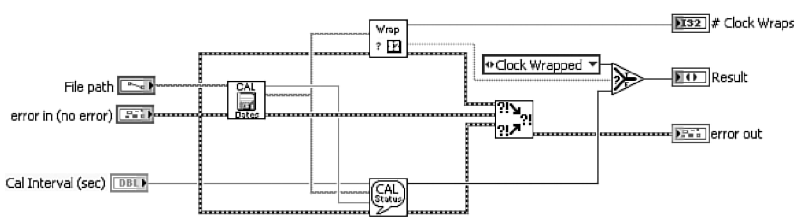


Рис. 4.2б. Каждую часть можно заменить на подприбор

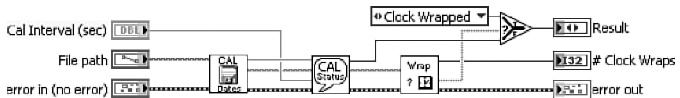


Рис. 4.2в. Соединим ВПП последовательно, связав их кластером ошибок

Правило 4.9 Избегайте тривиальных ВПП

ВПП предназначены для облегчения разработки, тестирования, отладки, поддержки и повторного использования отдельных модулей приложения. Если одинаковый набор функций или узлов свойств используется в разных частях блок-диаграммы, его нужно заменить на ВПП. Если группа обособленных узлов выполняет одну операцию, ее тоже нужно заменить на ВПП. Но случайно выбирать участки блок-диаграммы и преобразовывать их в подприборы просто для экономии места не стоит. Такие ВПП сложно будет использовать повторно, их основная функция непонятна. Также вредны ВПП с небольшим числом элементов. В этом случае ВПП просто скрывает внутренний код. Например, на рис. 4.3 ВПП предназначен для выделения элемента из массива с помощью одной функции Index Array. Внутренняя стандартная функция LabVIEW из vi.lib узнается сразу, а иконка, имя и описание ВПП ее прячут.

Правило 4.10 Иконка и понятное описание должны быть у каждого ВПП

Всегда создавайте наглядную иконку и понятное описание для каждого ВПП. Значение этого правила переоценить невозможно. В Bloomy controls оно обязательно к выполнению. Благодаря иконке и описанию всегда понятно назначение ВПП из вызывающей программы. Описание – это проверка на обособленность. Если вы не можете кратко описать ВП, значит, вы неправильно построили архитектуру приложения. Более подробно иконки и описания обсуждаются в главе 5 «Иконка и контакты» и в главе 9 «Документация».

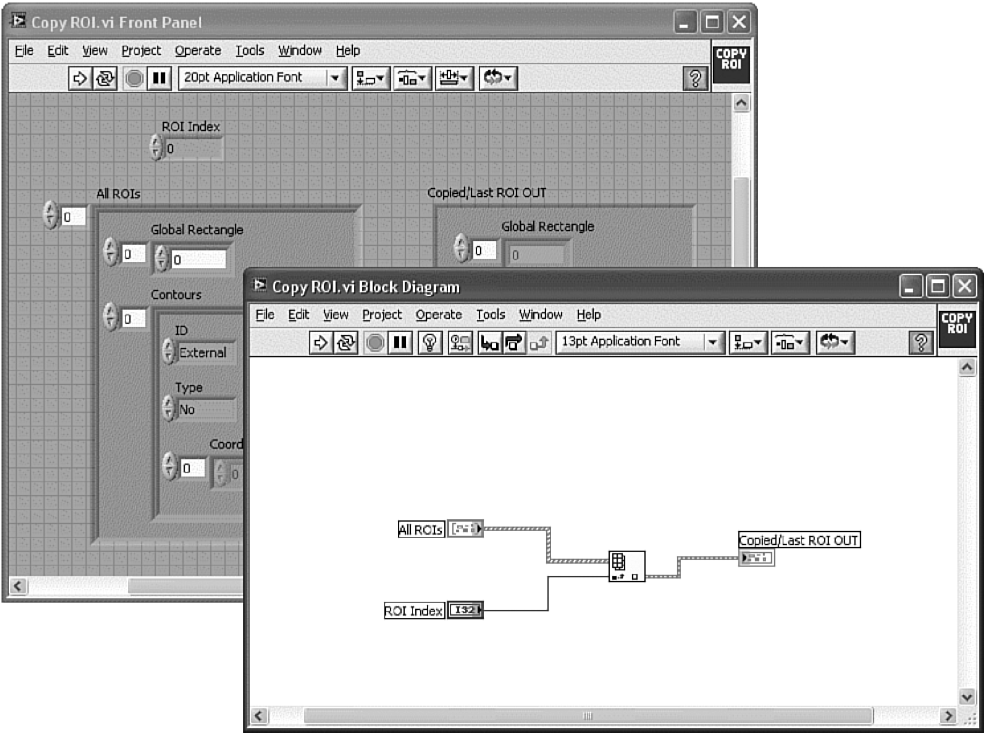


Рис. 4.3. В этом ВПП слишком мало узлов.
Его иконка, имя и описание маскируют функцию выбора элемента массива

4.2. Соединения

Правила этой главы позволят вам сделать ясную сеть соединений в программе. Также обсуждается объединение данных в кластеры.

4.2.1. Секреты аккуратного соединения

Правило 4.11 Избегайте петель и изгибов проводников

Чем меньше изгибов и петель на проводнике, тем понятнее будет диаграмма. Проводники можно прокладывать и оптимизировать как вручную, так и автоматически. Я предпочитаю проводить проводники вручную, потому что в автоматическом режиме появляются лишние изгибы вместо наложения на элементы. Избавиться от них можно только вручную. Автоматический режим можно отключить временно или полностью. Чтобы провести проводник вручную, прижмите клавишу **A**, начав соединение. Автоматический режим можно отключить в свойствах

LabVIEW: **Tools ⇒ Options ⇒ Block Diagram ⇒ Enable auto wiring** (Инструменты ⇒ Опции ⇒ Блок-диаграмма ⇒ Автоматический режим прокладки проводников). Обычно я сначала вручную соединяю элементы, а потом очищаю изгибы, в случае необходимости передвигая узлы и сегменты проводников. Также команда контекстного меню проводника **Clean Up Wire** позволит оптимизировать проводник автоматически.

Правило 4.12 Параллельные проводники должны проходить на одинаковом расстоянии

Расстояние между параллельными проводниками должно быть одинаковым на всем протяжении: включая изгибы и узлы. Во-первых, у всех связанных ВПП расположение разъемов должно подчиняться одному шаблону. Более подробно шаблоны расположения разъемов обсуждаются в главе 5. Во-вторых, перед соединением ВПП подровняйте их разъемы. Просто выберите элементы и выровняйте их средствами меню **Align Objects** (Выравнивание объектов). Блок-диаграмма ВПП формирования отчета (рис. 4.4) состоит из нескольких последовательных операций записи данных в файл. Проводники ссылки на файл (file refnum) и кластера ошибок параллельны. Благодаря расположению соответствующих разъемов на функциях работы с файлом, параллельность сохраняется на протяжении всей диаграммы. Однако при малейшем смещении на проводнике возникают изгибы. Блок-диаграмма на рис. 4.4а получена простым переносом функций с палитры на панель. В результате использования команды **Align Bottom Edges** (Выровнять нижние края) (рис. 4.4б) диаграмма стала гораздо аккуратнее – рис. 4.4в. Также с помощью соответствующего меню можно задать промежуток между функциями, но в данном случае нужно дополнительное место для цикла. Итоговая блок-диаграмма со строго параллельными проводниками приведена на рис. 4.4г.

Правило 4.13 Туннели должны быть расположены на вертикальных границах структур

Правило 4.14 Не проводите через структуры лишний раз

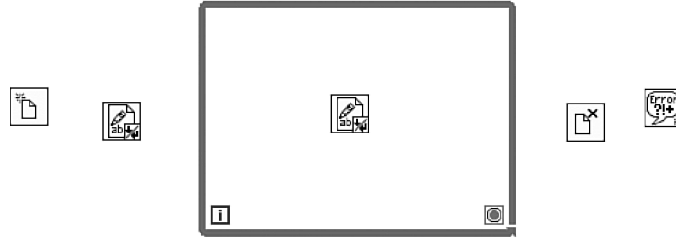


Рис. 4.4а. Для регистрации данных понадобятся несколько функций, шаблон разъемов у них одинаковый

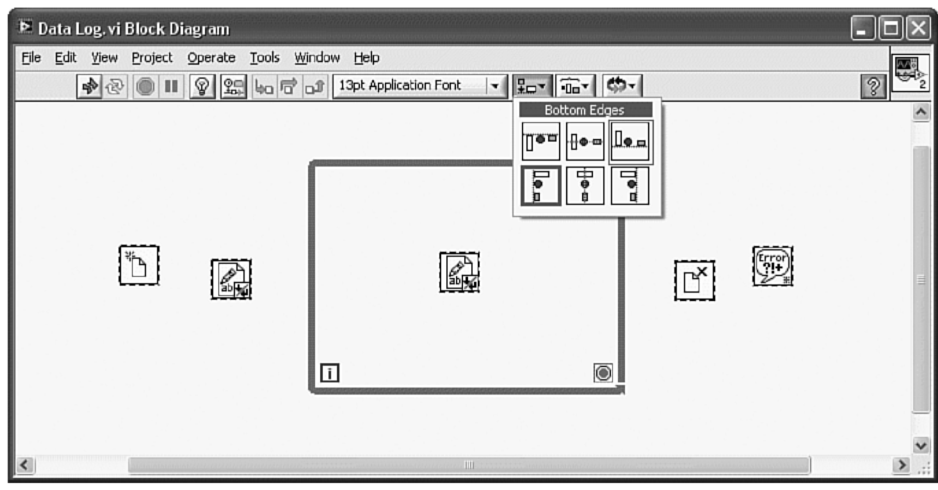


*Рис. 4.4б. С помощью команды **Align Bottom Edges** (Выровнять нижние края) функции располагаются на одной линии*

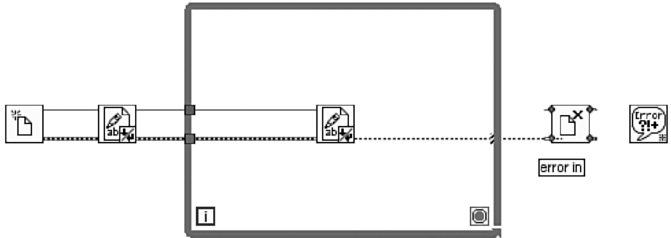


Рис. 4.4в. Проводники ссылки на файл (file refnum) и кластера ошибок параллельны, нет ни одного изгиба

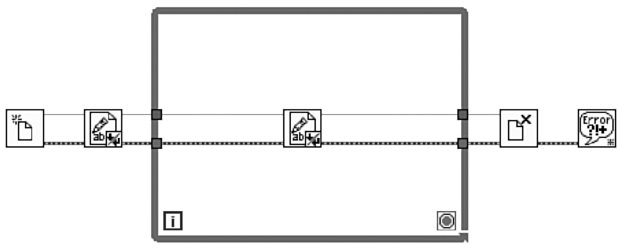


Рис. 4.4г. Соединение завершено. Можно вручную подобрать нужные расстояния между элементами

Проводник должен входить в структуру на ее левой границе и выходить на правой. Избегайте туннелей на верхней и нижней границах. Также не стоит пропускать неиспользуемые проводники через структуру без необходимости. Неприятно лазать по кадрам сложной структуры выбора (**Case**) или структуры событий (**Event structure**) в поисках места изменения данных. Однако в некоторых шаблонах, например конечном автомате, иногда через структуру пропускают пустые проводники, они могут понадобится на более поздних стадиях разработки. Такие случаи обсуждаются в главе 8. Не забудьте добавить к такому проводнику метку с пояснением: не используется.

Правило 4.15 Никогда не закрывайте узлы и проводники
• Избегайте наложения объектов на блок-диаграмме

Избегайте наложения объектов, из-за которых скрываются участки проводников или другие элементы блок-диаграммы. Случайное пересечение вертикального и горизонтального проводника не рекомендуется. Например, иногда необходимо передать номер итерации, терминал которого обычно расположен слева внизу, в верхнюю правую часть цикла. Через большинство циклов проходят горизонтальные проводники, хотя бы кластер ошибок. Если терминал числа итераций необходим в нижнем углу, пересечения не избежать, постарайтесь снизить их количество насколько возможно. Также не ведите проводник под узлом или объектом, его можно перепутать с входными или выходными данными.

Правило 4.16. Ограничивайте длину проводника: его начало и конец должны быть видны на экране

Правило 4.17. Никогда не используйте переменные просто ради удобства соединений

Правило 4.18. Добавляйте метки для длинных проводников

В идеале, начало и конец проводника должны быть видны на экране. Но это не всегда возможно, даже в небольших диаграммах: например, если данные передаются внутрь сложной структуры. В этом случае обязательно добавьте метки в тех местах, где начало или конец скрыты. Сокращать длину проводника важно, но лучше длинный проводник, чем вообще никакого. Локальные и глобальные переменные не предназначены для сокращения числа проводников. Их обработка занимает много процессорного времени, памяти и усложняет структуру приложения. Более того, переменные нарушают принцип непрерывного потока данных LabVIEW. Когда данные записываются и используются в разных частях диаграммы, сложно найти ошибку в порядке доступа. С проводниками всегда можно указать источник данных. Если конец проводника не виден, обязательно добавьте

метку с указанием источника. Знак больше (>) или меньше (<) позволяет указать направление потока данных. Правила использования переменных обсуждаются в разделе 4.3. «Поток данных».

Правило 4.19 Размещайте неиспользуемые терминалы элементов отдельно

Неиспользуемые терминалы элементов должны быть расположены в соответствующем месте, там, где программист может легко найти их. Обратите внимание, что данные с терминалов вне повторяющихся структур считываются только 1 раз. Кнопки (с действием latch), например, не могут вернуться в исходное состояние. Если элементу соответствует событие, поместите его в соответствующий кадр структуры событий. В этом случае значение элемента будет считываться каждый раз после события. Если терминал не используется и не связан с событием, поместите его слева от основной структуры программы.

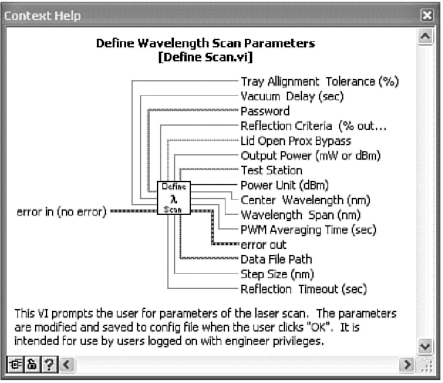
4.2.2. Использование кластеров

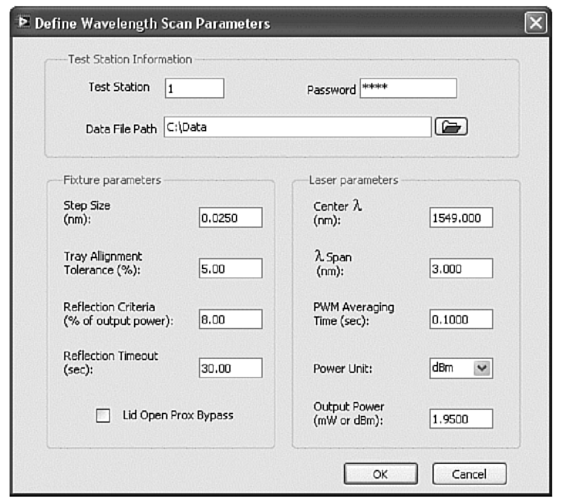
Правило 4.20 Объединяйте связанные данные в кластеры

С меньшим количеством проводников гораздо легче работать и делать красивую блок-диаграмму. Используйте кластеры, чтобы объединить связанные данные и уменьшить количество проводников. Если передается набор данных из одной области в другую, почему бы не объединить их в кластер, подобно тому, как мы объединяли связанные функции в ВПП? Но данные в кластере должны быть взаимосвязаны.

Например, рассмотрим задачу тестирования оптического фильтра (рис. 4.5). В приложении последовательно вводятся параметры сканирования, настраивается лазер, измеряется пропускание, отображаются на графиках и сохраняются в файл данные. На рис. 4.5а приведена лицевая панель, скрытые индикаторы и описание ВП **Define Scan** – диалогового ВП настройки параметров. Этот ВП – первый в последовательности исполнения (рис. 4.5б), в нем определяются все параметры, которые используются в дальнейшем. Как видно из описания ВП, на соединительной панели множество разъемов, соответствующих каждому параметру. На рис. 4.5б эти проводники проходят через несколько простых элементов. Соединительная панель ВП сохранения результатов сканирования (**Save scan**) также очень насыщенная. Блок-диаграмма просто читается благодаря грамотному расположению проводников, но на ее создание затрачены немалые усилия, еще больше потребуется для минимальных изменений. При изменении состава параметров придется переделать всю сеть проводников, все ВПП и разъемы.

На рис. 4.5в приведен другой вариант ВП Define Scan, в котором все параметры объединены в кластер. Схема использования параметров изменена для кластера (рис. 4.5г). Количество проводников и затраченные усилия значительно сократились. Также можно добавлять параметры в кластер без необходимости изменять





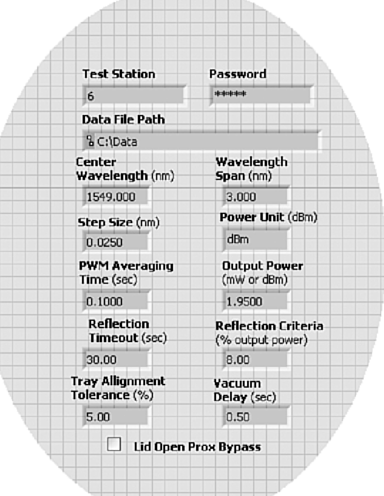


Рис. 4.5а. ВП Define Scan – диалоговый ВП настройки 15 параметров, которые передаются по отдельным проводникам

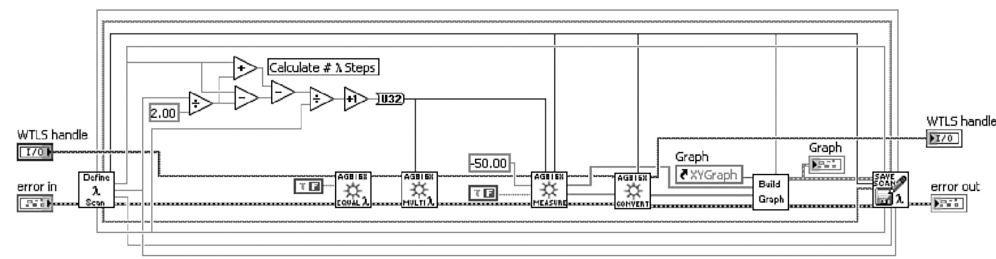


Рис. 4.5б. Программа измерений запускает настройку параметров и передает полученные данные по отдельным проводникам. Диаграмма очень насыщенная, работать с ней сложно

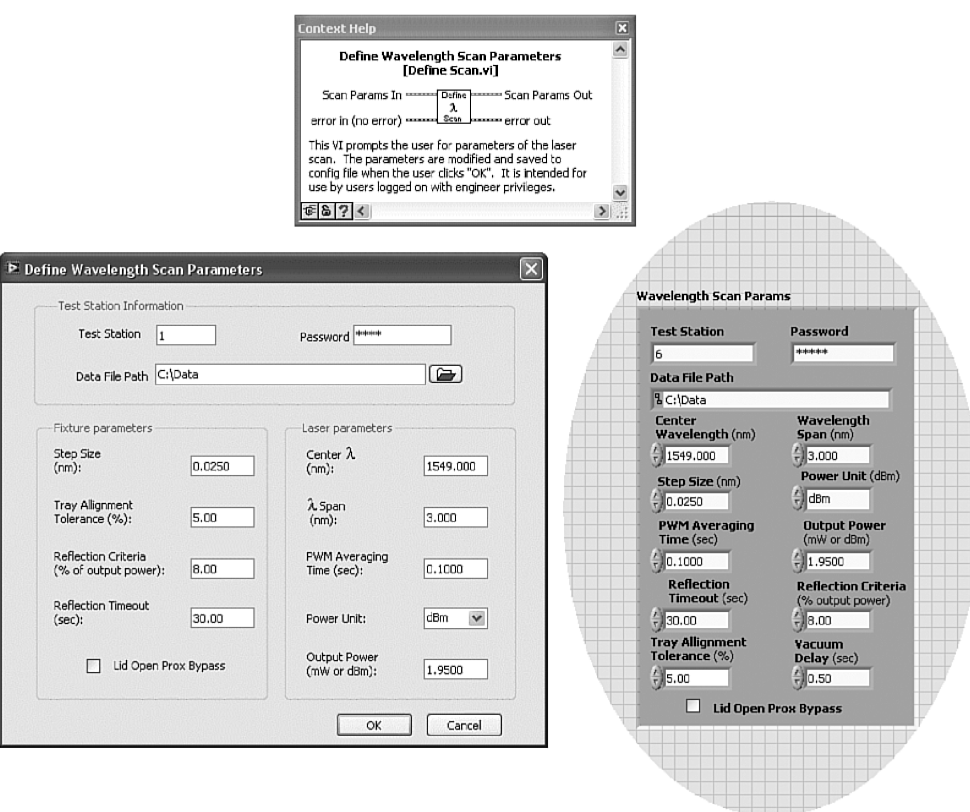


Рис. 4.5в. Параметры сканирования объединены в кластер.
Работать с разъемами стало намного проще

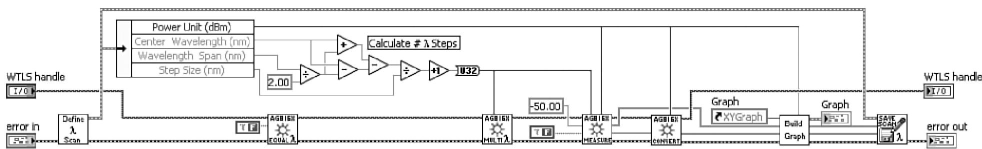


Рис. 4.5г. Кластер параметров проходит через всю диаграмму.
Количество проводников уменьшено. Можно добавлять параметры
без изменения блок-диаграммы и ВПП

разъемы и ВПП. Поэтому кластеры могут помочь улучшить внешний вид диаграммы и сэкономить время и усилия разработчика.

В некоторых ситуациях удобно объединить в кластер всего 2 параметра. Маленькие кластеры используются, если данные в них очень тесно связаны и, за редкими исключениями, используются вместе. Например, верхний и нижний пределы параметра можно считать из базы данных, предложить ввести пользователю, сравнить измеренные значения с этим диапазоном и сгенерировать отчет и файл с результатами. Кластер из двух элементов – верхнего и нижнего предела – позволяет использовать только один проводник, логически связывающий операции. В этом случае большая ценность кластера не в уменьшении числа проводов, а в объединении данных.

В примере с тестированием оптического фильтра, длина волны и массив мощностей передаются по двум отдельным проводникам. Как видно из рис. 4.6а, эти данные используются вместе, за исключением ВП преобразования мощности Convert Power Units. На рис. 4.6б этот ВП перенесен внутрь ВП Measure, и мощность преобразуется в более удобные единицы. Также длина волны и мощность объединены в кластер, а участок кода, вычисляющий число разбиений длины волны, преобразован в ВПП.

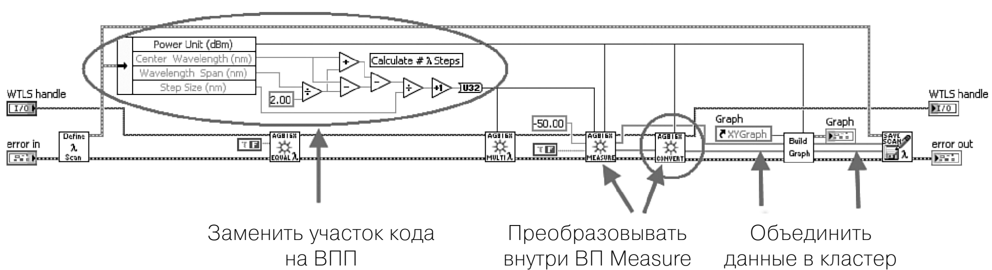


Рис. 4.6а. Два отдельных проводника длины волны и мощности
используются вместе и могут быть объединены в кластер.
Программа вычисления количества разбиений длины волны преобразована в ВПП.
Преобразование мощности перенесено в ВП Measure

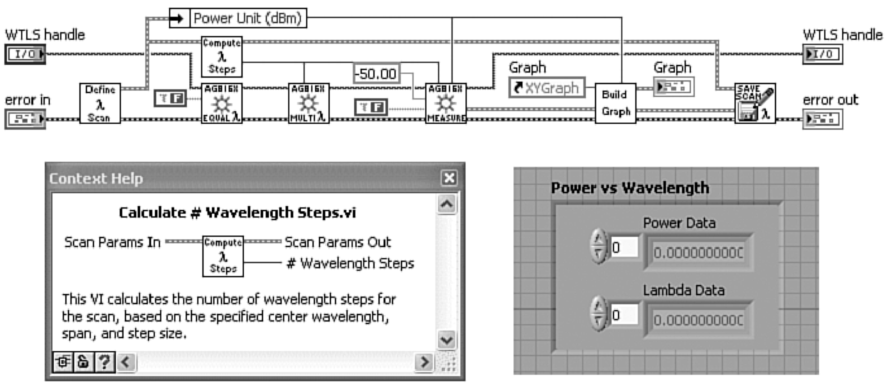


Рис. 4.6б. Модульность приложения увеличена
с помощью ВП **Calculate # Wavelength Steps VI** вычисления количества разбиений
длины волны и кластера, объединяющего длину волны и мощность

Правило 4.21 Сохраняйте кластеры как определения типа

Я не могу переоценить значение этого правила. (Обсуждение продолжается в главе 6 «Структуры данных»). Сохраняйте каждый кластер как определение типа или строгое определение типа. Определение типа (**type definition**, или сокращенно **typedef**) – это элемент лицевой панели, параметры которого хранятся в файле CTL. Чтобы вставить определение типа на лицевую панель в качестве элемента управленияили диаграмму в качестве константы, просто перетащите его из окна проекта или выберите с помощью **Select a Control** (Выбрать элемент лицевой панели) из палитры **Controls**, или **Select a VI** (Выбрать ВП) из палитры **Functions**. Тип данных всех копий определения типа задается в исходном файле. Таким образом, всеми этими копиями данных можно управлять из одного места: редактора элементов (Control Editor). Наиболее часто этот прием используется именно для кластеров, потому что они используются во многих ВП, а состав может изменяться. Например, чтобы добавить новые параметры сканирования (**Wavelength Scan Params**) в блок-диаграмму на рис. 4.5в, просто вставьте их в кластер в определении типа. Все ВПП на рис. 4.5г с этим определением типа обновятся сразу после сохранения файла.

Строгое определение типа (**strict type definition**), также известное как **strict typedef**, хранит в файле не только тип данных, но и все свойства элемента, поэтому у различных копий строгого определения типа одинаков и внешний вид и поведение. Я предпочитаю пользоваться именно ими, потому что они позволяют синхронизировать диапазон, значение по умолчанию и внешний вид всех копий. Тип элемента лицевой панели определяется в редакторе элементов в списке статуса (**Typedef Status**)). Более подробно определения типа и кластеры обсуждаются в главе 6.

4.3. Поток данных

В этом разделе обсуждается поток данных (data flow) – основа LabVIEW. Описываются основные правила программирования и стиля, в частности использования переменных и структур последовательности, а также методы оптимизации потока данных. Примеры включают как простейшие блоки, так и законченные приложения.

4.3.1. Основы потока данных

В LabVIEW данные двигаются от терминала источника данных до принимающего терминала. Любой узел блок-диаграммы исполняется, только если он получил данные по всем входным терминалам. После завершения работы данные формируются на выходных терминалах узла и передаются дальше. Поток данных – это основное отличие LabVIEW от текстовых языков программирования.

Вот несколько основных правил работы с потоком данных.

Правило 4.22 Всегда направляйте поток данных слева направо

Правило 4.23 Пользуйтесь кластером ошибок

За редкими исключениями, данные двигаются слева направо. Это священное и основное правило при программировании в LabVIEW. Исключения – узел обратной связи (feedback node) и локальный терминал (sequence local). По моему мнению, узел обратной связи – удобное исключение: короткий участок с неправильным направлением иногда позволяет избавиться от части проводников сдвигового регистра. Локальные терминалы используются в компактных структурах последовательности (Stacked Sequence), которых лучше избегать.

Посмотрим внимательно на участок кода из ВП «Спагетти» главы 1 (рис. 4.7): сразу видно несколько проводников с неправильным направлением потока данных, некоторые из них выделены. Есть и другие нарушения: лишние петли и изгибы (Правило 4.11), перекрывающиеся проводники и узлы (Правило 4.15), множество локальных переменных (Правило 4.17) и другие нарушения, которые и привели к образованию спагетти.

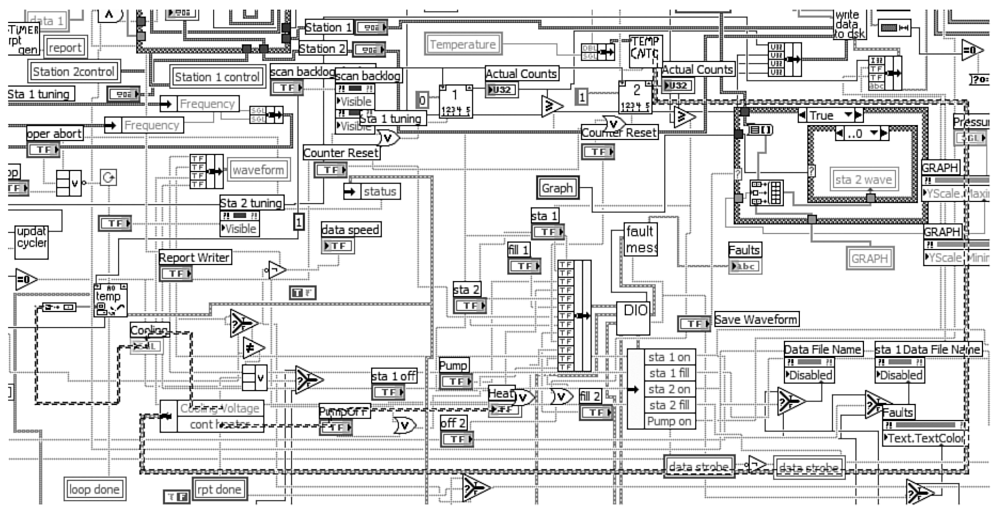


Рис. 4.7. Направление потока данных некоторых проводников на ВП «Спагетти» нарушает Правило 4.22

Чтобы не нарушать правило потока данных, сначала расположите связанные узлы в нужном порядке, а потом соедините их. В большинстве случаев порядок работы определяется взаимосвязью данных. Просто соедините общие параметры, например кластер ошибок, в нужном порядке, как показано на рис. 4.4.

Правило 4.24 Избегайте приведения типов у кластеров и массивов

Точка приведения типа (coercion dot) появляется, если вы соединили численные данные с разным представлением (representation). По умолчанию они серые в версиях LabVIEW до 8.2 и красные, начиная с этой версии. Цвет можно изменить в параметрах LabVIEW: **Tools ⇒ Options ⇒ Colors** (Инструменты ⇒ Опции ⇒ Цвета), отключите опцию **use default colors** (Использование стандартных цветов) и выберите нужный цвет точки приведения типов. Эта точка показывает, что LabVIEW преобразует данные из одного типа в другой, на эту операцию требуется дополнительная память и процессорное время. Избегайте точек приведения типа, особенно в массивах и кластерах. Методы использования одинаковых типов данных описаны в главе 6. Следите за согласованием типов!

Правило 4.25 Создавайте константы и элементы управления из контекстного меню терминалов

При создании индикаторов, элементов управления и констант с помощью контекстного меню терминала **Create ⇒ <Control/Indicator/Constant>** (Создать ⇒ <индикатор/ элемент управления / константу>) вы не только экономите время, но и соблюдаете одинаковый тип данных. Автоматически создается элемент нужного типа, соединенный с терминалом и с меткой этого терминала. Одной командой – сразу 4 действия!

Правило 4.26 Отключите точки на узлах проводников

Эти точки не несут никакой функциональной нагрузки, они просто привлекают ваше внимание к пересечению. По умолчанию они больше и выделяются внешне лучше точек приведения типа. На насыщенной диаграмме они помогают отличать узлы от пересечений без контакта, но если вы стараетесь избегать пересечений, то выделять соединения не нужно. Я считаю, что большие узлы проводников немного портят аккуратную диаграмму с правильным потоком данных. Все узлы проводника можно увидеть с помощью тройного щелчка, поэтому я советую отключить эту опцию: меню **Tools ⇒ Options ⇒ Block Diagram ⇒ Show dots at wire junctions** (Инструменты ⇒ Опции ⇒ Блок-диаграмма ⇒ Показать точки на узлах проводников).

Правило 4.27 Избегайте структур последовательности без необходимости

Пользуйтесь структурами последовательности, только если без них не обойтись, в частности не навязывайте последовательное исполнение участков кода, если их можно выполнить параллельно. Это дурная привычка текстовых языков программирования, когда инструкции исполняются в порядке их записи. Избегайте структур последовательности, наоборот, старайтесь, чтобы функции исполнялись параллельно. Например, показанная на рис. 4.8а структура не нужна, как и переменные. После исправления этих недостатков диаграмма становится более аккуратной и эффективной (рис. 4.8б). Правила использования структур последовательности приведены ниже.

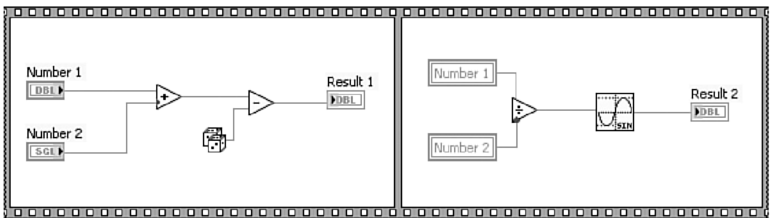


Рис. 4.8а. Последовательное исполнение участков кода не требуется, структура последовательности не нужна

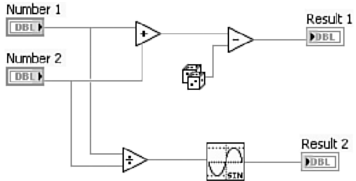


Рис. 4.8б. Вместо структуры последовательности программу можно разбить на два параллельных потока – ее эффективность повысится

Правило 4.28 Избегайте глубоких (больше 3) вложений

Вложение – это размещение одних структур внутри других. Как мы видели на примере ВП «Многоуровневый» в главе 1, при увеличении уровня вложений сложно понять логику исполнения ВП, проследить поток данных и возможные варианты исполнения. Большой уровень вложенности часто бывает следствием непродуманной структуры программы, большого числа структур последовательности и незнания стандартных шаблонов. Это очень плохой признак, потому что нарушенная логика программы затрудняет поиск и локализацию ошибок. Потратьте какое-то время и составьте блок-схему программы, таблицу истинности и карту Карно, они помогут вам понять логику работы программы, и только потом приступайте собственно к программированию. В главе 8 перечислены шаблоны, которые позволяют снизить уровень вложенности. Но даже в самых простых программах и шаблонах есть вложенные структуры. Стандартные шаблоны можно узнать сразу, поэтому при подсчете вложенности их учитывать не нужно. Сохраняйте программу, выбрав наиболее важный кадр структуры – при следующем за-

пуске диаграмма откроется там же. Самый важный кадр структуры исполняется чаще всех или содержит большинство узлов. Также отметим, что если вы соблюдаете правила потока данных и редко пользуетесь структурами последовательности, уровень вложенности будет минимальным.

4.3.2. Когда нужны переменные и последовательности

Сколько раз я уже повторял: избегайте переменных и структур последовательности, они нарушают поток данных. Давайте все же узнаем, зачем они могут понадобиться, иногда без них не обойтись.

Правило 4.29 Инициализируйте элементы управления с помощью локальных переменных

Правило 4.30 Передавайте данные между простыми параллельными ВП с помощью глобальных переменных

Локальные переменные – это самый удобный способ задания значений элементов управления. Часто необходимо программно установить начальные значения, например, считав их из файла. Также локальные и глобальные переменные позволяют обмениваться данными между параллельными процессами: циклами одного ВП или нескольких. Но у переменных чтения данных нет входных параметров, и наоборот, поэтому порядок этих действий нельзя задать с помощью потока данных. У узлов свойств и разделяемых переменных есть кластеры ошибок, ими можно пользоваться вместо переменных. Узел свойств, считывающий или записывающий свойство Value (Значение) выполняет ту же роль, что и локальная переменная, но он вынуждает программу переключиться в поток пользовательского интерфейса, чтобы получить данные прямо с лицевой панели. Локальные переменные обращаются к терминалу данных, для этого не требуется изменить поток исполнения и обновить интерфейс. Разделяемая переменная одного процесса (single process shared variable) работает аналогично глобальной переменной с коррекцией ошибок. Локальные и глобальные переменные обычно проще и эффективнее узлов свойств и разделяемых переменных, поэтому, если необходимо задать порядок исполнения, понадобятся структуры последовательности.

Правило 4.31 Структуры последовательности нужны, чтобы задать порядок действий, не связанных потоком данных

Правило 4.32 При необходимости пользуйтесь только плоской структурой последовательности

Пользуйтесь структурами последовательности, только если требуется задать порядок действий, не связанных потоком данных. Так, с помощью структур последовательности можно провести инициализацию, например, с помощью переменных, чтобы это произошло точно в начале приложения. Аналогично можно в самом конце завершить работу LabVIEW. На рис. 4.9 приведен пример инициализации, работы двух параллельных циклов и выключения. Основная цель этого приложения – выполнение длинного процесса сбора данных после команды **Run Test** (Запуск). Процесс сбора данных очень длинный, поэтому пришлось разделить программу на два параллельных цикла – обработки событий пользовательского интерфейса и собственно сбора данных. Ведущий цикл первый: он запускает сбор данных по команде. Параллельные циклы позволяют реагировать на команды пользователя во время сбора данных, эти задачи LabVIEW запускает в различных потоках.

С помощью локальной переменной кластер настройки сенсора (**Sensor Scaling**) инициализируется считанными из файла значениями. Глобальная переменная открывает доступ к логическим данным обоих циклов. Глобальная переменная Acquire запускает сбор данных из цикла событий (Event Loop). Глобальная переменная Stop останавливает цикл сбора данных после остановки цикла событий, и наоборот. В цикле событий значение глобальной переменной записывается в кадре событий **Quit value change** (Выход: значение изменилось). Когда какой-нибудь цикл останавливается, по туннелю цикла передается значение ИСТИНА на узел записи глобальной переменной Stop. На следующей итерации другой цикл считывает это значение и также останавливается. Глобальные переменные требуют меньше усилий, чем разделяемые переменные, события (occurrence), уведомители (notifiers) и очереди (queue). Если нужно передать простые данные между небольшим числом узлов и заведомо в разное время, то глобальные переменные для этой задачи подходят хорошо. Для такой незначительной цели не стоит заниматься разделяемой переменной или другими упомянутыми средствами.

Как видно на рис. 4.9а, вне циклов расположены две связанные проводниками структуры последовательности, каждая из одного кадра. Большая структура слева инициализирует переменные и соединяется двумя проводниками с обоими циклами. Данные по этим проводникам не пойдут до тех пор, пока не исполнятся все внутренние действия структуры, поэтому переменные инициализируются до этой передачи данных. Циклы получают данные из структуры, они не могут начать работу до ее завершения. Таким образом, возникла зависимость данных (**data dependency**). Обратите внимание, что эти данные логических переменных в циклах не используются, проводники обрываются на границе циклов. Тем не менее, этот проводник обеспечивает порядок исполнения, а зависимость данных называется искусственной (**artificial data defendency**).

Структура завершения работы (Shutdown) состоит из структуры последовательности с функцией выхода из LabVIEW. Она получает данные из обоих циклов, чтобы начать работу после их завершения. Также обратите внимание, что через ВП обработки ошибок (General Error Handler) проходит кластер ошибок от цикла сбора данных до структуры завершения работы. Таким образом, обработка ошибок произойдет после завершения сбора данных, но до завершения работы. Блок-диаграммы на рис. 4.9б и 4.9в аналогичны представленным на рис. 4.9а, но

на них используются трехкадровая компактная (stacked) и плоская (flat) структуры последовательности соответственно. Компактная структура менее предпочтительна, потому что поток данных на ней закрыт, для передачи данных между кадрами используются локальные терминалы, а также в каждый момент виден только один кадр структуры. Плоская (она же открытая) структура последовательности (Flat sequence) передает данные между кадрами с помощью туннелей. При использовании обоих структур последовательности не требуется искусственная зависимость данных. По моему мнению, из этих трех примеров наиболее предпочтителен вариант с открытой структурой последовательности (рис. 4.9в).

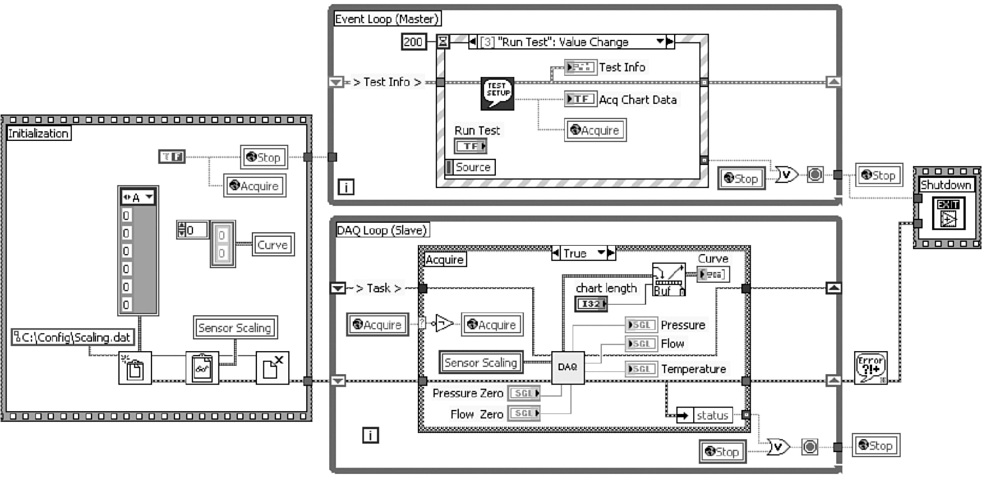


Рис. 4.9а. Последовательность исполнения инициализации, двух циклов и завершения работы определяется с помощью двух однокадровых структур последовательности

4.3.3. Здесь переменные и последовательности не нужны

Во многих приложениях большая часть переменных и структур последовательности не требуется, часто они используются потому, что разработчики не умеют управлять потоком данных. Проще всего научиться управлять потоком данных, приняв правило не использовать переменные и последовательности, кроме исключительных случаев. Получается, что знание потока данных и отсутствие переменных и структур – это синонимы. У примера на рис. 4.9 есть более эффективная реализация. Лишние переменные выделены на рис. 4.10а. Глобальная переменная Acquire запускает сбор данных после события **Run Test Value Change** (Запуск теста: изменилось значение). В цикле сбора данных происходит проверка значения глобальной переменной, сами функции сбора данных расположены в кадре ИСТИНА структуры варианта.

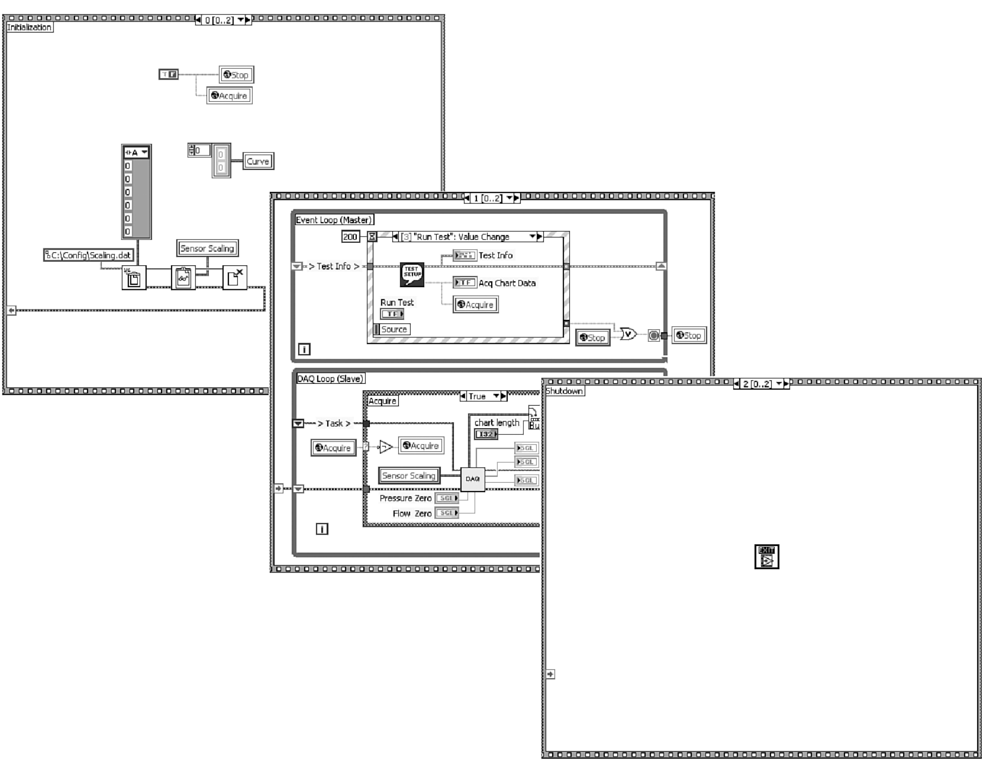


Рис. 4.9б. Для передачи данных между кадрами компактной структуры требуются локальные терминалы

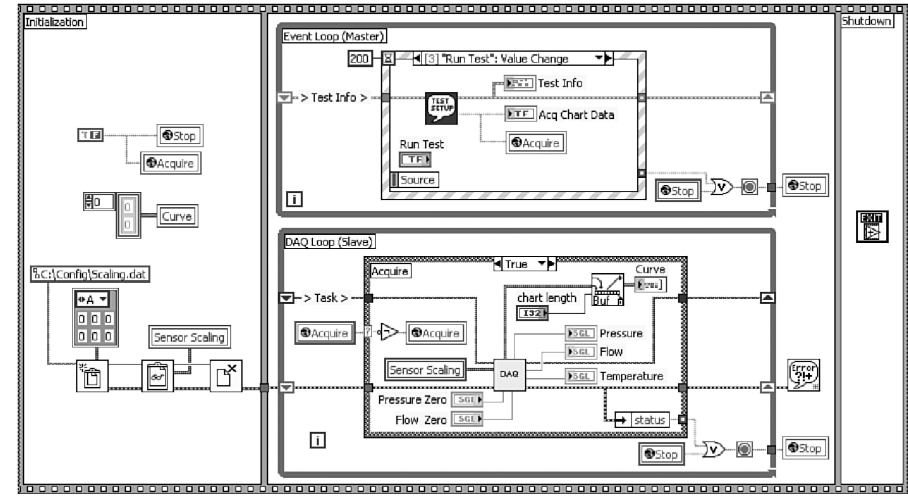


Рис. 4.9в. Поток данных в открытой структуре последовательности соблюдается с помощью туннелей, все кадры видны. Это предпочтительный вариант

Правило 4.33 Избегайте опроса переменных в циклах

Правило 4.34 Избегайте переменных, если задачу можно решить с помощью проводников

Опрос (polling) – это периодическое обращение к ресурсу и ожидание определенного состояния. Никогда не опрашивайте переменные в цикле. На рис. 4.10а цикл сбора данных работает с максимальной скоростью, ожидая изменения переменной. Вместо переменной лучше воспользоваться методами синхронизации данных: событиями (occurrence), уведомителями (notifiers) или очередями (queue). При их использовании ведомый цикл не работает до получения команды. Еще один недостаток варианта на рис. 4.10а – переменные чтения и записи кластера параметров можно заменить проводниками. Переменную можно оставить, если считанные из файла данные нужно передать в элемент управления. Но читать из нее данные в цикле сбора данных совсем не нужно. Наконец, локальная переменная Curve (кривая) устанавливает значение по умолчанию для графика, она не нужна, ведь при открытии ВП значение элементов и так выбирается по умолчанию.

ВП с исправленными недостатками приведен на рис. 4.10б: Глобальная переменная Acquire заменена событиями, переменная Curve не используется, переменные Sensor Scaling заменены проводниками. События позволяют освободить процессор от работы цикла сбора данных до запуска события в ведущем цикле или по истечении 200 мс. Время ожидания (timeout) в цикле событий и для

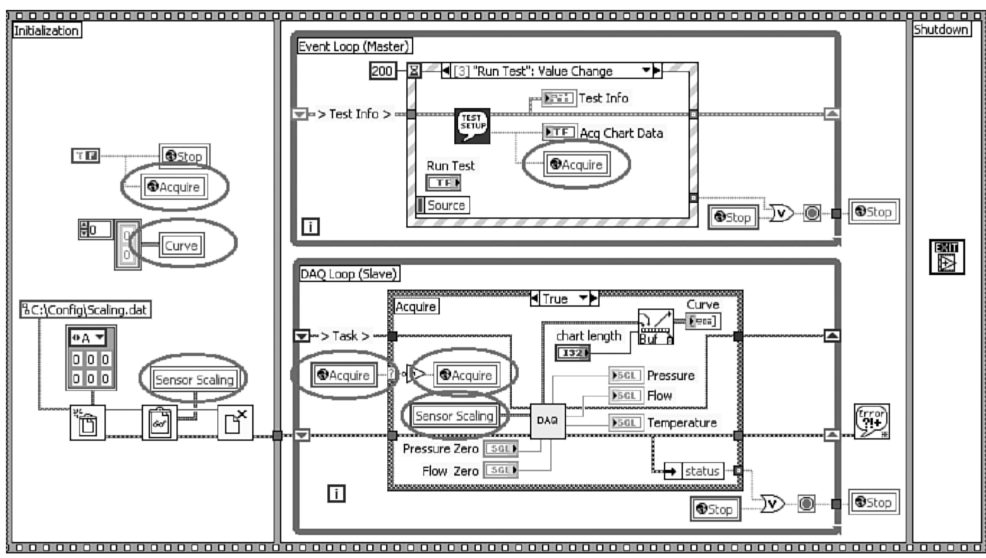


Рис. 4.10а. Избегайте опроса переменных в цикле без задержки (переменная Acquire в цикле сбора данных). Локальные переменные Sensor Scaling можно заменить проводником, а переменная Curve не нужна

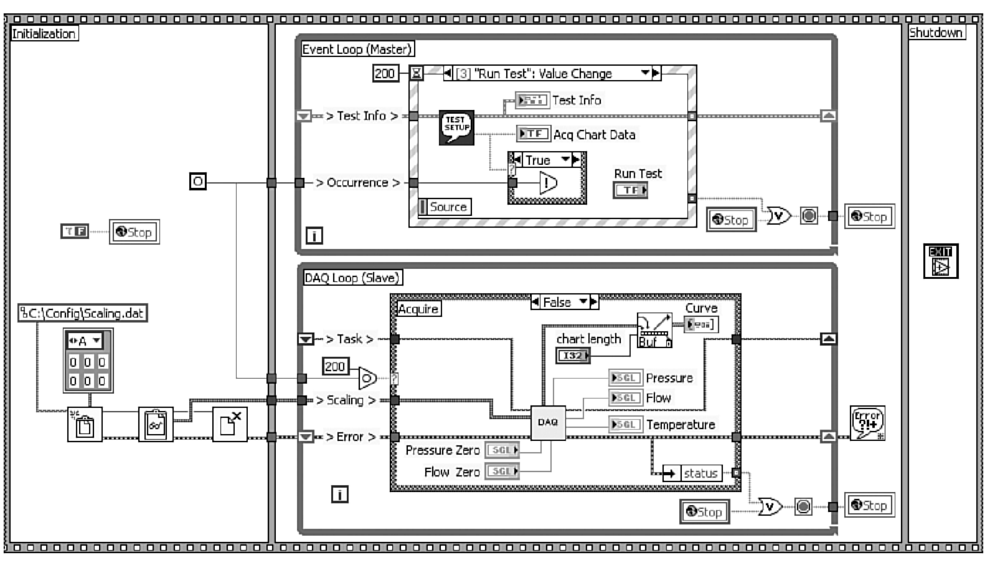


Рис. 4.10б. Глобальная переменная Acquire заменена событиями, переменная Curve не используется, переменные Sensor Scaling заменены проводниками

occurrence равно 200 мс, с таким периодом происходит опрос переменной Stop. Опрос переменной Stop намного эффективнее непрерывного опроса в цикле, но он все равно нарушает Правило 4.33.

При использовании уведомлений (notifier) обеспечивается поток данных и увеличивается эффективность – рис. 4.11. Уведомитель – это структура синхронизации, аналогичная occurrence. Но, в отличие от **Wait on Occurrence** (Ожидание события), функцию **Wait on Notification** (Ожидание уведомления) можно прервать программно. Также с помощью уведомлений можно передавать данные. При событии **Run test** (Запуск теста) вызывается уведомление с логической ИСТИНОЙ для запуска цикла сбора данных. Он продолжает работу с функции **Wait on Notification** и передает полученные данные на терминал выбора, запускается сбор данных. После завершения ведущего цикла вызывается функция **Release Notifier** (Освобождение уведомления). Например, пользователь нажимает кнопку **Quit** (Выход). Происходит событие **Quit Value Change** (Выход: изменение состояния), структура событий исполняется, как показано на рис. 4.11б. Цикл останавливается, вызывается функция освобождения уведомления. Функция **Wait on Notification** возвращает логическую ЛОЖЬ на терминале уведомления (notification) и ошибку на кластере ошибок. Структура выбора исполняет кадр ЛОЖЬ, который очищает ошибку и останавливает цикл. Таким образом, цикл событий полностью управляет циклом сбора данных без переменных. Также обратите внимание, что при возникновении ошибки в одном из циклов останавливаются оба цикла и тоже без использования переменных. Как это сделано: в обоих циклах из кластера выделяется статус ошибки и передается на терминал выхода из цикла.

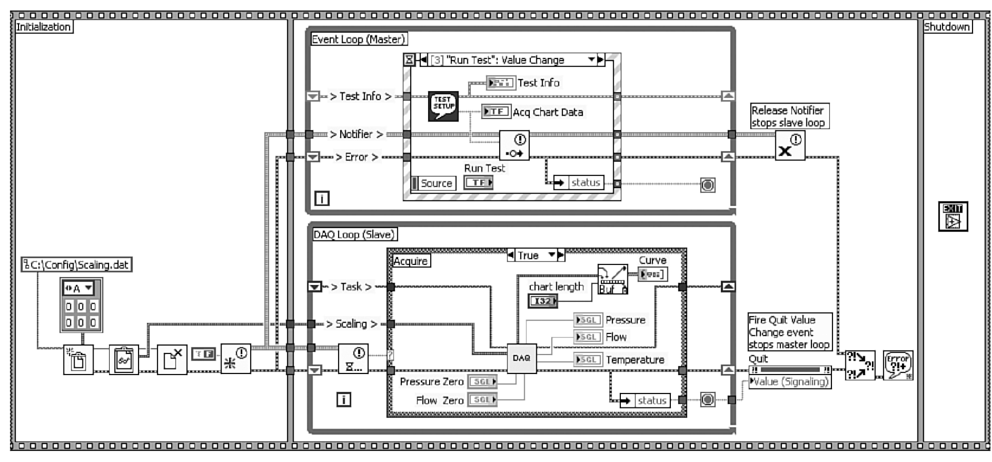


Рис. 4.11а. События заменены уведомлениями, глобальные переменные Stop не используются. Эффективность синхронизации циклов максимальная

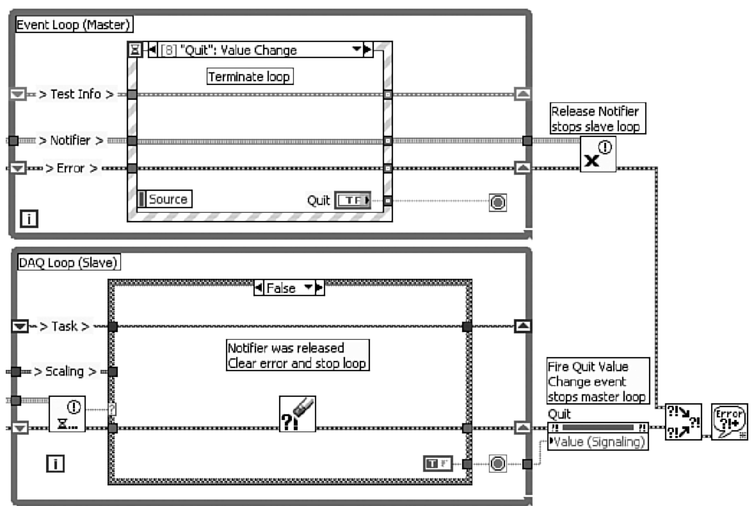


*Рис. 4.11б. Событие **Quit Value Change** останавливает оба цикла. Вызывается функция освобождения уведомления после цикла событий. Функция **Wait on Notification** возвращает логическую ЛОЖЬ на терминале уведомления (notification) и ошибку на кластере ошибок. Структура выбора исполняет кадр ЛОЖЬ, который очищает ошибку и останавливает цикл сбора данных*

Если статус ИСТИНА в цикле событий, цикл останавливается и вызывает функцию освобождения уведомления, как описано выше, цикл сбора данных останавливается. Если произошла ошибка в цикле сбора данных, он останавливается, и значение ИСТИНА передается на узел свойств. Свойство **Quit Value Signaling** (Выход,

значение с уведомлением) вызывает событие **Quit Value Change** (Выход: значение изменилось), которое останавливает цикл событий. Поэтому локальные переменные не нужны совсем, эффективность синхронизации циклов максимальная.

4.3.4. Оптимизируем поток данных

Как мы уже поняли, профессиональная организация потока данных в LabVIEW неотрывно связана с отказом от локальных и глобальных переменных и структур последовательности. Чуть выше мы рассмотрели редкие исключения из этого правила и гораздо более частые ошибки. В этом разделе мы обратимся к средствам, которые помогают решить задачу в спорных случаях: это сдвиговые регистры и структура выбора в цикле.

Правило 4.35 Сдвиговые регистры предпочтительней локальных и глобальных переменных

Правило 4.36 Располагайте большинство сдвиговых регистров в верхней части цикла

Правило 4.37 Добавляйте метки-пояснения к левым терминалам сдвиговых регистров

Сдвиговые регистры (**Shift registers**) – это терминалы на границе структур циклов, которые позволяют получить данные с предыдущих итераций. По сути и функционально они эквивалентны проводникам, протянутым с конца предыдущей до начала следующей итерации. Когда область использования данных ограничена циклом, они позволяют заменить локальные и глобальные переменные. В отличие от них, при использовании сдвиговых регистров не появляется дополнительной копии данных при чтении значений, кроме того, сдвиговые регистры гораздо эффективнее.

Чтобы избежать путаницы проводов, располагайте большинство сдвиговых регистров в верхней части цикла. В этом случае получается строгий поток данных с минимальным количеством пересечений проводников. Оставьте только у каждого проводника достаточно места для свободной метки рядом с левым терминалом. Исключения из этого правила – кластеры ошибок и данные выбора для Case. Кластеры ошибок обычно проходят снизу, а терминал выбора в структуре – в середине. Поэтому эти типы проводников обычно не включены в сеть проводников с другими данными.

Правило 4.38 Структура выбора с циклом предпочтительнее последовательности

Часто используется структура выбора внутри цикла (**Looped Case structure**). Если в разных кадрах структуры разместить различные участки кода, она будет

подобна структуре последовательности с теми же участками. Отличие между ними будет заключаться в том, что порядок исполнения последовательности фиксирован и определяется расположением участков кода, а в цикле может быть выбран программно. Для передачи данных между итерациями можно пользоваться сдвиговыми регистрами. Они предпочтительнее локальных терминалов структуры последовательности, потому что лучше соответствуют правильному потоку данных: все данные двигаются слева направо.

На рис. 4.12 предложено 4 варианта теста из 12 последовательных ВП. У каждого ВП есть общие каналы данных, включая задачу DAQmx и кластеры коэффициентов масштабирования сенсора, пределов теста и ошибок. Каждый кластер добавляет строку в таблицу результатов теста **Test results** для отчета. Эти данные обновляются сразу после завершения теста. Также пределы для каждого теста каждый раз новые и вычисляются на предыдущем шаге.

На рис. 4.12а используется компактная структура последовательности, которая нарушает правила 4.27 и 4.32. Кластер ошибок передается между кадрами с помощью локальных терминалов на внутренней границе структуры. Так как терминалы могут быть использованы для записи только один раз, для передачи кластера через 12 кадров требуется 11 терминалов. При этом нарушаются различные правила, и получается запутанная сеть проводов. На большинстве кадров проводники пересекаются или передают данные справа налево при чтении данных с правой границы или записи на левую. Также в этом случае используется 48 локальных переменных: по две на каждый кадр для обновления результатов теста и изменения пределов измерений.

На рис. 4.12б используется структура выбора в цикле с фиксированным числом итераций. Участок кода в каждом кадре структуры эквивалентен соответствующему участку в структуре последовательности. Чтобы заменить 48 локальных переменных и 11 локальных терминалов, требуется всего 5 сдвиговых регистров. Сеть проводов логичная, без пересечений или нарушения направления передачи данных. На рис. 4.12в структура выбора расположена в цикле По условию. Единственное отличие от предыдущего варианта следующее: если во время теста происходит ошибка, все измерения прекращаются.

При использовании цикла По условию можно не ограничивать порядок следования кадров. Реализация теста с помощью гибкого указания следования данных

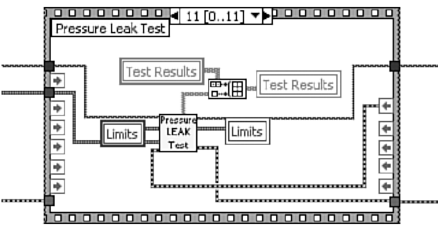


Рис. 4.12а. Тест реализован с помощью 12 кадров структуры последовательности. Используется 48 локальных переменных и 11 локальных терминалов, поток данных нарушен

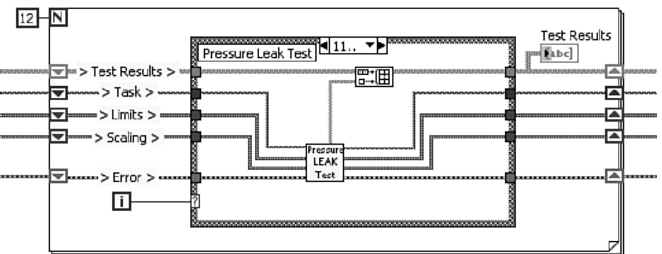


Рис. 4.12б. Тест реализован с помощью структуры выбора в цикле с фиксированным числом итераций. Локальные терминалы и переменные заменены на 5 сдвиговых регистров

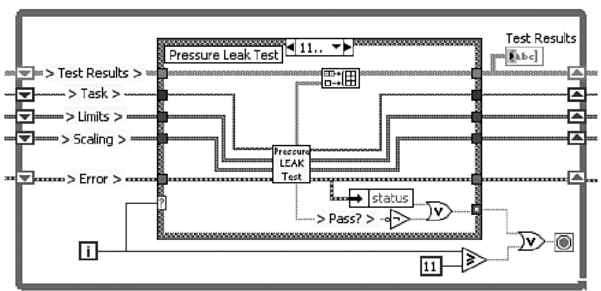


Рис. 4.12в. Тест реализован с помощью структуры выбора в цикле По условию с прекращением работы в случае ошибки

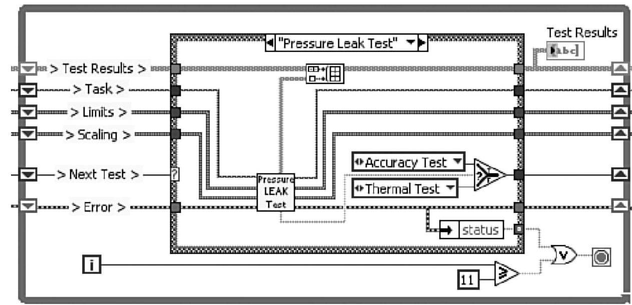


Рис. 4.12г. Гибкая последовательность исполнения теста в конечном автомате. В нумерованном списке перечислены все стадии. Последовательность их исполнения формируется динамически

в конечном автомате приведена на рис. 4.12г. Классический конечный автомат, который подробно обсуждается в главе 8, состоит из структуры выбора в цикле По условию и нумерованного списка возможных вариантов. Вместо номеров в этом случае используются пояснительные метки, которые помогают избежать

комментариев в каждом кадре. Также в конечном автомате следующий шаг определяется результатами предыдущего, и последовательность исполнения формируется динамически. Вариант на рис. 4.12г отличается наибольшей гибкостью среди рассмотренных.

Давайте теперь воспользуемся последним шаблоном в более крупном приложении. Изменим ВП тестирования с рис. 4.11 так, чтобы в цикле измерений исполнялась последовательность тестов. На первый взгляд, нужно поместить рис. 4.12г внутрь структуры выбора, но тогда получится 4 уровня вложенности и нарушится Правило 4.28. Однако мы можем избавиться от цикла измерений, заменив его, например, шаблоном конечного автомата с очередями (**Queued State Machine**). В этом шаблоне (рис. 4.13а) для синхронизации циклов используются очереди, а не уведомители, переменная выбора – не логическая, а нумерованный список, в структуре варианта перечислены возможные стадии теста. Очередь похожа на буфер FIFO, который определяет порядок проведения тестов. Как видно на рис. 4.13а, в ведущем цикле 12 раз вызывается функция добавления в очередь (Enqueue Element). Обратите внимание на рис. 4.13б: функции чтения параметров из файла расположены в кадре **Initialize** (Инициализация) в ведомом цикле. Этот кадр исполняется первым: это обеспечивает функция Enqueue Element в начале программы. Поэтому конечный автомат не только проводит измерения, но и может выполнять множество других действий, инициализация – одно из них. Этот вариант аккуратнее и более функционален, чем на основе уведомлений (см. рис. 4.11).

На рис. 4.13в приведены действия программы для прекращения работы по команде пользователя или при ошибке. Если пользователь нажимает кнопку **Quit**

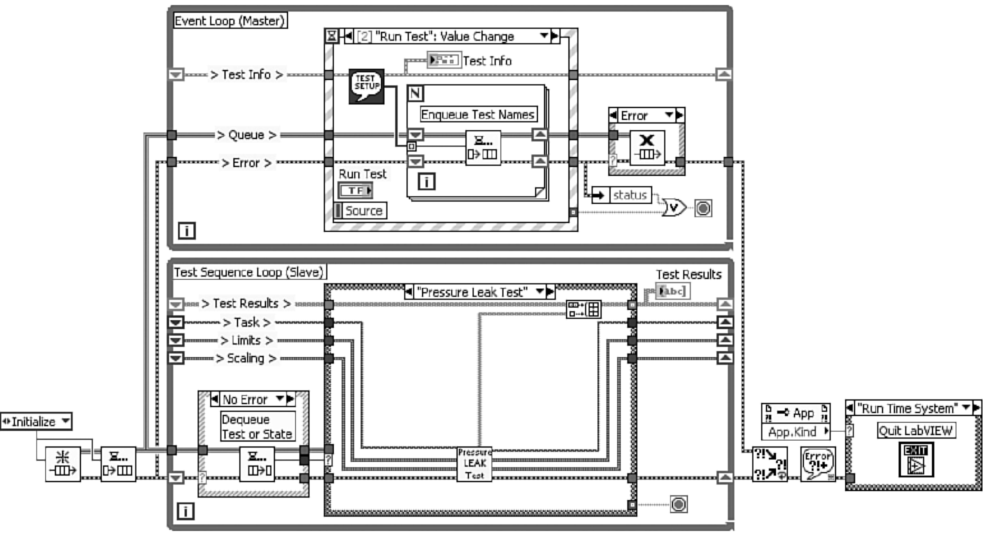


Рис. 4.13а. Объединение примеров с рис. 4.11 и 4.12: ведущий цикл запускает процедуру из 12 измерений с помощью конечного автомата с очередью

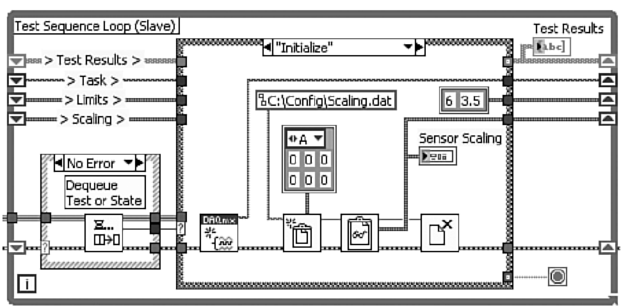


Рис. 4.13б. Кроме проведения измерений, конечный автомат выполняет инициализацию. В кадре *Initialize* из файла считываются масштабные параметры

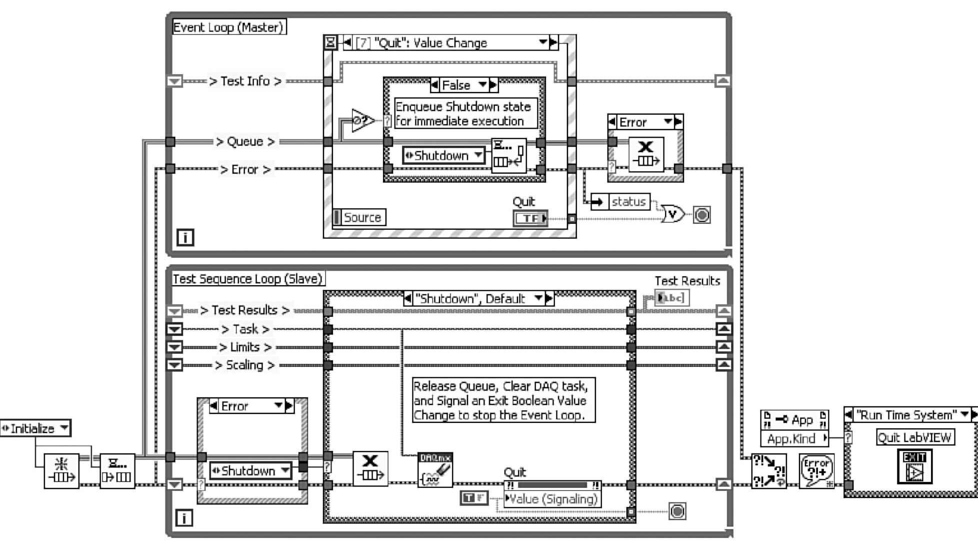


Рис. 4.13в. Если в каком-нибудь цикле происходит ошибка или пользователь нажимает кнопку выхода, оба цикла останавливают друг друга

(Выход), в кадре структуры событий **Quit Value Change** в начало очереди добавляется действие **Shutdown** с помощью функции Enqueue Element Opposite End. Это действие выполнится сразу после завершения текущего шага измерений. Если ошибка происходит в ведущем цикле, очередь разрушается в соответствующем кадре структуры выбора; функция получения элементов из очереди Dequeue Element возвращает ошибку; исполняется вариант, установленный по умолчанию (завершение работы). Если ошибка происходит в цикле измерений, в конечном автомате исполняется кадр завершения работы: очищается задача сбора данных, генерируется нажатие на кнопку **Quit** (Выход) (событие **Quit Value Change**). Шаблон конечного автомата с очередью подробно описан в главе 8.

4.4. Примеры

В этом разделе приведены разные примеры: как хорошие, так и плохие. Начнем с самого плохого и дойдем до идеала. Плохие примеры показывают, как много правил, которые можно нарушить, мы уже рассмотрели.

4.4.1. ВПП из участка кода

Как мы говорили в главе 3 «Стиль лицевой панели», инструмент создания ВПП из участка кода – это самый хороший способ разрушить стиль программы. Для этого нужно просто выбрать участок блок-диаграммы и выбрать пункт меню **Edit** ⇒ **Create SubVI** (Редактировать ⇒ Создать ВПП). Самое плохое, что вы можете сделать, – не отредактировать получившийся ВПП. Необходимо исправить метки и расположение терминалов, сеть проводов, назначение разъемов, иконку и описание. Иногда кажется, что на блок-диаграмме получившихся ВПП взорвалась бомба, они никогда не бывают выполнены с хорошим стилем.

На рис. 4.14а показана блок-диаграмма ВПП, созданного из участка кода (SubVI from Selection VI), который мы уже видели в главе 3. Несколько признаков с головой выдают метод создания ВПП: проводники неровные, элементы расположены тесно, их метки бессмысленные. Например, элементы управления названы **task ID out** и **error out**. Также у одного из индикаторов стандартная метка **Numeric** (Числовой). Как ни странно, у этого ВПП нормальная иконка, описание

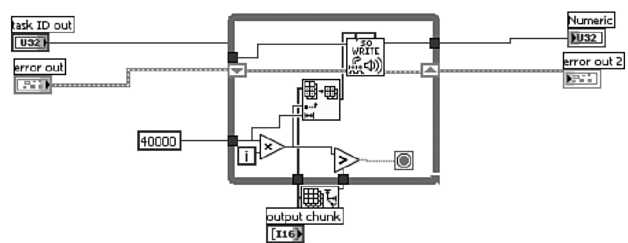


Рис. 4.14а. Эта блок-диаграмма получилась из участка кода

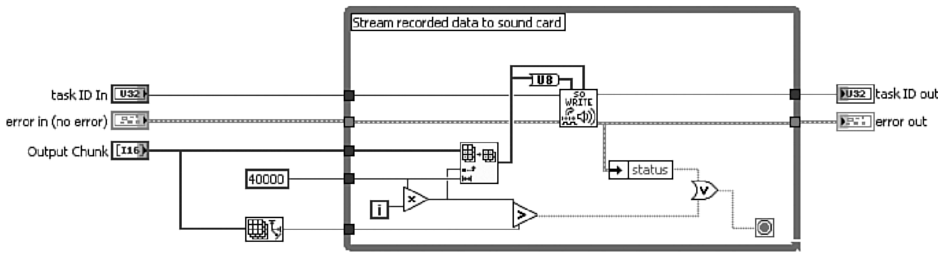


Рис. 4.14б. Эта блок-диаграмма переработана: изменены метки, расположение элементов и сеть проводов

и расположение разъемов; наверное, над ним все-таки поработали. Почистим проводники и метки терминалов – результат приведен на рис. 4.14б. На рис. 4.14в приведена обновленная версия этого ВПП, использующая функции генерации звука LabVIEW 8.0.

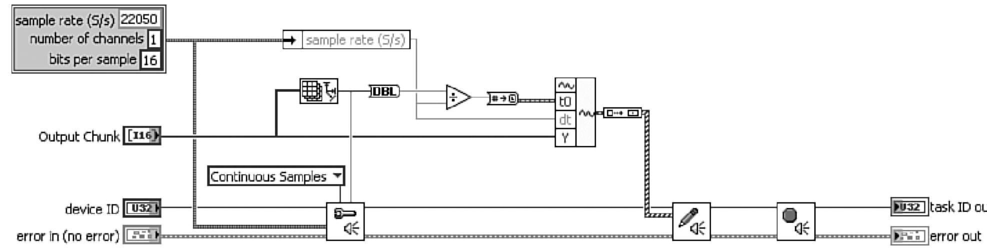


Рис. 4.14в. На этом ВПП используется тип данных: осциллограмма (waveform) и ВПП вывода звука LabVIEW 8.0

Расположение: Диаграмма (рис. 4.14а) маленькая, объекты расположены тесно

Модульность: Не вычислялась: слишком мало проводов. Но сам ВПП – попытка пользователя повысить модульность приложения

Сеть проводников: Проводники изломаны – признак создания ВПП из участка блок-диаграммы. Один из проводов проходит по границе цикла

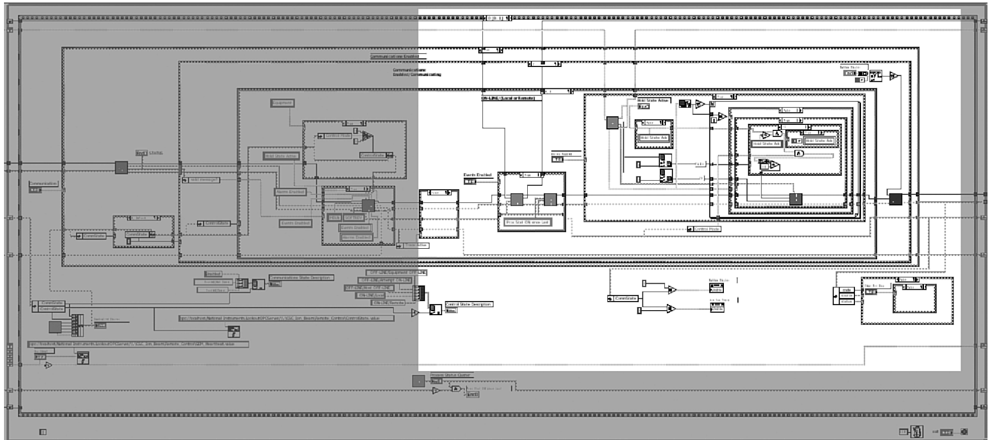
Поток данных: Данные двигаются, в том числе и по вертикали: два туннеля расположены на нижней границе цикла. При преобразовании данных звука из I16 в U8 появляется точка приведения типов

4.4.2. ВП Excessively Nested

Блок-диаграмма ВП Excessively Nested (Излишне вложенный) приведена на рис. 4.15. Она слишком большого размера. В верхней части показано окно навигации – это единственный способ ориентироваться во всей диаграмме. Нижняя часть рисунка – участок диаграммы с многоуровневой структурой, при разрешении 1280×1024 он занимает меньше половины экрана. При навигации приходится не столько прокручивать экран, сколько продираться сквозь вложенные кадры и варианты структур. На приведенном участке уровень вложенности достигает 11. Даже профессиональным разработчикам (и игрокам в покер) трудно понять все варианты работы программы. Избегайте таких больших диаграмм и высокого уровня вложенности. Приведенные в главе 8 стандартные шаблоны помогут вам избежать таких программ.

Расположение: Диаграмма слишком большая, излишне вложенная и неуклюжая

Модульность: Всего 40 пользовательских ВП на 2040 узлов, индекс модульности 1,9. Модульность отсутствует



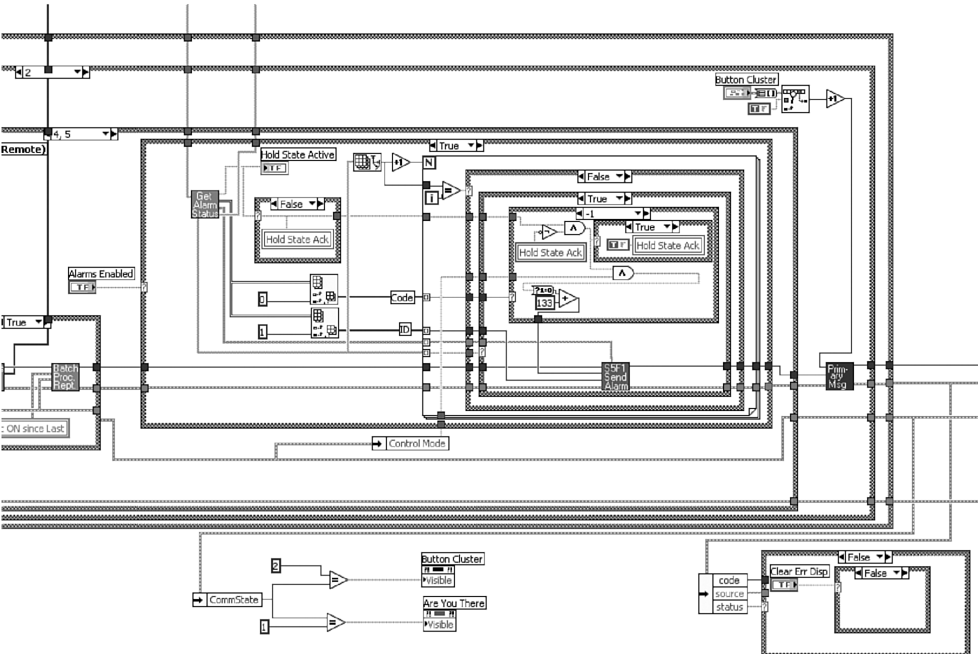


Рис. 4.15. Чтобы ориентироваться в этой диаграмме, требуется окно навигации (сверху). На нижней части рисунка приведено 9 из 11 уровней вложенности диаграммы.

Сеть проводников: Слишком много длинных проводников с ненужными изгибами

Поток данных: Вложенность нарушает нормальный поток данных. Есть нарушения направления: справа налево и по вертикали в структуры

4.4.3. ВП Haphazard

Диаграмма ВП Haphazard (Бессистемный) приведена на рис. 4.16: поток данных зачастую случаен, туннелей на нижней и верхней границах больше, чем на левой. Много лишних изгибов и петель на проводниках. Данные двигаются слева направо, справа налево, вверх, вниз и по кругу. Также перед циклом инициализируется несколько локальных переменных, но при этом параллельно считываются данные с соответствующих терминалов. Зависимости данных между инициализацией и чтением нет, поэтому порядок этих операций не определен. Таким образом, инициализация проводится неправильно. В этой ситуации требуется структура последовательности: либо из одного кадра перед циклом с искусственной зависимостью данных, либо из нескольких кадров, как мы уже рассмотрели на рис. 4.9а и 4.9в. Также используются нестандартные иконки Правила оформления иконок обсуждаются в главе 5.

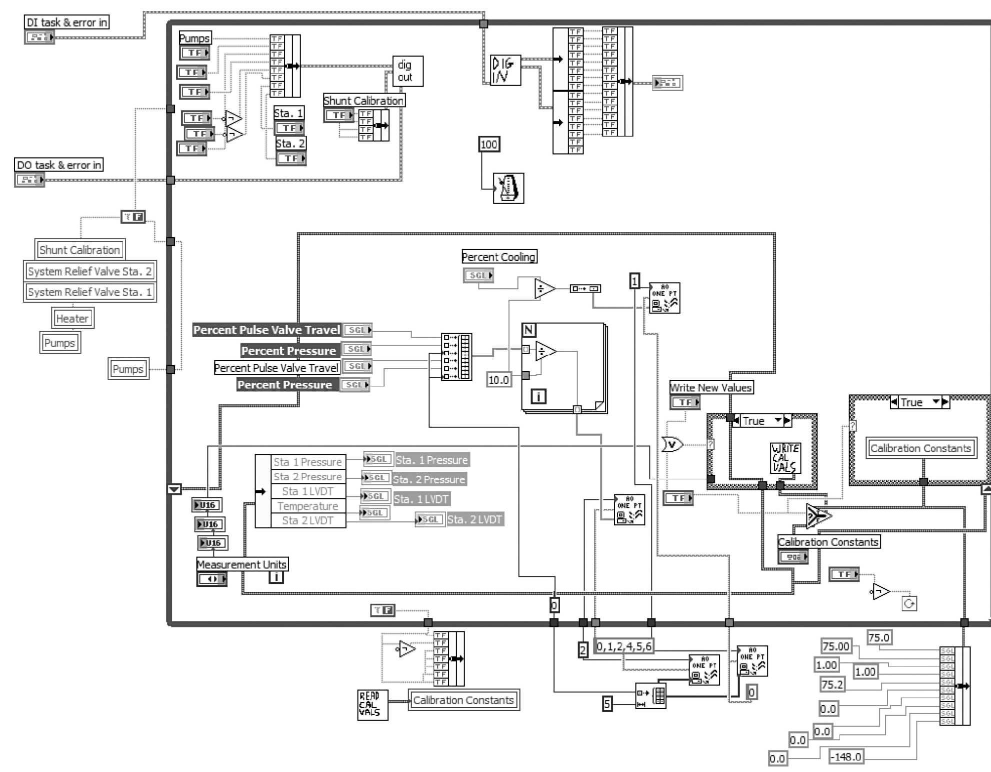


Рис. 4.16. На диаграмме много лишних изгибов проводников, бессистемный поток данных и нестандартные иконки. Порядок исполнения операций чтения и записи с локальными переменными не определен

Расположение: Размер диаграммы соответствующий, плотность элементов средняя и низкая

Модульность: На диаграмме 13 ВПП и 396 узлов, индекс модульности 3,3. Участок кода вне цикла лучше заменить одним инициализирующим ВПП

Сеть проводников: Лишние изгибы и петли, много вертикальных туннелей, также есть излишне длинные проводники

Поток данных: Данные двигаются во всех направлениях. Порядок инициализации данных не определен. Есть приведения типов

4.4.4. ВП Right to Left

ВП Right to Left (Справа налево) – это приложение, в котором до определенного события регистрируются объем и давление, после события (рис. 4.17) последние

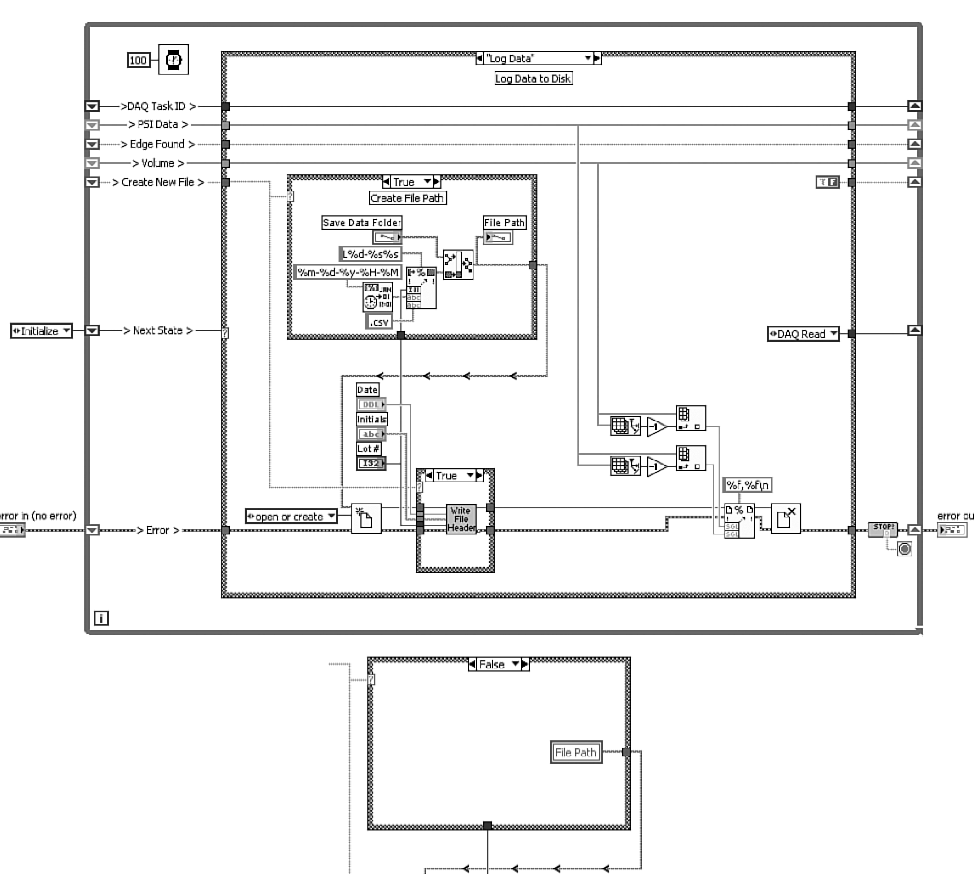


Рис. 4.17. ВП Right to Left – это ВП верхнего уровня с группой низкоуровневых функций, формирующих путь к файлу, локальной переменной, вертикальным туннелем и потоком данных справа налево

массивы данных записываются в текстовый файл. Поток данных организован эффективно, с достаточными комментариями. Однако по одному из проводников в центре основной структуры выбора данные передаются справа налево. Метки на проводнике подчеркивают направление потока данных. На первый взгляд, такая организация потока данных не испортит хороший стиль приложения, один проводник справа налево никому не навредит. Однако это все-таки нарушение правила потока данных, и его лучше избежать.

При более подробном рассмотрении оказывается, что этот проводник передает путь к файлу, который строится с помощью низкоуровневых функций внутри структуры событий. В соответствии с Правилом 4.7 обособленную группу низкоуровневых функций лучше объединить в ВПП. Также в кадре ЛОЖЬ структуры выбора путь к файлу считывается из локальной переменной. В данном случае можно заменить переменную сдвиговым регистром. Наконец, из терминала номера серии «Lot #» данные передаются в структуру варианта по вертикальному туннелю, нарушая Правило 4.13. Таким образом, один проводник с неправильным направлением данных вызывает сразу 4 нарушения.

Расположение: Это стандартный конечный автомат с низкой плотностью элементов

Модульность: Приложение простое, состоит из 6 ВПП и 173 узлов, индекс модульности 3,5. В кадре записи данных группу низкоуровневых функций можно заменить на ВПП в соответствии с Правилом 4.7

Сеть проводников: Проводники прямые, пересечения необходимы

Поток данных: Большая часть данных передается слева направо, за исключением одного проводника. Для передачи данных между итерациями цикла используются сдвиговые регистры. Локальную переменную с расположением файла можно заменить еще одним регистром

4.4.5. ВП Left to Right

На рис. 4.18 предыдущий ВП переработан: Путь к файлу формируется в ВПП, между итерациями хранится в сдвиговом регистре. На этой диаграмме больше не требуется локальной переменной, вертикального туннеля и нарушенного направления потока данных. Также размеры структур и промежутки между ними уменьшены, плотность блок-диаграммы повысилась. Этот пример демонстрирует взаимосвязь правил стиля. Чем точнее вы придерживаетесь базовых правил, тем легче вам поддерживать стиль всего приложения.

Расположение: Классический конечный автомат с высокой плотностью объектов

Модульность: Благодаря дополнительному ВПП индекс модульности возрос до 4.0

Сеть проводников: Проводники прямые, все пересечения необходимы

Поток данных: Все данные передаются слева направо. В дополнительном сдвиговом регистре хранится путь к файлу

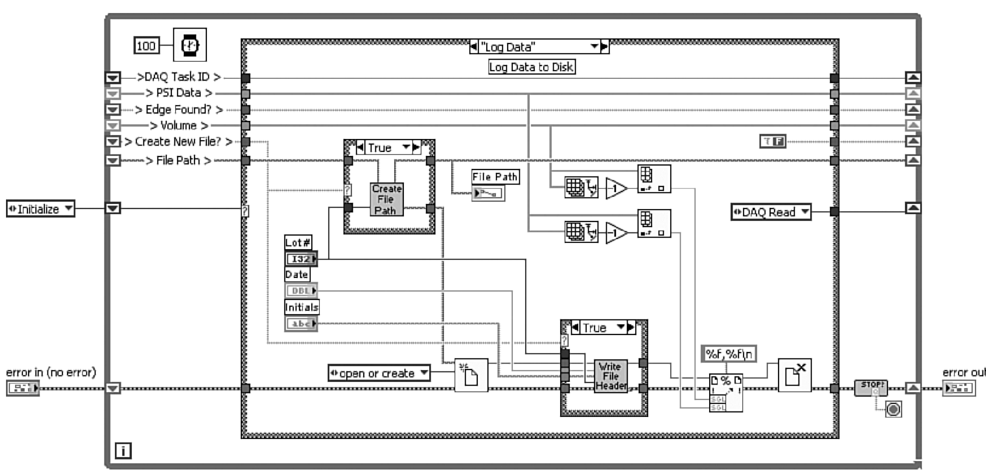


Рис. 4.18. ВП Left to Right функционально совпадает с ВП Right to Left, но не нарушает правил стиля

4.4.6. ВП Centrifuge DAQ

Блок-диаграмма ВП Centrifuge DAQ приведена на рис. 4.19. Она очень аккуратная, понятная и с хорошими пояснениями. Основные данные считываются из сдвиговых регистров и передаются по туннелям в основную структуру варианта, не нарушая поток данных. У каждого проводника есть пояснение перед структурой, внутри нее проводники расположены более тесно. В нижней части блок-диаграммы массив ссылок на управляющие элементы передается в ВПП для управления их свойствами. Шаблон конечного автомата с очередью легко расширяется: нужно добавить кадры в структуру варианта и дополнительные элементы в нумерованный список. Более подробно это приложение обсуждается в главах 6 и 8.

Расположение: Фактически диаграмма состоит из двух параллельных циклов, на мониторе виден только один, он приведен и на рисунке. Этот участок представляет собой конечный автомат с очередью

Модульность: Используется 113 ВПП и 2887 узлов, индекс модульности 3,9. Для экономии места свойства элементов лицевой панели изменяются в ВПП

Сеть проводников: Данные объединены в кластеры. Тесное размещение проводников внутри структуры обеспечивает дополнительное место

Поток данных: Данные двигаются строго слева направо. Для передачи данных между параллельными циклами используются очереди. При необходимости применяются сдвиговые регистры. В приложении используется 50 переменных, в основном для инициализации значений элементов

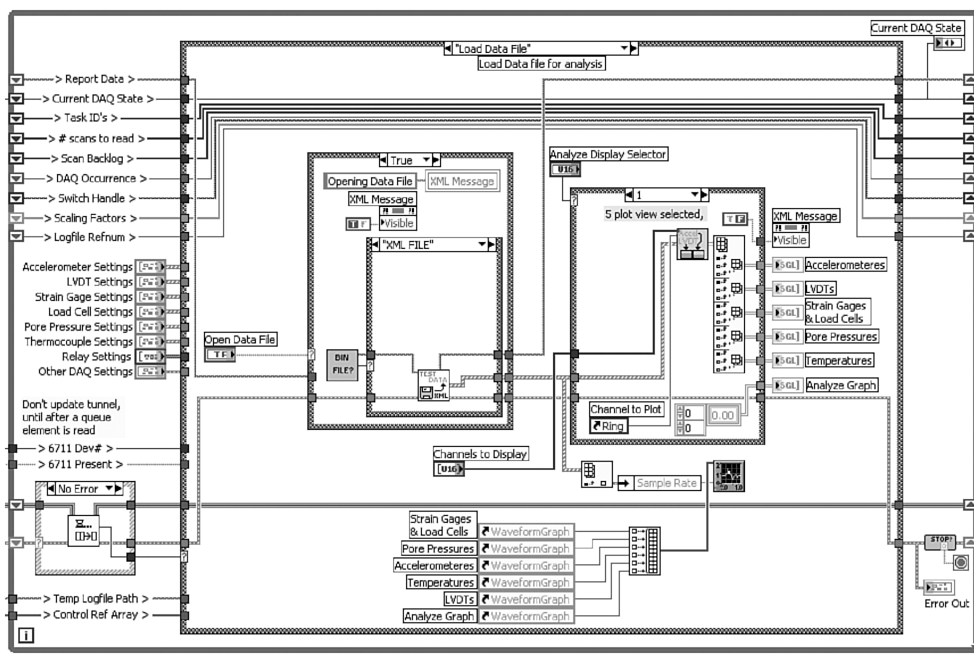


Рис. 4.19. ВП Centrifuge DAQ сделан на высоком уровне: используются ссылки на элементы и шаблон конечного автомата с очередью

4.4.7. ВП Screw Inspection

На рис. 4.20а приведена блок-диаграмма ВП Screw Inspection (Анализ винтов). Диаграмма чистая, с многоуровневым потоком данных, высокой модульностью и удобными иконками ВПП. На первый взгляд, правила расположения, сети проводников и потока данных не нарушены. Для передачи данных используются сдвиговые регистры, а не переменные. Объекты и проводники расположены не слишком тесно. Однако при более пристальном рассмотрении видны многочисленные нарушения, они выделены на рис. 4.20б. К ним относятся лишние изгибы проводников и петли (Правило 4.11); перекрывающиеся объекты (Правило 4.15); длинные проводники без меток (Правило 4.18), включая и проводники из сдвиговых регистров (Правило 4.37); поток данных справа налево (Правило 4.22); обрыв кластера ошибок (Правило 4.23) и точки приведения типов (Правило 4.24). В левой части излишняя мешанина меток и констант. Константы не обязательны, потому что они совпадают с данными по умолчанию для соответствующих сдвиговых регистров при первой загрузке в память. Обработка ошибок практически не организована. Все перечисленные недостатки исправлены на рис. 4.20в. Обсуждение передачи кластера ошибок отложено до главы 7 «Обработка ошибок».

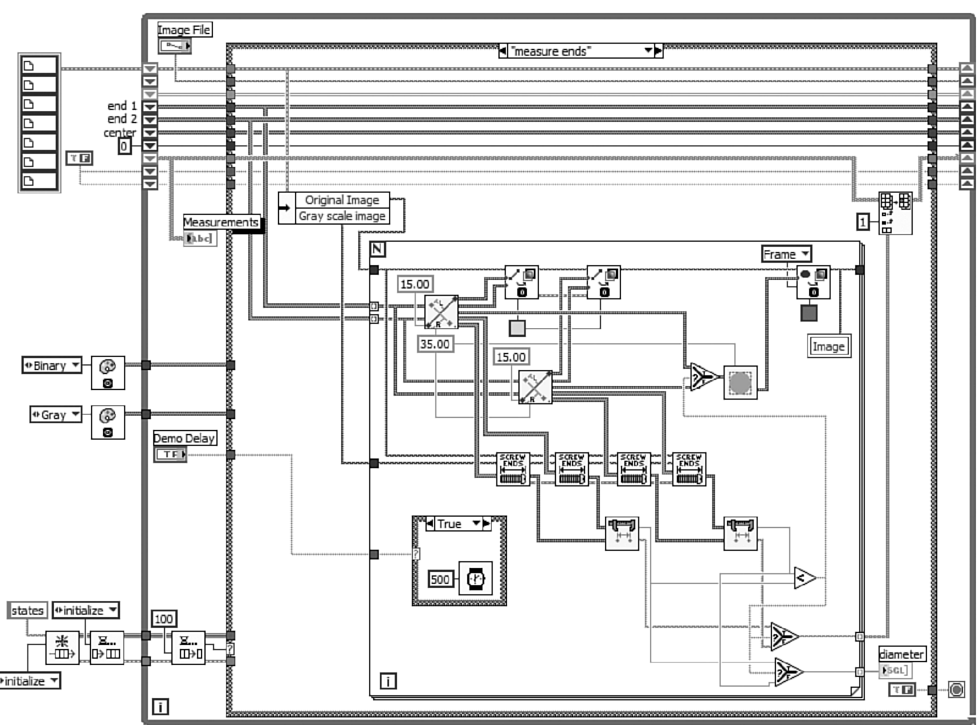


Рис. 4.20а. ВП Screw Inspection достаточно аккуратный, поток данных организован на разных уровнях, выделяются специализированные иконки. На первый взгляд, правила расположения, сети проводников и потока данных не нарушены

Расположение: Плотность блок-диаграмм на рис. 4.20а и 4.20б разумная, вся блок-диаграмма помещается на 1 экран. Есть одна накладывающаяся метка

Модульность: ВП состоит из 7 ВПП и 292 узлов, индекс модульности 2,4

Сеть проводников: В общем, сеть аккуратная. Данные объединены в кластеры. Однако встречаются лишние изгибы и петли, перекрывающиеся объекты и проводники без меток, включая сдвиговые регистры

Поток данных: Данные передаются между уровнями: слева направо и сверху вниз. Сдвиговые регистры помогают избавиться от переменных. Однако обработка ошибок не организована, и есть отдельные проводники с нарушенным направлением потока данных

4.4.8. ВП Optical Filter Test

Вернемся к задаче тестирования оптического фильтра. На рис. 4.21 приведен ВП верхнего уровня, запускающий стадии тестирования в конечном автомате с очередями. Набор тестов (см. рис. 4.6а) перенесен в кадр **Normal Sweep** (Нормальный режим) конечного автомата. Порядок исполнения определяется программно

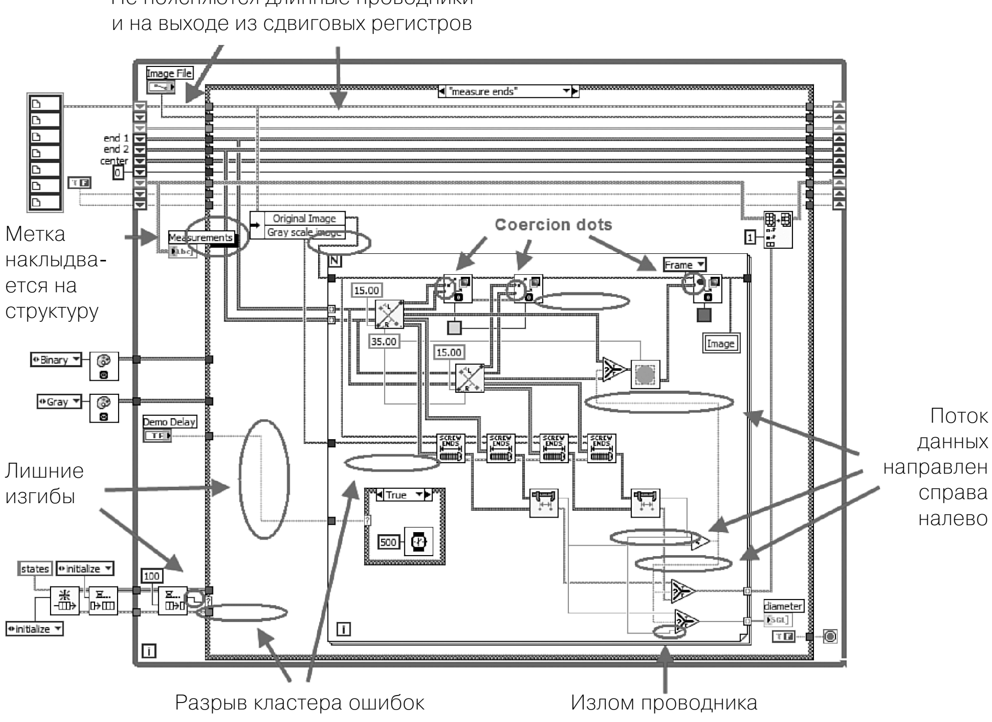


Рис. 4.20б. При пристальном рассмотрении можно найти многочисленные нарушения стиля

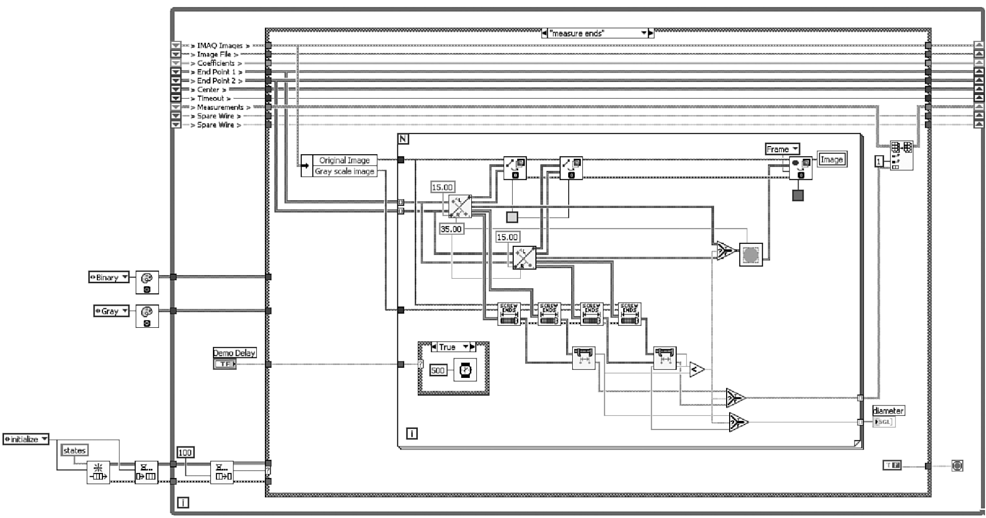


Рис. 4.20в. ВП Screw Inspection с многочисленными исправлениями. Передача ошибок отложена до главы 7

с помощью очереди. Данные передаются с помощью сдвиговых регистров. Однако они расположены случайно (Правило 4.36) и не содержат поясняющих меток (Правило 4.37). Поэтому данные передаются в середине структуры, разделяя ее на несколько участков. Также возникают различные нарушения потока данных и сети проводников: пересечение проводников (Правило 4.15), поток справа налево (Правило 4.22). А именно кластер ошибок по пути к функции выделения элемента из очереди (**Dequeue Element**) пересекает проводники данных из сдвиговых регистров. Также есть группа низкоуровневых функций для расчета шагов по длине волны, которая передает результат в два ВП драйверов. И наконец, удоб-

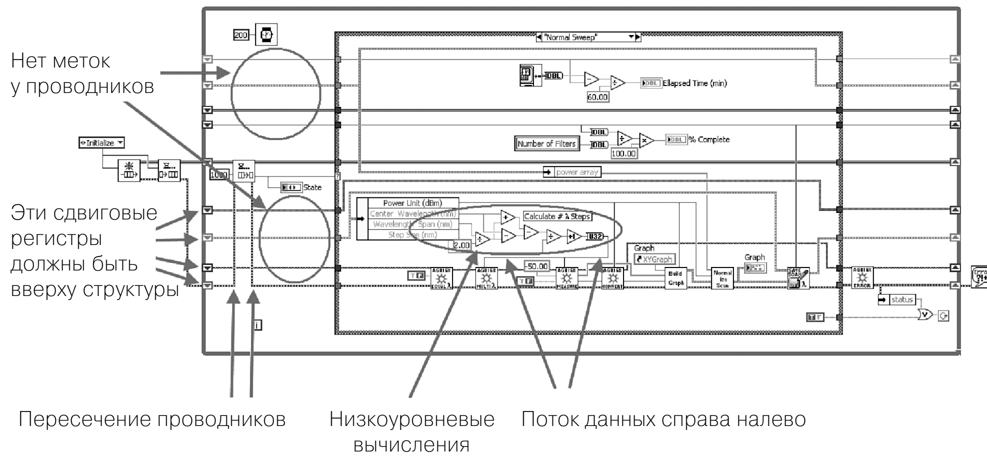


Рис. 4.21а. Приложение тестирования оптического фильтра основано на шаблоне конечного автомата с очередями. Сдвиговые регистры не сгруппированы, их проводники без меток, есть участки с нарушенным направлением потока данных и с группами функций низкого уровня

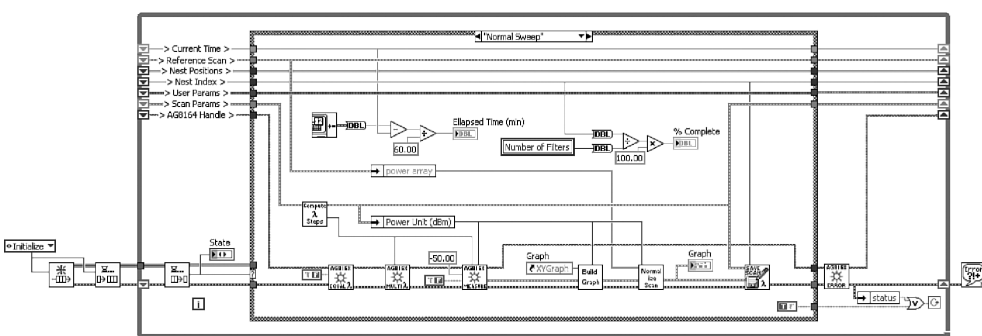


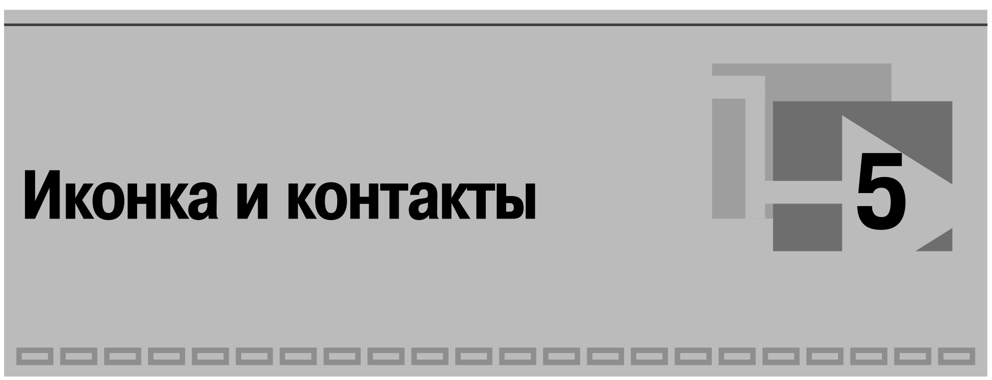
Рис. 4.21б. Сдвиговые регистры сгруппированы в верхней части, добавлены метки к проводникам, уменьшено число пересечений, размер массива вычисляется в ВПП, массивы длины волны и мощности объединены в кластер, отсутствует поток данных справа налево

ство работы с ВП портится из-за нарушенного потока данных, сети проводников и нескольких низкоуровневых функций на ВП высокого уровня (Правило 4.7).

На рис. 4.21б к сдвиговым регистрам добавлены метки, и они сгруппированы в верхней части структуры для лучшей организации потока данных. Функции работы с очередью размещены в нижней части блок-диаграммы, что позволяет избавиться от пересечений с кластером ошибок. Вместо группы функций расчета шага по длине волны используется ВПП, таким образом, решается проблема с неправильным направлением потока данных. Также длина волны и мощность объединены в кластер. Благодаря этим исправлениям диаграмма рис. 4.21б стала более удобной и организованной, чем на рис. 4.21а.

Ссылки

На зоне разработчиков NI Developer Zone можно загрузить драйверы более чем к 5000 приборов и примеры с открытым кодом. Адрес: www.ni.com/devzone/.



Иконка и контакты

Представьте себе внешний вид и методы создания исходного кода в текстовой среде разработки. Тысячи линий черного текста с рассеянными по нему цветными ключевыми словами и комментариями. Мириады символов, названий функций, модификаторов и чисел в шестнадцатеричном коде. Чтобы проследить поток данных, приходится постоянно запускать поиск нужной переменной. Исходный код – это текст, иногда цветной и с разными символами. Чтобы его редактировать, нужен, разумеется, текстовый редактор. Не кажется ли вам, что чего-то не хватает? Заходим в LabVIEW. Цветные и понятные блок-диаграммы, вместо функций – иконки; чтобы увидеть поток данных, нужно просто посмотреть. Добро пожаловать в LabVIEW из скучного мира разработчиков ПО.

Иконки – это отличительная особенность графического программирования. Их интересно создавать и с ними приятнее работать, чем со строками текста. Лучше один раз увидеть, чем сто раз прочитать, – одна иконка может заменить сотни строк кода. Редко когда инженерам, программистам и ученым выпадает возможность порисовать аккуратные небольшие картинки, LabVIEW – исключение.

Иконка и соединительная панель – это итог создания ВП, только после этой стадии ВП можно назвать подпрограммой, ВПП. Иконка – это графическое описание назначения ВПП. На соединительной панели расположены контакты, обеспечивающие поток данных от проводников вызывающего ВП к терминалам этого ВПП и далее к его функциям и узлам. После завершения работы данные возвращаются: через выходные терминалы в сеть проводников ВП. Графически этот процесс описан на рис. 5.1а–5.1г. Если лицевая панель ВПП закрыта и на диаграмме нет узлов свойств, то лицевая панель в память не загружается. Данные передаются напрямую: через разъемы к узлам блок-диаграммы.

Таким образом, иконка и соединительная панель обеспечивают вызов ВПП. Использование ВПП повышает модульность, структурированность системы и позволяет использовать исходный код многократно. Более того, рисовать иконки и создавать соединительную панель просто интересно. В этой части приведены правила хорошего стиля для них.

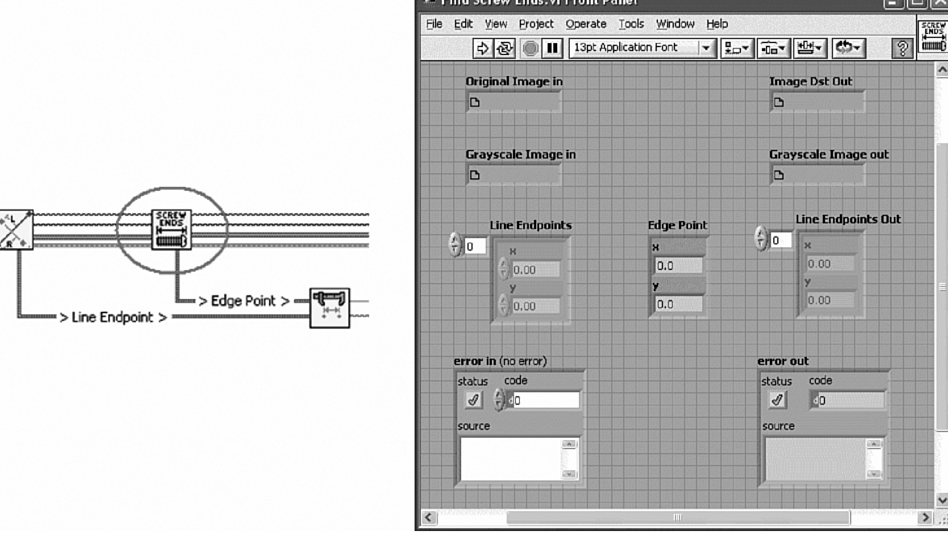


Рис. 5.1а. На диаграмме вызывается ВПП Find Screw Ends (Поиск границ винта). Лицевая панель ВПП открыта, чтобы проследить за потоком данных. Сначала управляющие элементы и индикаторы не активны

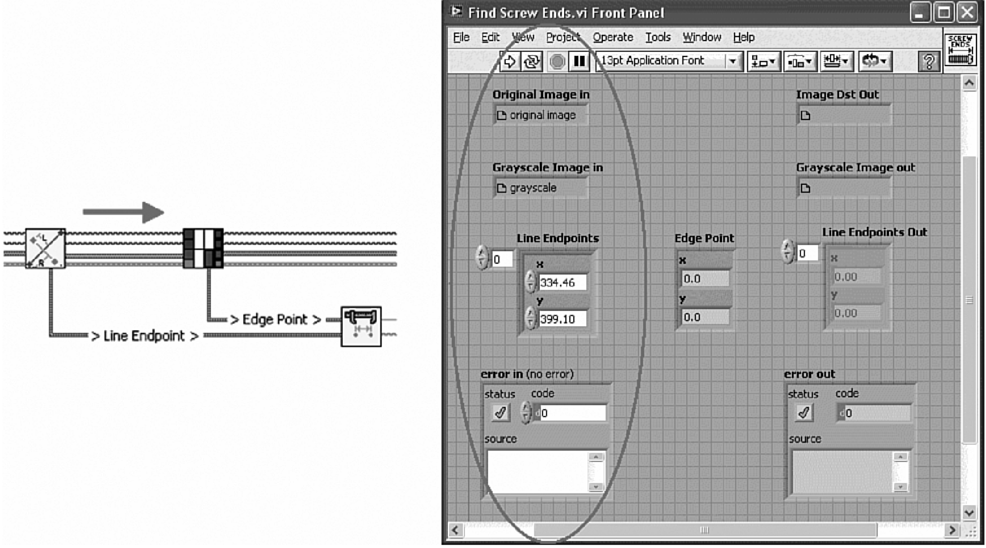


Рис. 5.1б. Вызов ВПП с режимом отображения контактов. Данные передаются от входных контактов на управляющие элементы лицевой панели и далее к терминалам блок-диаграммы

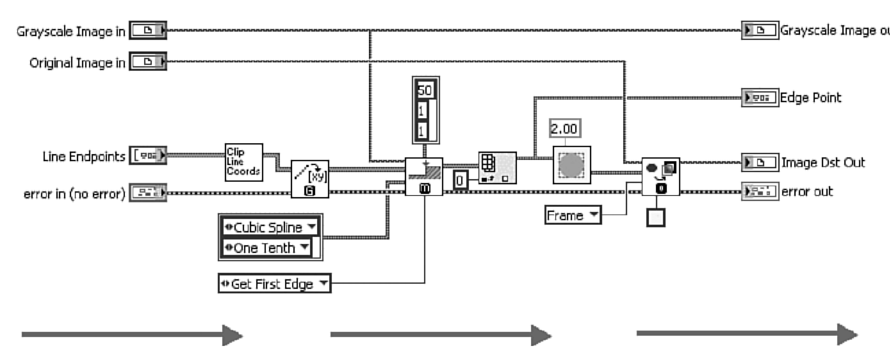


Рис. 5.1в. Данные передаются по терминалам, проводникам и узлам блок-диаграммы

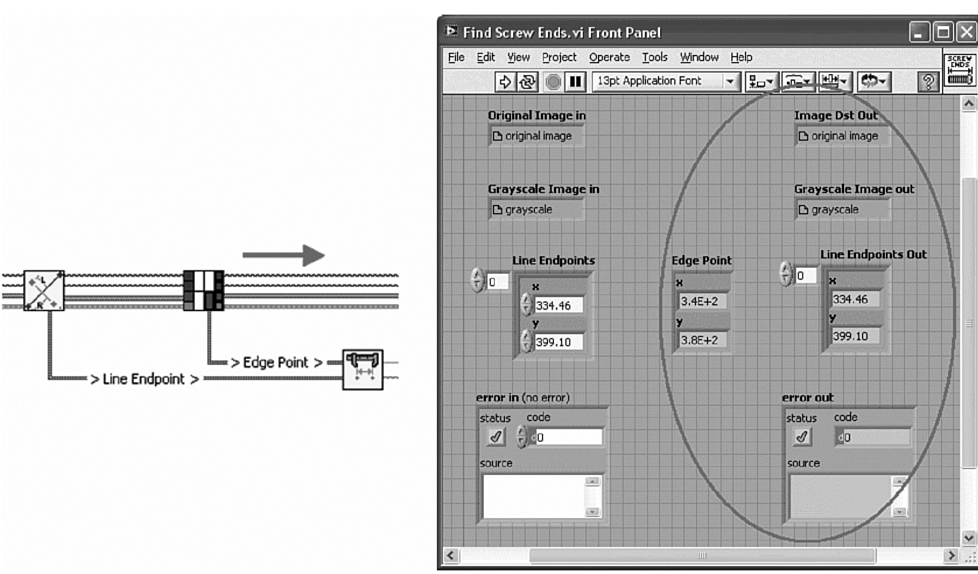


Рис. 5.1г. ВПП возвращает данные в проводники вызвавшего ВП с помощью индикаторов лицевой панели – выходных контактов

5.1. Иконка

Иконка – это изображение размера 32×32 пикселя с произвольной картинкой и текстом. Иконки можно создать с нуля с помощью графического редактора LabVIEW или скопировать из файла или другого приложения. Нарисовать иконку можно в любом приложении. Но большинство из правил этого раздела не требуют ничего, кроме редактора, встроенного в LabVIEW.

5.1.1. Основные правила

Начнем с основных правил создания иконки.

Правило 5.1 Получите удовольствие от создания иконки

Очевидно, что создатели LabVIEW делают все возможное, чтобы мы получили удовольствие от работы. Текстовые программисты могут попытаться испортить нам всю радость, считая, что мы просто рисуем веселые картинки, но на самом деле они нам завидуют, работая медленно и без удовольствия. Создание иконок – это отдых для любого разработчика. Чем больше секретов вы узнаете, тем более полезным он станет.

Правило 5.2 У каждого ВП должна быть своя, причем поясняющая, иконка

Правило 5.3 Никогда не пользуйтесь стандартными иконками LabVIEW

Правило 5.4 Сохраняйте ВП с отображением ВПП в виде иконок, а не контактов

Иконки описывают ВПП, позволяют легче ориентироваться в программе и являются одним из этапов документирования работы. Поэтому очень важно, чтобы они описывали назначение и смысл каждого ВПП. Стандартные иконки LabVIEW унифицированы, они не несут никакого смысла. Всегда заменяйте их чем-нибудь осмысленным. Сохраняйте ВП с отображением ВПП в виде иконок, а не контактов. В следующий раз, когда вы его откроете, будут видны именно иконки.

Правило 5.5 Пользуйтесь черной границей

Черная граница по периметру иконки помогает отделить ВПП от остальной диаграммы. Она должна быть всегда, за редким исключением тех случаев, когда импортируемая картинка занимает все пиксели.

Правило 5.6 Лучшим стилем обладают цветные иконки с глифами и текстом

Правило 5.7 Глифы должны быть описательными

Правило 5.8 Набирайте текст размером 8 или 10

Иконки очень маленькие: всего 32×32 пикселя, поэтому они должны быть краткими, но понятными. Пользуйтесь символами и шрифтами, которые будут легко читаться при таких размерах. Глиф – это графический, легко узнаваемый символ. К ним относятся, например, знаки дорожного движения. Это отличный пример совмещения графики и текста на ограниченном пространстве для сообщения важной информации. Библиотеки глифов, значков и других символов обычно есть в любом приложении и на многих сетевых ресурсах. Например, в зоне разработчиков (NI Developer Zone) такая билиотека есть по адресу www.ni.com/devzone/idnet/library/icon_art_glossary.htm.

Если набирать текст 13-м шрифтом, на иконку влезет всего 3 строки по 5 символов. Этот шрифт подойдет для иконок, для которых достаточно небольшого числа символов, например преобразования градусов по Фаренгейту в градусы по Цельсию: **F -> C**. Чтобы написать полное слово, уменьшите шрифт до 10 или даже 8 и пользуйтесь сокращениями: это минимальный размер текста, который можно прочитать без сильного напряжения, а информации влезет больше всего. Заглавными буквами шрифта размера 8 помещается 4 строчки по 7 символов. Я не призываю вас заполнять каждую иконку только текстом. В соответствии с Правилом 5.6, требуются картинки. Шрифты размера 8 и 10 оставляют для них больше места. Несколько иконок с разным шрифтом и регистром приведено на рис. 5.2.

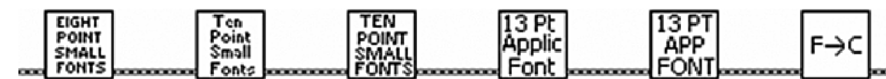


Рис. 5.2. На рисунке приведены иконки с разным шрифтом и регистром. Наименьший шрифт, который можно прочитать, – размера 8 заглавными буквами

Лучшие иконки состоят из глифа, лаконичного текста и окрашены в 2 или 3 цвета. Глиф занимает половину или 2/3 места, текст – от одного до трех слов или понятных сокращений. Окрасить любую точку иконки можно в любой цвет, но обязателен сильный контраст между фоном и основным изображением. Хороший пример – иконка ВП поиска краев винта (Find Screw Ends), рис. 5.3. Изображение винта занимает нижнюю часть картинки, 2 слова – верхнюю треть. Винт и текст – черного и серого цветов, фон – ярко желтый.[1] Это сочетание цветов не подходит для лицевой панели, но в самый раз – для иконки. Объединение графики и текста идеально: каждый элемент дополняет другой. У глифа может быть несколько значений, краткий текст уточняет и поясняет его.

Правило 5.9 Стиль взаимосвязанных ВП должен быть единообразным

Если несколько ваших ВПП взаимосвязаны, предназначены для выполнения одной задачи, например регистрации данных или управления прибором, стиль их иконок должен быть единообразным. Наиболее простой метод – выбрать фон, глиф и шрифт, которые будут общими для всех ВПП и отличать их поясняющим

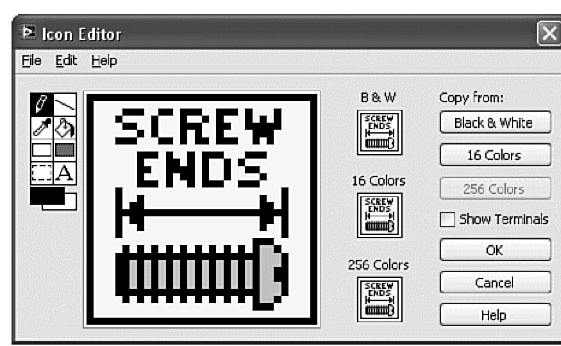


Рис. 5.3. На иконке ВП Find Screw Ends (Поиск краев винта) отлично сочетаются изображение винта и шрифт размера 8

текстом. На рис. 5.4 приведены несколько наборов ВПП, каждый со своим стилем иконки. На иконках ВПП, связанных с приборами контроля давления, глиф – логотип производителя, а слово внизу описывает назначение ВПП. ВПП работы с настроечным файлом объединяются черным текстом на серо-голубом фоне и словом «GET» (Считать) в верхней части, считываемый параметр приведен внизу и отличает файлы. ВПП генерации отчета содержат страницу с отличительным текстом на сером фоне. Для преобразования единиц измерения используются ВПП со стрелкой, словом «UNIT» (Единица измерения), они отличаются измеряемым параметром, который приведен сверху. Таким образом, графические символы, цвет и шрифты для набора взаимосвязанных ВПП одинаковы, они отличаются поясняющим текстом.

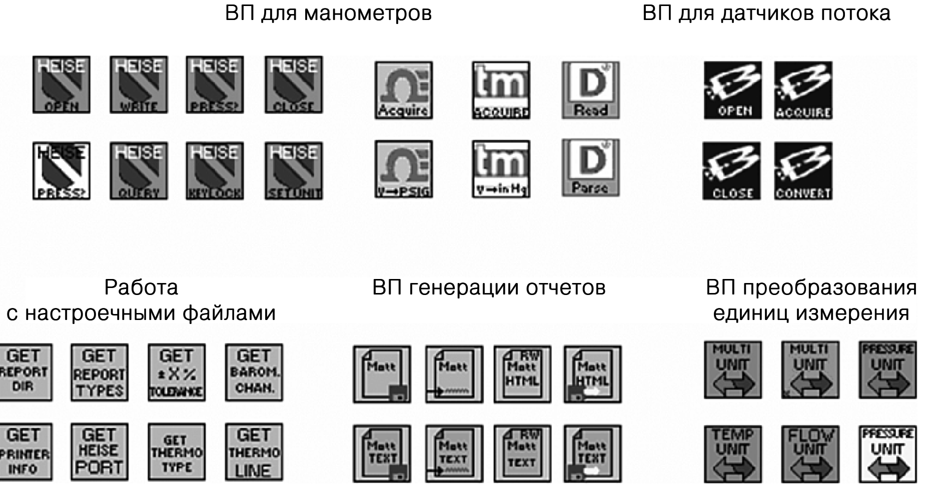


Рис. 5.4. Стили иконок нескольких наборов ВПП

Правило 5.10 Время на создание иконки должно соответствовать планам на ВПП

Усилия и время, которые вы потратите на создание иконки, должны соответствовать планам на использование этого ВПП в будущем. При создании библиотеки разработчика или набора драйверов на «говорящую» иконку можно отвести значительное время. Если же вы работаете над огромным проектом с жесткими сроками и не планируете пользоваться большинством ВПП в будущем, вам потребуются средства делать иконки быстро и единообразно. Это предмет следующего раздела.

5.1.2. Хитрости создания иконок

Создать отличную иконку самостоятельно с нуля почти невозможно. К тому же у многих разработчиков чувство прекрасного отсутствует или ограничено жесткими сроками. В большом приложении нереально превращать каждую иконку в шедевр. Следующие правила помогают сэкономить время при создании простых, но значащих иконок.

Правило 5.11 Быстрее всего сделать иконку с черной границей, цветным фоном и текстом

Правило 5.12 Контраст между текстом и фоном должен быть высоким

Правило 5.13 Выберите цветовую схему для целого набора ВП

По моему опыту, быстрее всего простые, но значащие иконки содержат черную границу, цветной равномерный фон и текст. Другими словами, глиф не обязателен. Выберите цвет фона и текста с большим контрастом: темный на светлом фоне или наоборот. Пусть цвет будет определять тип ВП. В редакторе иконок начните с нуля: стандартной иконки. Выберите основной и фоновый цвета и дважды щелкните по инструменту **Filled Rectangle** (Закрашенный прямоугольник), как показано на рис. 5.5. Обратите внимание, что по умолчанию основной цвет – черный. После этого вместо стандартной иконки появится пустая картинка с нужным фоном и тонкой границей. Так я начинаю делать большинство иконок. После этого шрифтом размера 8 или 10 можно набрать поясняющий текст. Для готовой иконки требуется минимальное время, усилия и воображение. Если вы можете придумать имя для ВПП, то создать иконку с его сокращением или другим кратким описанием труда не составит. Потом, при завершении проекта, можно будет вернуться к иконкам и поразукрашивать их. Если же времени и/или желания нет, как это обычно и бывает, можно ничего не делать: стиль ваших иконок и так хороший.

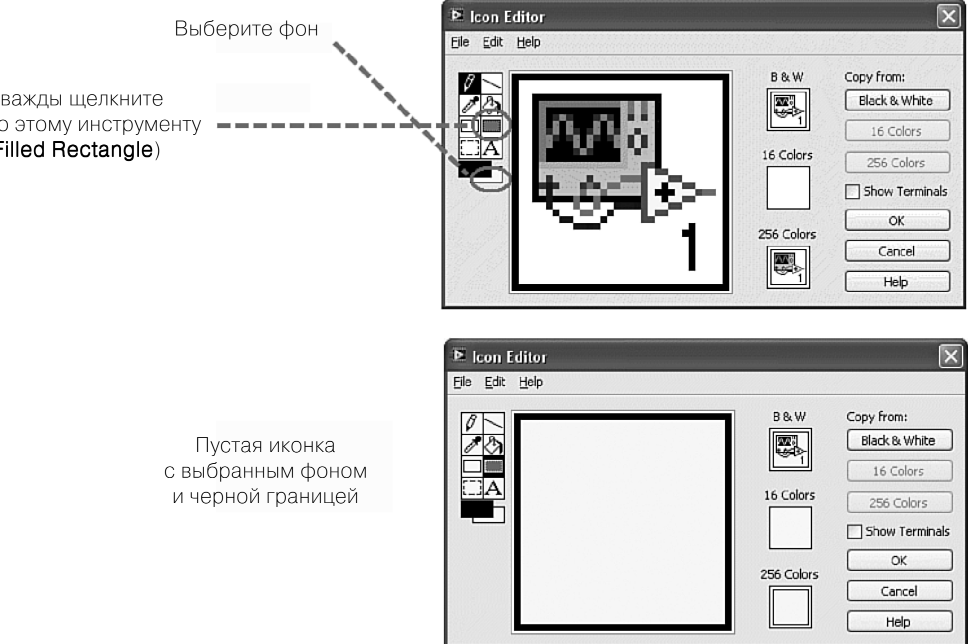


Рис. 5.5. Сверху ВП со стандартной иконкой. Выберите нужный фон и дважды щелкните по Filled Rectangle*, появится пустая иконка с черной границей и выбранным фоном*

Правило 5.14 Создайте шаблон иконки для взаимосвязанных ВП

Правило 5.15 Пользуйтесь одним глифом, цветовой схемой и шрифтом для взаимосвязанных ВП

В коммерческих проектах, например драйверах или специальных библиотеках, иконки должны быть высокого качества. Сделайте в библиотеке проекта или отдельном ВП шаблон иконки и пользуйтесь им для всех ВП приложения. При использовании библиотеки проекта (project library) просто выберите этот шаблон по умолчанию, он будет загружаться для всех новых ВП проекта вместо стандартной иконки. Также можно сделать шаблон ВП с нужной иконкой, контактами, управляющими элементами и индикаторами и сохранить его с расширением VIT. Создайте один глиф, соответствующий типу ВП этой группы и оставьте место для строки текста. Создавайте новые ВП на основе этого шаблона, вам придется только добавить нужный текст. В этом случае вам придется рисовать только один глиф, но воспользоваться им вы сможете сколько угодно раз. Также обратите внимание, что у драйверов есть специальные соглашения, касающиеся иконок и контактов. Прочитайте руководство NI по созданию драйверов[2].

На рис. 5.6а приведена иконка-шаблон библиотеки для тестирования полупроводниковых пластин. На нем изображен глиф пробника над полупроводниковым прибором и текста с аббревиатурой производителя и обозначением модели. Как показано на рис. 5.6б на дереве структуры приложения, этот шаблон используется во всех ВП приложения. Разработчик потратил время на одну иконку, а результат используется во всем приложении. Мы вернемся к этой иконке в разделе 5.3. «Примеры».

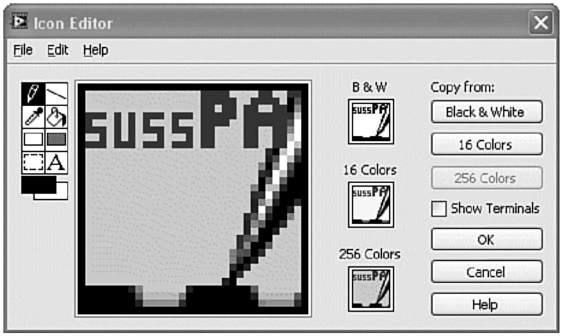


Рис. 5.6а. Шаблон иконки библиотеки управления Suss Interface Toolkit состоит из пробника над полупроводниковым прибором и текста с обозначением производителя и модели

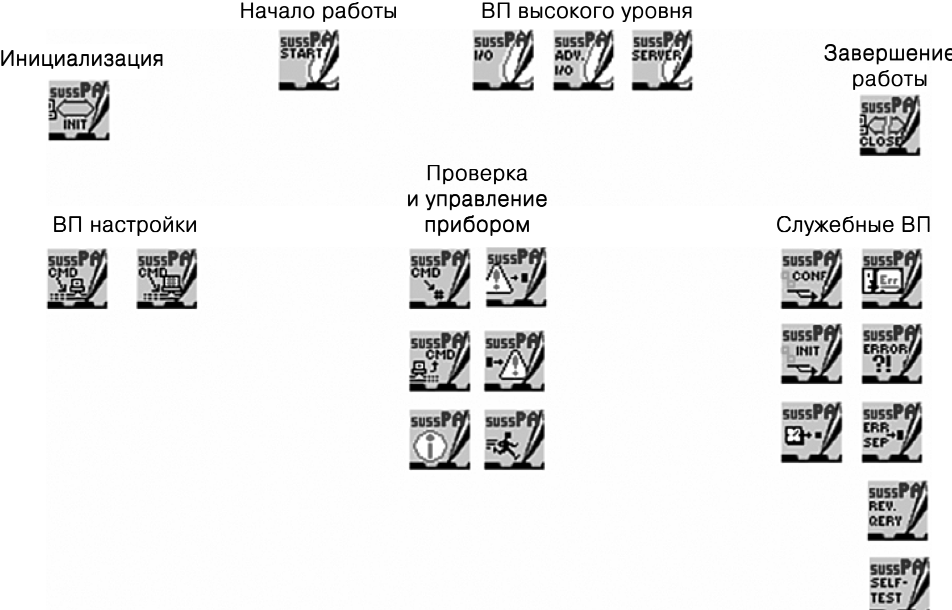


Рис. 5.6б. Этот шаблон используется во всех иконках библиотеки

Правило 5.16 Копируйте графику

Если картинка не защищена законом об авторском праве, скопировать ее лучше, чем нарисовать с нуля. Копируйте части иконок LabVIEW, импортируйте изображения из различных библиотек и файлов. При копировании сделайте так. чтобы ваши иконки нельзя было перепутать с оригиналом. Например, иконки работы с файлами OpenG File Tools напоминают (рис. 5.7) иконки с соответствующей палитры LabVIEW из папки vi.lib. Иконки OpenG просто скопированы оттуда, но их фон – светло-зеленый, и их нельзя перепутать со стандартными иконками. Еще один пример: для иконок ВП регистрации данных мне нравятся изображения дискетки и карандаша, которые часто встречаются при работе с файлами (рис. 5.8, ВП Save Scan). Текст, цвет фона и граница отличаются от стандартных, их не перепутаешь.

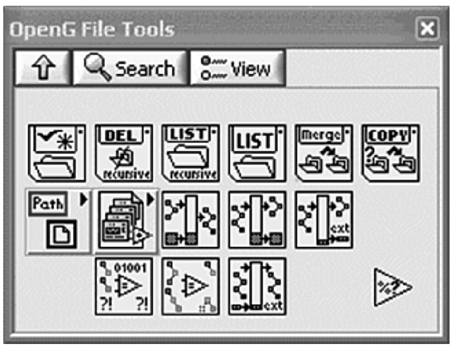


*Рис. 5.7. Иконки работы с файлами из палитр **OpenG File Tools** и **File I/O** аналогичны. Они отличаются только цветом фона*

*Рис. 5.8. Изображение дискеты и карандаша я скопировал из экспресс-ВП с палитры **File I/O**. Чтобы не перепутать, фон и текст сделаны другими*

В сети есть много бесплатных библиотек иконок. Большинство браузеров позволяют сохранять содержащиеся на страницах картинки. В Internet explorer нужно просто в контекстном меню картинки выбрать пункт **Copy** (Копировать). После этого вставьте изображение из буфера в редактор иконок LabVIEW. Можно просто перетащить картинку из браузера на иконку лицевой панели ВП. Если нужно сжать картинку до размера 32×32, сначала сохраните в файл, например, с помощью Microsoft Paint, отредактируйте и масштабируйте изображение, сохраните файл и импортируйте в буфер в редакторе иконок с помощью меню **Edit ⇒ Import Picture to Clipboard** (Редактировать ⇒ Импорт картинки в буфер). LabVIEW поддерживает следующие стандарты картинок: BMP, JPEG, PNG, MNG и GIF. Также можно перетащить файл изображения на иконку на лицевой панели. В последнем случае LabVIEW автоматически масштабирует картинку до размера 32×32. Обратите внимание на коллекцию в зоне разработчиков[3]. Все глифы в ней уже подходящего размера.

5.1.3. Международные иконки

LabVIEW используется в разных странах, локализованные версии вышли уже на нескольких языках. Иконки – это обычные изображения, они выглядят одинаково вне зависимости от языка системы. Это обстоятельство нужно учитывать, если вы разрабатываете приложение для других стран. Таким приложением может быть библиотека разработки или драйвера. Чтобы не вызвать недоразумений, соблюдайте следующее правило.

Правило 5.17 Избегайте локализованных текста и графики

Текстовые иконки проще и быстрее всего сделать, но понимание текста зависит от языка. Лучше пользоваться глифом и строкой дополняющего текста. Например, во всех драйверах приборов LabVIEW используется обозначение производителя и модели, которое называется префиксом прибора (instrument prefix). Это стандарт, принятый организацией VXIplug&play Consortium.

Многим разработчикам LabVIEW нравятся глифы, косвенным образом относящиеся к действию ВП. Например, дровосек или поленица часто обозначают регистрацию данных, ружье – подготовку (arming) триггера, унитаз – очистку буфера или файла. Еще один пример приведен на рис. 5.9: на иконке ВП преобразования файла («вытаскивания» строки из файла с преобразованием) (file casting) изображен рыбак, забрасывающий (to cast) удочку. Эти иконки подойдут для личного пользования, но в международных проектах их стоит избегать.

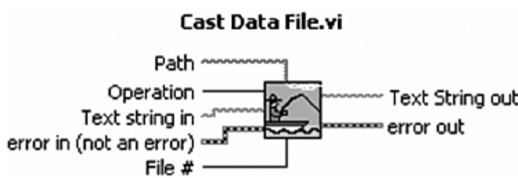


Рис. 5.9. Изображение рыбака обозначает операцию преобразования файла. Эти термины обозначаются одним словом только в английском языке, но, например, в русском аналогия не так очевидна и не используется

5.2. Соединительная панель

Соединительная панель (connector pane) – это терминал подключения контактов к ВПП. В этом разделе приведены правила ее оформления.

Правило 5.18 Выберите расположение контактов и их назначение, чтобы обеспечить правильный поток данных и не усложнять соединения

Если вы правильно выбрали расположение контактов и их назначение, вам будет проще поддерживать поток данных на вызывающем ВП. Правила, касающиеся сети проводников и потока данных, приведены в предыдущей главе. Напомним здесь некоторые из них для удобства.

Правило 4.11 Избегайте петель и изгибов проводников

Правило 4.12 Параллельные проводники должны проходить на одинаковом расстоянии

Правило 4.22 Всегда направляйте поток данных слева направо

Правило 4.23 Пользуйтесь кластером ошибок

Мы начинаем с некоторых правил соединений, которые помогут предотвратить спутанность проводов.

Правило 5.19 Оставляйте на шаблоне панели пустые разъемы

С развитием приложения часто приходится добавлять дополнительные входные и выходные каналы к ВПП, для этого нужно добавлять подключения соединительной панели. Если у вас нет запасных разъемов, приходится изменять шаблон контактов, при этом нарушаются связи проводников между ВПП и вызывающей его диаграммой. В лучшем случае проводники просто сдвинутся, но могут и порваться. Придется вручную восстановить связь каждой копии ВПП командой **Relink to SubVI** из контекстного меню. Действия простые, но они могут понадобиться в разных частях всего приложения, придется потратить на них много времени. Если есть запасные разъемы, шаблон соединительной панели не изменяется.

Правило 5.20 Шаблон взаимосвязанных ВПП должен быть единообразным

У взаимосвязанных ВПП часто есть общие проводники, которые нужно соединить между собой. Например, в приборах сбора данных необходимо связать все ВПП в цепочку проводниками Task (Задачи) и кластером ошибок (error cluster). Аналогично соединяются имена ресурсов VISA (VISA resource name) и опять же кластеры ошибок в ВПП управления приборами. Если шаблон ВПП единообразный с одинаковым расположением одинаковых данных, проводники идут аккуратно и без лишних изгибов. Как показано на рис. 5.10, у всех драйверов DAQmx шаблон 5×3×3×5, при этом терминал Задачи – верхний (левый и правый), кластер ошибок внизу. Во всех ВПП управления цифровым осциллографом (niDMM) используется шаблон 4×2×2×4 – рис. 5.11.

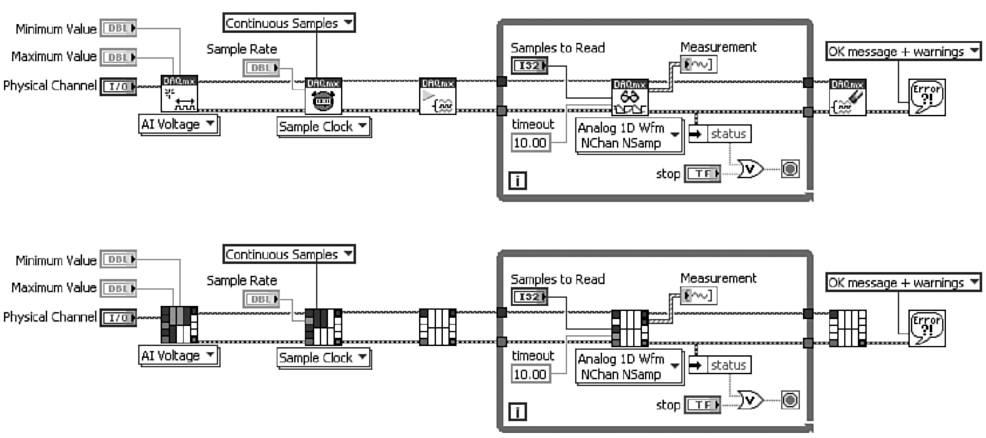


Рис. 5.10. ВП Cont Acq&Graph Voltage-Int Clk (Непрерывный сбор и отображение напряжения, тактирование по внутреннему генератору) состоит из нескольких ВП сбора данных. Стандартный режим отображения иконок приведен сверху, как видно на нижнем рисунке, шаблон этих ВПП – 5×3×3×5

Правило 5.21 Для большинства ВП подойдет шаблон 4×2×2×4

Давайте усилим правило использования стандартных шаблонов и выберем один для большинства ВП, вне зависимости от их цели и назначения. Наилучшие диаграммы получаются, если все соединительные панели единообразны, а для каждого типа данных есть свое место. Начиная с версии LabVIEW 8.2, шаблон 4×2×2×4 автоматически устанавливается для новых ВПП. По моему мнению, он обеспечивает более высокий уровень модульности программы, чем шаблоны с большим числом терминалов, но обеспечивает достаточное количество контактов для данных. Если вам не хватает 12 контактов, то, скорее всего, один ВПП решает слишком много задач или некоторые данные нужно объединить в кластер.

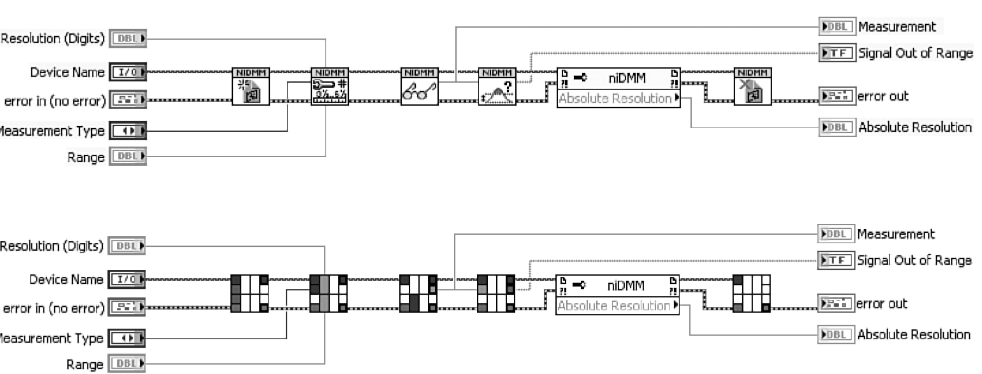


Рис. 5.11. ВП DMM Measurement (Измерения цифровым осциллографом) – это пример использования драйверов niDMM. Как видно на нижнем рисунке, шаблон этих ВПП – 4×2×2×4

Например, в подприборе поиска границ винта (ВП Find Screw ends – рис. 5.1) используется стандартный шаблон. Координаты концов линии (**Line Endpoints**) объединены в кластер, аналогично положение границы на выходе также представляет собой кластер. Вместе с дополнительными 6 входными и выходными параметрами эти данные занимают 9 из 12 контактов, остается 3 свободных. Как показано на рис. 5.10, у всех ВПП драйверов DAQmx шаблон 5×3×3×5, хотя на рисунке используется всего 11 из 16 контактов. Эти ВП полиморфные, они поддерживают множество разного оборудования, в некоторых случаях 12 контактов стандартного шаблона может не хватить.

Всегда могут оказаться исключения, когда стандартный шаблон 4×2×2×4 не подойдет. Однако этот шаблон с назначенными стандартными терминалами, например кластера ошибок, может сэкономить не меньше минуты рабочего времени для КАЖДОГО из ваших ВП. В LabVIEW есть такой шаблон: ВП с обработкой ошибок (**SubVI with Error Handling**). Чтобы его открыть, выберите команды меню **File** ⇒ **New...** ⇒ **VI** ⇒ **From Template** ⇒ **Frameworks** (Файл ⇒ Новый... ⇒ ВП ⇒ Из шаблона ⇒ Стандартные).

Правило 5.22 Входные данные должны быть слева, результаты справа

Правило 5.23 Не допускайте пересечения проводников в окне контекстной помощи

Соединительная панель не должна нарушать поток данных, не допустимы лишние изгибы и пересечения проводников. Для этого управляющие элементы должны быть связаны с левыми контактами соединительной панели, а индикаторы – с правыми. В противном случае проводники входных и выходных данных будут пересекаться. Средние терминалы можно использовать как для ввода, так и

для вывода данных, но только если при этом проводники не пересекаются. Чтобы проверить правильность назначения контактов, воспользуйтесь окном контекстной помощи.

У ВП на рис. 5.12а используется шаблон 5×3×3×5 для ввода и вывода большого количества различных массивов. Входные массивы – перечень параметров для аналоговых модулей генерации данных типа FieldPoint, а выходные – результаты измерений, полученные с аналоговых приборов регистрации данных. Один из входных массивов соединен с верхним правым контактом, а два выходных – с верхними левыми контактами, из-за этого проводники в окне контекстной справки пересекаются. Это либо грубейшая ошибка, либо результат лени разработчика изменять назначение контактов при редактировании соединительной панели. Разумеется, после изменения назначения существующих контактов придется найти все копии этого прибора и отредактировать их. Исправленный ВПП приведен на рис. 5.12б – входные данные находятся слева, выходные справа, в окне контекстной справки нет никаких пересечений. На рис. 5.12в все входные данные находятся слева и снизу, а выходные – справа и сверху. Варианты на рис. 5.12б и 5.12в эквивалентны, допускаются и тот, и другой. Но при использовании шаблона 5×3×3×5 количество проводников и сложность блок-диаграммы увеличиваются. На рис. 5.12г входные и выходные массивы объединены в кластеры, количество проводников сразу уменьшилось. Если нужно добавить новые массивы данных, можно изменить кластер, не трогая соединительную панель. Также теперь можно воспользоваться более компактным шаблоном 4×2×2×4.

Правило 5.24 Расположение контактов и элементов лицевой панели должно соответствовать друг другу

FP Measure & Control with Crossover.vi

FP-AO-210 @8 SetPoints
Main RT VI Refnum
FP-AO-200 @6 SetPoints
FP-AO-200 @7 SetPoints
Error In (no error)
Non-PID setpoints
FP-AI-110 @5 Measurements
FP-AI-110 @1 Measurements
dup Main RT VI Refnum
FP-AI-110 @2 Measurements
FP-AI-110 @3 Measurements
Error Out
FP-AI-110 @4 Measurements

Рис. 5.12а. Используется шаблон 5×3×3×5. В окне контекстной помощи видно пересечение проводников

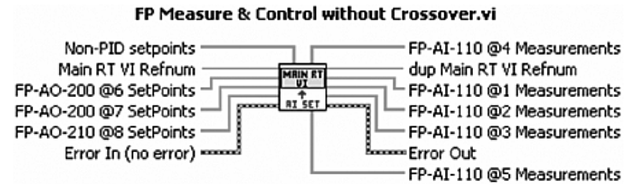


Рис. 5.12б. Входные данные слева, выходные справа, нет пересечений проводников

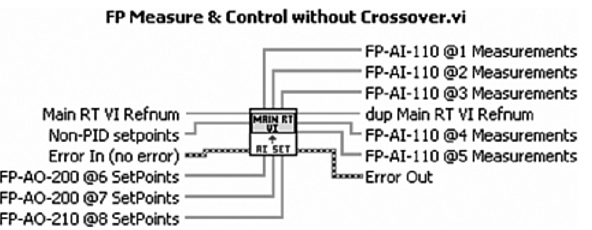


Рис. 5.12в. Входные данные слева и снизу, выходные справа и сверху, нет пересечений проводников

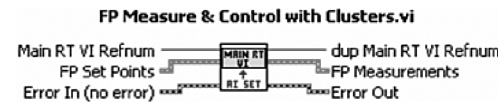


Рис. 5.12г. Входные массивы параметров, а также результаты измерений объединены в кластеры, используется шаблон 4×2×2×4. Количество проводников значительно сократилось, блок-диаграмма стала проще

Правило 5.25 Кластеры ошибок должны быть снизу с краю

Правило 5.26 Ссылки и имена ресурсов должны быть сверху с краю

Уже давно принято назначать кластеры ошибок нижним крайним терминалам, а ссылки на ресурсы и их имена – верхним угловым. Сверху соединяются ссылки на файлы (file refnum), приложения сервера ВП (VI Server application) и ВП, а также имена задач DAQmx (DAQmx task), каналов (channel names), ресурсов VISA (VISA resource name) и другие аналогичные данные. Оставшиеся элементы управления и индикаторы назначаются другим контактам, причем расположение контактов и элементов должно соответствовать друг другу. Другими словами, соединительная панель должна быть миниатюрным изображением лицевой панели.

Такое расположение управляющих элементов и индикаторов позволяет легко ориентироваться на лицевой панели. Также указанные правила помогают расположить контакты многих проводников на одном уровне и, таким образом, избежать лишних изгибов и путаницы. Обратите внимание, что в примере на рис. 5.12 ссылки на ресурс (**Main RT VI Refnum**) грамотно расположены в верхних углах, а кластеры ошибок – в нижних углах.

Правило 5.27 Контакты на вертикальных границах соединительной панели предназначены для основных данных

Правило 5.28 Контакты на горизонтальных границах соединительной панели предназначены для дополнительных параметров

Обратите внимание на важность поступающих и выходных данных. Их приоритет можно указать либо явно, установив будет ли данный разъем *обязательным* (required), *рекомендуемым* (recommended) или *дополнительным* (optional) из контекстного меню, либо неявным расположением контакта. Разъемы важных данных располагаются на вертикальных границах соединительной панели, дополнительных параметров – на горизонтальных. Расположение элементов на лицевой панели должно соответствовать контактам, как мы уже обсудили в главе 3 «Стиль лицевой панели».

Правило 5.29 Установите свойство «обязательный разъем» для критически важных данных

Правило 5.30 Укажите свойство «дополнительный разъем» для необязательных параметров

Если ВПП работает с файлом, занимается сбором данных или управляет прибором, то есть работает с ресурсом, который был открыт до вызова этого ВПП, ему необходимо описание этого ресурса: ссылка, задача или имя. У контактов этих и других необходимых данных должно быть свойство «обязательный разъем». Для другой группы разъемов в большинстве случаев используется значение по умолчанию, у них можно установить свойство «дополнительный разъем». Благодаря этим правилам разработчик сразу видит выделенные жирным шрифтом важные контакты в окне контекстной помощи. Для большинства остальных контактов стоит оставить приоритет по умолчанию: «рекомендуемый разъем».

На рис. 5.13 приведено окно контекстной помощи с ВП поиска границ винта. Ссылки на **Original Image in** (Исходное изображение) и **Grayscale Image in** (В оттенках серого) необходимы для работы ВП. Имена соответствующих контактов

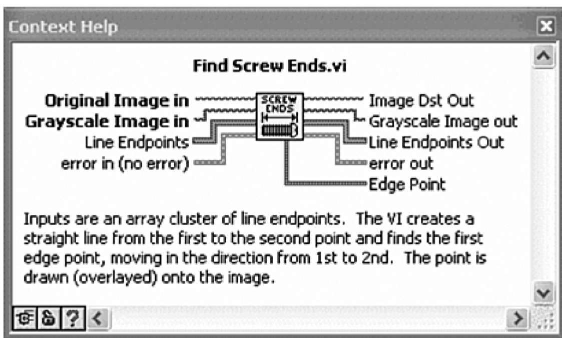


*Рис. 5.13. Как видно в окне контекстной помощи ВП Find Screw Ends (Поиск границ винта), исходное изображение (**Original Image in**) и **Grayscale Image in** (В оттенках серого) необходимы для данного ВП, все остальные данные рекомендуемые*

выделены жирным шрифтом, подчеркивая высокий приоритет этих данных. Все остальные контакты рекомендуемые. Для всех входных данных есть соответствующие им выходные, их контакты расположены на одном уровне. Это сделано для большей аккуратности и удобства построения блок-диаграммы. Но из-за этого пришлось вынести важный терминал координат узла (**Edge Point**) на нижнюю границу лицевой панели.

Обратите внимание, что во многих ВП используется номер итерации цикла: например, если это первая итерация, нужно инициализировать или запустить прибор. Терминал счетчика итераций обычно располагается в левом нижнем углу, поэтому и соответствующий контакт ВПП может быть на нижней границе.

5.3. Примеры

В этом разделе содержатся примеры различных соединительных панелей и иконок. Они иллюстрируют и подчеркивают правила главы, но иногда лучше доказать правило от противного. Начнем с таких примеров.

5.3.1. Доказательства от противного

На рис. 5.14а приведена иконка, данные и контакты ВПП, созданного из участка блок-диаграммы (**SubVI from Selection. VI**), с которым мы уже встречались в предыдущих главах. Этот ВПП был создан автоматически из участка блок-диаграммы с помощью команды **Edit ⇒ Create SubVI** (Редактировать ⇒ Создать ВПП). Такие приборы можно сразу узнать по стандартной иконке, случайному расположению элементов лицевой панели, путающей метки контактов и хаотичной блок-диаграмме. Как мы уже убедились в предыдущих главах, без чистки хорошего стиля с такими ВПП никогда ничего не получится. Как можно использовать такие метки: **task ID out**, **task ID out 2**, **error out 2** и **output chunk** (выходной сигнал)? Во входных данных нельзя использовать предлог out, имена не могут различаться только порядковым номером. Также стандартная иконка LabVIEW неинформативна. И наконец, на шаблоне 3×2 не остается свободных контактов. Таким образом, нарушены Правила 3.21, 5.2, 5.3, 5.19 и 5.21.

В результате исправления указанных недостатков получился ВП SubVI from Selection w Cleanup, он показан на рис. 5.14б. Имена контактов различаются не номерами, а указателями типа данных: для входных **in**, для выходных – **out**. У входного массива данных некорректный префикс **output** заменен на **input**.

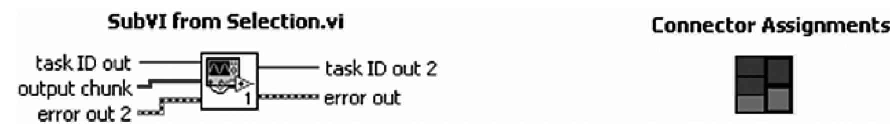


Рис. 5.14а. В оформлении ВП нарушено множество правил, например стандартная иконка и путающие имена данных

Рис. 5.14б. Недочеты в оформлении ВП устранены: иконка отражает смысл ВП, имена терминалов не противоречат их назначению, используется стандартный шаблон соединительной панели

Иконка ВП теперь отражает его назначение. На стандартном шаблоне соединительной панели 4×2×2×4 осталось множество свободных контактов.

Сколько раз мне встречались иконки с нарисованным от руки текстом вместо соответствующего инструмента. Два таких примера приведены на рис. 5.15 и 5.16. На рис. 5.15а приведен один из подприборов ВП Haphazard, с которым мы познакомились в главе 4. Также можно заметить, что один из выходных параметров расположен слева, а ссылка на задачу (task reference) объединена с кластером ошибок. Эти данные не связаны между собой и объединять их не следует. Исправленный вариант представлен на рис. 5.15б: улучшен внешний вид иконки, выбран правильный шаблон соединительной панели и исправлены недостатки в назначении контактов. Слова **Digital inputs** (Цифровые данные) написаны шрифтом размера 8 и влезли на иконку без сокращений. Глиф – очки, он скопирован из ВП LabVIEW DAQmx Read. Шаблон соединительной панели заменен на стандартный 4×2×2×4, имя задачи (**DAQmx Task**) проходит через верхние угловые терминалы.

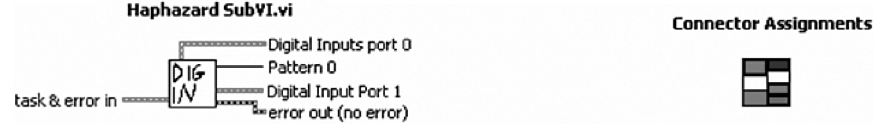


Рис. 5.15а. Текст иконки написан от руки, назначения терминалов не соответствуют стандартным

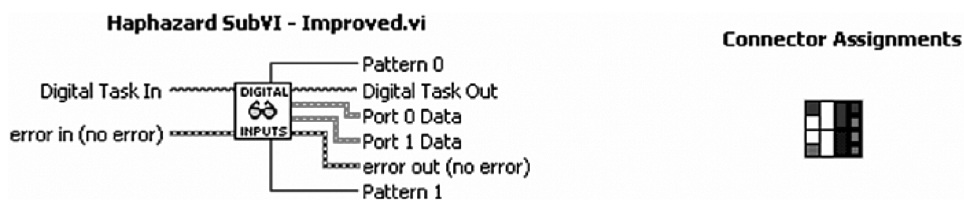


Рис. 5.15б. Исправлен внешний вид иконки: текст написан стандартным шрифтом и добавлен глиф. Имя задачи (DAQmx Task) проходит через верхние угловые терминалы

На рис. 5.16а приведена иконка со шрифтом, разработанным в Bloomy Controls; работники называют этот шрифт *Bob's Bold* (Жирный Боб) по имени ее создателя Боба Гамбургера (Bob Hamburger). Боб – уважаемый специалист, коммерческий директор компании. Он также разработал иконки ВП поиска границ винта (рис. 5.4), которые всегда приводятся как пример хорошего стиля. Я бы хотел поблагодарить Боба за иконки, как великолепные, так и отвратительные, которые он предоставил в общее пользование. Символы Жирного Боба такие большие, что кажется – они кричат на вас с экрана (взгляните на рис. 5.16б). Этот шрифт – отличный способ привлечь внимание к иконке, только не стоит им злоупотреблять, чтобы не оглохнуть.

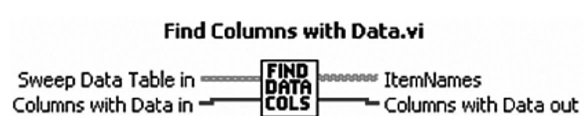


Рис. 5.16а. Для текста на этой иконке используется шрифт Bob's Bold (Жирный Боб)

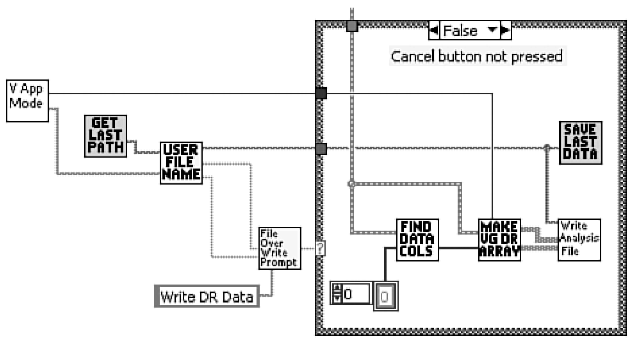


Рис. 5.16б. Иконки с Жирным Бобом кричат с блок-диаграммы

5.3.2. Драйверы приборов

В этом разделе приведены примеры иконок и соединительных панелей с драйверов приборов. В руководстве по использованию драйверов (NI Instrument Driver Guidelines) приведены подробные инструкции по созданию Plug and Play драйверов приборов. Большинство из этих правил перекликаются с правилами этой главы и книги, редкие исключения я буду всегда подчеркивать.

На рис. 5.17а приведен ВП Initialization (Инициализация) вымышленного медицинского прибора, PM-1000. На иконке расположено три линии текста и глиф: изображение пациента на больничной койке. Глиф замечательный, но текст слишком плотный, и иконка получилась немного тяжеловесной. Шаблон соединительной панели 5×3×3×5, 12 из 16 контактов рекомендуется использовать. Если подключить все 12 разъемов, в путанице проводов разобраться будет непросто.

Если внимательнее посмотреть, можно увидеть, что все 14 символов двух строчек иконки относятся к производителю и модели прибора. Это не очень хорошо.

В соответствии с общепринятым соглашением стандарта VXI*plug&play* префикс прибора должен быть кратким. Первые 2 или 3 символа должны характеризовать производителя, оставшиеся 4 или 5 символов – модель или семейство. Получившаяся аббревиатура из 6–8 букв и цифр используется и в имени файла, и на иконке. Также 5 контактов отвечают за настройку последовательного порта, в дополнение к отдельному разъему **Serial Port**. Эти 5 данных тесно связаны между собой, их можно объединить в кластер и снизить число входов. Тогда хватит рекомендуемого шаблона 4×2×2×4. Вход **Serial port** – имя ресурса ввода/вывода, он корректно расположен в левом верхнем углу в соответствии с Правилом 5.26.

На рис. 5.17б указанные недостатки устранены: 2 верхние строки иконки заменены на префикс прибора PHPM1k. Также глиф теперь расположен в центре, иконка стала более чистой и приятной. Данные последовательного порта объединены в кластер, шаблон стал 4×2×2×4. Приоритет входных данных ID Query и Reset ниже, чем у остальных, поэтому соответствующие разъемы расположены посередине, левая граница оставлена для более важных.

На рис. 5.17в приведена иконка и контакты ВП Initialize из шаблона драйверов. Этот ВП создан автоматически командой **Create New Instrument Driver Project** из меню **Tools** ⇒ **Instrumentation** ⇒ **Create Instrument Driver Project** (Инструменты ⇒ Работа с приборами ⇒ Создать проект драйвера приборов). Контакты данных **ID Query** и **Reset** расположены на левой границе, а **Serial Port Settings** – сверху посередине. Последние два рисунка эквивалентны, но последний лучше соответствует стандартам, чем на рис. 5.17б. На рисунке расположен стандартный глиф, который размещается на всех ВП инициализации. Стандартный глиф помогает визуально выделить ВП драйверов не только по тексту. Также он экономит часть усилий разработчика, которому не требуется его рисовать. Тем не менее стандартный глиф не связан с конкретным прибором и ослабляет внешнюю связь иконки с назначением прибора. Таким образом, единственным связующим звеном с прибором остается префикс. Поэтому нужно сделать выбор: либо глиф относится к назначению ВП, либо к прибору. Стандарт NI склоняется к первому варианту, я предпочитаю второй.

Перед тем как вернуться к версиям PHPM1k Initialize VI, необходимо сделать несколько замечаний. В зависимости от структур данных приложения, для пере-

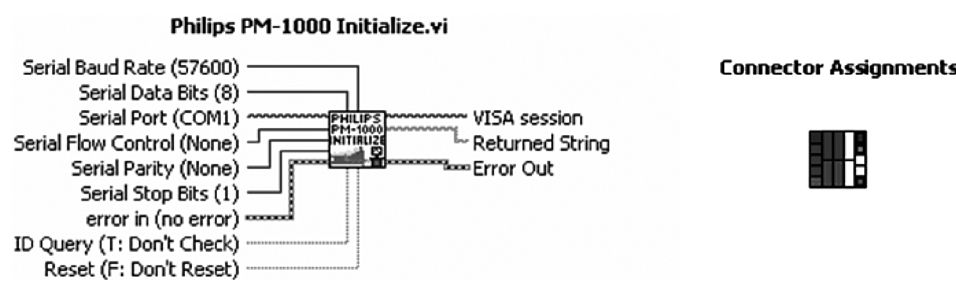


Рис. 5.17а. Драйвер фиктивного медицинского прибора со слишком большим количеством разъемов и тяжеловесной иконкой

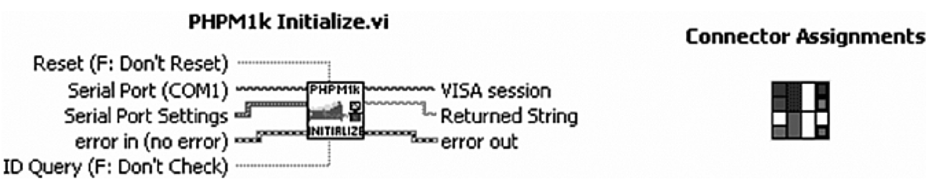


Рис. 5.17б. Недостатки исправлены: в заголовке иконки префикс прибора, глиф в центре, данные сгруппированы в кластер, количество контактов уменьшено

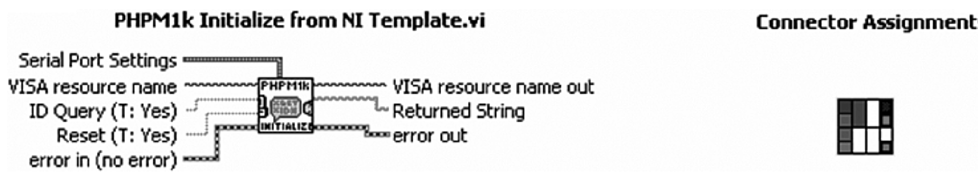


Рис. 5.17в. Иконка и разъемы ВП, созданного автоматически. Сгенерировались стандартный глиф, имена разъемов и назначение терминалов

дачи данных последовательного порта в ВП необходимо объединить их в кластер. Надо пойти еще на один компромисс: либо много разъемов на иконке, либо дополнительные операции с кластером. Как отмечается в руководстве по созданию драйверов NI, при использовании кластера возрастает сложность приложения. Тем не менее, мне кажется, лучше придерживаться одного стиля в приложениях и в драйверах. А именно – я считаю, что кластеры позволяют объединить данные близкого назначения в одном месте. Также использование кластеров позволяет лучше структурировать данные по всему приложению. Например, программист сможет связать несколько разных ВП в цепочку этим кластером настроек последовательного порта. Более подробно типы данных и структуры обсуждаются в главе 6 «Структуры данных».

На рис. 5.18 приведено три версии палитры функций из библиотеки Suss Interface, а также иконка ВП DoProberCommand. Этот драйвер прибора, с которым мы уже познакомились в разделе «Ссылки иконок» в этой главе, управляет системой контроля размещения полупроводниковых пластин. Иконка этих драйверов отличается следующими особенностями: заголовок-префикс прибора **SussPA**, изображение датчика рядом с полупроводниковым прибором. Ярко-желтый фон создает отличный контраст и выделяет иконки на блок-диаграмме. Стиль иконок подменю на палитре аналогичный, но их цвет более темный. Так они сразу отличаются от ВП, и создается впечатление объема изображения. Иконки выглядят очень профессионально и отлично подходят для коммерческого приложения. При создании глифа сначала подобрали изображение пробника полупроводникового прибора в общедоступном архиве, а потом оптимизировали его к размеру 32×32. Тот же самый глиф есть на всех иконках этой библиотеки, как и подобает качественной коммерческой разработке. Таким образом, в данном случае на стиль иконки нужно было потратить заметное время и усилия.

ВП **DoProberCmd** – одна из программ с палитры **Action/Status** (Действия/Статус). Иконка на рис. 5.18а соответствует стилю палитры с тонким изображением бегуна (runner). Но имя и изображение на иконке могли бы лучше соответствовать друг другу. Почему бы не переименовать ее в SussPA**Run**ProberCmd? Или заменить бегуна на слово DO. Более того, эта библиотека предназначена для международного распространения, а ассоциация между бегуном (runner) и исполнением (run) команд существует не во всех языках. Также обратите внимание, что входные данные **Registration In** (Регистрация) и **Command** (Команда) – только рекомендуемые.

На рис. 5.18б приведена палитра с аналогичным драйвером, созданная на основе шаблона LabVIEW с помощью мастера драйверов (Create New Instrument Driver Project). На ней расположены субпалитры и ВП с глифами стандартных функций вместо глифа прибора. Единственный элемент, определяющий прибор, – заголовок иконки. Изображение получилось достаточно общим, без всякой связи с тестированием полупроводников. На иконке ВП DoProberCmd расположен зеленый треугольник, который обычно показывает операции запуска задачи. Этот глиф лучше подходит для описания функции ВП, но связь с прибором слабая.

На рис. 5.18в объединены достоинства стандартных глифов с назначением прибора. Этот ВП лучше совместим со стандартами Plug and Play и лучше выделяется на блок-диаграмме. Получившийся драйвер интуитивно-понятный и соответствует всем стандартам.

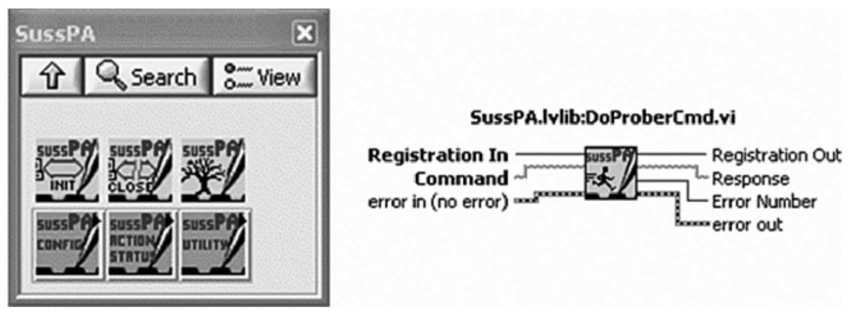


Рис. 5.18а. Палитра функций библиотеки Suss Interface для LabVIEW, коммерческого приложения по управлению системой контроля размещения полупроводниковых пластин. На всех иконках есть общий глиф прибора. ВП DoProberCmd из палитры **Action/Status** с силуэтом бегуна и общим глифом

К сожалению, не всегда возможно совместить глифы прибора и назначения ВП на одной иконке 32×32 точки, иногда даже невозможно. Если у ваших иконок есть заголовок с префиксом прибора, выделяющаяся цветовая схема и разумный баланс между графикой и текстом, а все требуемые ВП и их палитры организованы в соответствии с рекомендациями NI, то, скорее всего, ваши иконки соответствуют стандарту LabVIEW Plug and Play instrument driver. Очевидная связь между иконками и прибором гораздо важнее одинакового глифа на всех изобра-

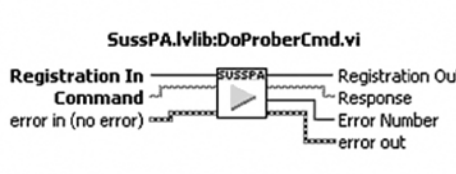


Рис. 5.18б. Альтернативный стиль иконок: стандартные глифы из шаблона драйверов. Иконка DoProberCmd с зеленым треугольником запуска или инициации задачи. Стиль иконок слабо связывает их с тестированием полупроводников

Рис. 5.18в. Стандартные глифы шаблонов совмещены с глифами приложения, иконки отражают и назначение ВП, и приложение

жениях. Также пользователи должны понимать назначение прибора по его имени и расположению в палитре.

5.3.3. Разные примеры

В этом разделе приведены несколько примеров, которым требуются различные исправления. ВП, иконка и разъемы которого приведены на рис. 5.19а, преобразует структуру данных в форматированную строку. На иконке находится замечательный глиф телефона и поясняющий текст. Я просто восхищаюсь работой программиста, все равно, нарисовал он глиф самостоятельно или нашел где-нибудь. Соединительная панель, однако, требует исправлений. Во-первых, метки элементов слишком длинные. Имена **Communication Parameters Cluster** (Кластер параметров связи) и **Communication Parameters String** (Строка параметров связи) обрезаются окном контекстной помощи. Слова **Cluster** (Кластер) и **String** (Строка) необязательны, потому что тип данных можно определить по стилю отрезка проводника как на блок-диаграмме, так и в окне контекстной помощи. Сокращения **Com** (Communications) и **Params** (Parameters) общеприняты. Но имена эле-

ментов управления и индикаторов не должны совпадать, в противном случае соответствующие узлы свойств и локальные переменные отличить друг от друга будет сложно.

На рис. 5.19б эти недостатки исправлены: использованы сокращения и добавлены предлоги **in** и **out** для входных и выходных данных соответственно. Соответствующие метки терминалов краткие, уникальные и понятные. Как мы уже обсуждали в главе 3, эти свойства очень важны. Необходимо подчеркнуть, что я не сторонник сокращений, только если они не являются общепринятыми или очевидными, как в этом случае. Также я добавил стандартный кластер ошибок для сохранения потока данных.

Рис. 5.19а. Длинные имена терминалов этого ВП обрезаются в окне контекстной помощи

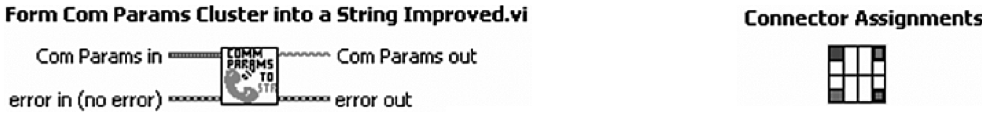


Рис. 5.19б. Имена сокращены, добавлен кластер ошибок для сохранения потока данных

На рис. 5.20 приведен пример иконки и соединений Экспресс-ВП сбора данных (DAQ Assistant) в режиме отображения иконки (выбрать пункт View as Icon из контекстного меню). В этой конфигурации входные данные расположены справа, а не слева. Из-за этого возникают пересечения проводников как в окне контекстной помощи, так и на блок-диаграмме.

На рис. 5.21а приведена иконка и соединительная панель ВП Confirm Quit (Подтвердить выход). На иконке расположен стандартный символ Windows, который часто возникает при завершении приложения. Глиф большой и его ни с чем не перепутаешь, но иконка не отражает тип сообщения. Из данных ВП возвращает только одно значение – из среднего разъема шаблона 3×3. На рис. 5.21б приведена исправленная версия ВП: добавлены кластеры ошибок, шаблон изменен на 4×2×2×4, на иконку добавлен текст. Текст нужен, чтобы указать тип сообщения: глиф уменьшен и добавлено слово **quit?** шрифтом размера 9. Кластеры ошибок нужны, чтобы установить порядок исполнения ВП с помощью потока данных. Стандартный шаблон 4×2×2×4 позволяет избежать изгибов проводников при размещении ВП. В будущем этот ВП можно дополнить: например, строкой сообщения и его типом.

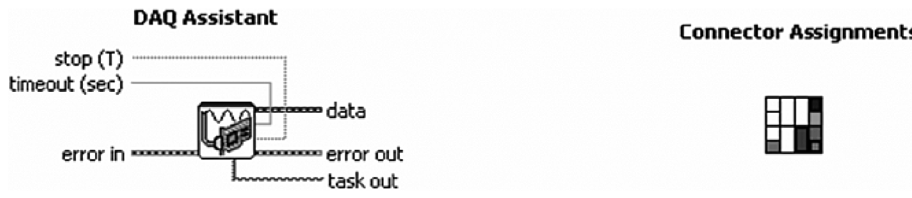


Рис. 5.20. Из-за того, что входные параметры Экспресс-ВП DAQ Assistant расположены справа, на блок-диаграмме вызывающего ВП возникнут пересечения проводников

Рис. 5.21а. На иконке ВП Confirm Quit расположен стандартный символ из диалогов Windows. По иконке нельзя судить о типе сообщения

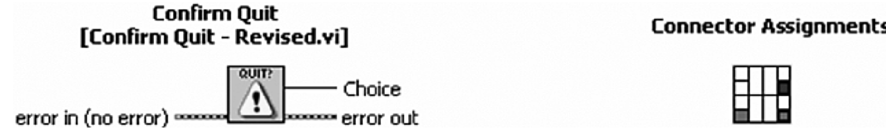


Рис. 5.21б. По исправленной иконке можно понять тип сообщения. Используется стандартный шаблон 4×2×2×4 и кластеры ошибок, чтобы установить порядок исполнения ВП

5.3.4. Показательные примеры

Иконка прибора для печати ВП в альбомном режиме (Print VI in Landscape mode), показанная на рис. 5.22 на 3 четверти состоит из графики и 1 четверть занимает текст. На иконке изображена панель графического интерфейса и надпись **Print VI** (Напечатать ВП). Шаблон соединительной панели стандартный: 4×2×2×4. Иконка отлично выглядит в цвете, очень аккуратная, недостатков нет.

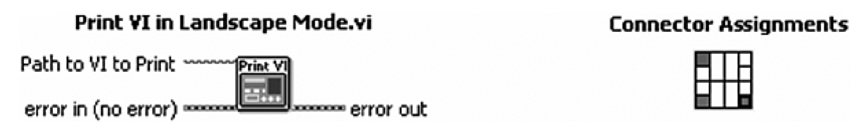


Рис. 5.22. Иконка ВП Print VI in Landscape Mode (Напечатать ВП в альбомном режиме) на s состоит из графики и на j из текста

Иконка ВП Dynamic VI Path Builder (Динамическое формирование пути) (рис. 5.23) на 2/3 состоит из отличного рисунка, оставшуюся треть занимает надпись **Dynamic VI Path Builder** (Динамическое формирование пути к файлу). На рисунке иконки изображена развилка желтой дороги на зеленом поле. Эту иконку полностью оценить можно только в цвете. Обратите внимание, что опять же слово path в английском языке означает сразу и дорогу, и путь к файлу. В исходном файле шаблон соединительной панели 3×3, один входной параметр и один выходной. Предпочтительнее использовать стандартный шаблон 4×2×2×4 и добавить кластеры ошибок (рис. 5.23, справа). Обработка ошибок обсуждается в главе 7.

Рис. 5.23. Иконка ВП формирования пути к файлу на 2/3 состоит из очевидной графики и 1/3 занимает текст

Следующие 2 примера демонстрируют использование специальных инструментов для создания иконок с нестандартным размером и формой. Граница иконки в LabVIEW определяется крайними **не белыми** пикселями. Необходимо отметить, что все терминалы входных и выходных параметров должны быть расположены в пределах этой иконки. Терминалы, выходящие за границы ВП, на блок-диаграмме вызывающего ВП не видны. Поэтому шаблон соединительной панели нужно выбирать таким, чтобы терминалы были расположены внутри иконки. Чтобы увидеть расположение терминалов при создании иконки, воспользуйтесь соответствующей командой: **Show Terminals**, в этом режиме можно легко нарисовать иконку с учетом расположения терминалов.

В ВП очистки ошибок (Clear Error All or Specified) (рис. 5.24) форма ВП нестандартная, а шаблон соединительной панели 5×3×3×5, повернутый на 90°. Этот

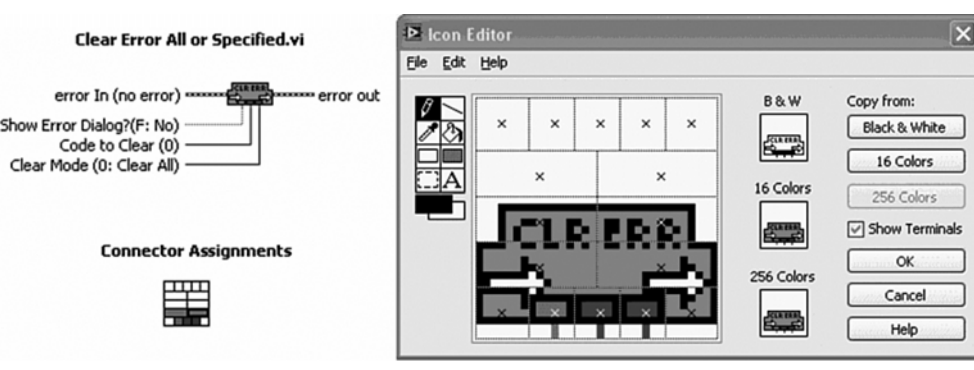


Рис. 5.24. Форма иконки у ВП очистки ошибок (Clear Error All or Specified) нестандартная, шаблон соединительной панели 5×3×3×5, повернутый на 90°

шаблон дает возможность использовать 9 терминалов в пределах иконки, задействовано 5. Повернуть соединительную панель на 90° можно с помощью пункта контекстного меню соединительной панели **Rotate 90 Degrees**. Эта функция подробнее обсуждается в главе 7.

Форма иконки ВП на рис. 5.25 также отличается от стандартной. Цель этого ВП – приостановить исполнение на заданное количество миллисекунд, не используя структуру последовательности. Как мы знаем, это признак хорошего стиля. Благодаря небольшому размеру этот ВП легко поместится в любом месте потока данных и не повлияет на ближайшие проводники. Шаблон соединительной панели – 5×3×3×5, используется 3 терминала.

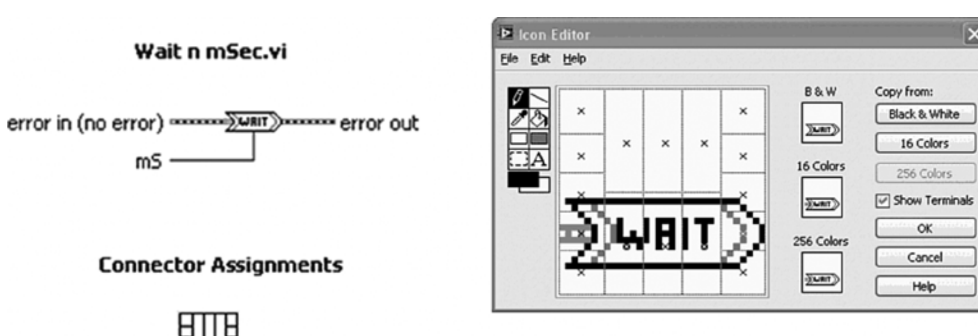
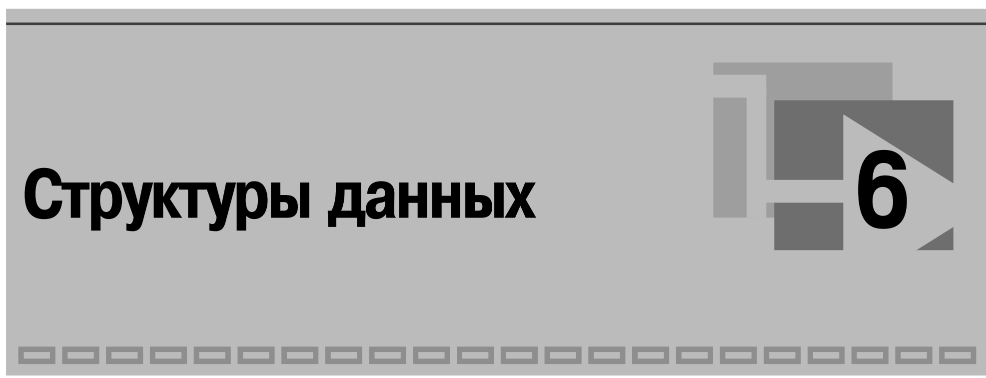


Рис. 5.25. ВП задержки на заданное число миллисекунд – пример хорошего стиля. Шаблон его соединительной панели – 5×3×3×5, используется 3 терминала

Ссылки

1. Цветные версии иллюстраций к главе 5 можно скачать с сайта издателя английской версии книги: www.prenhallprofessional.com/title/0131458353. Цветную электронную версию книги можно купить по адресу www.prenhallprofessional.com/title/0132414813.
2. Руководство по созданию драйверов приборов (NI Instrument Driver Guidelines) находится в зоне разработчиков NI (NI Developer Zone). Адрес www.ni.com/devzone/idnet/library/instrument_driver_guidelines.htm.
3. Библиотека элементов иконок находится в зоне разработчиков NI (NI Developer Zone) по адресу www.ni.com/devzone/idnet/library/icon_art_glossary.htm.



Структуры данных 6

Структуры данных в приложении состоят из встроенных типов данных LabVIEW и конструкций данных, определенных разработчиком (далее – конструкты). Типы данных – это фундаментальные элементы, изображаемые на блок-диаграмме определенным типом провода или характерным терминалом. Конструкты – это наборы данных, использующие фундаментальные типы данных, например массивы и кластеры. Структуры данных – это и типы данных, и конструкты, используемые приложением. Структуры данных определяются как выбором элементов управления, массивов и кластеров на лицевой панели ВП, так и операциями на блок-диаграмме.

Типы элементов управления определяют то, как пользователь понимает и взаимодействует с лицевыми панелями. Сюда входят и GUI (Graphic User Interface – графический интерфейс пользователя), и ВП – те, с которым непосредственно взаимодействует пользователь, и ВПП, которые видны только разработчикам. Как показано в главе 3 «Стиль лицевой панели», грамотный выбор элементов управления делает лицевую панель интуитивно понятной, дружелюбной к пользователю и надежной. К тому же каждый тип элементов управления поддерживает один или ограниченный набор типов данных. Соответственно, выбор этих элементов разработчиком помогает определить структуры данных приложения.

Конструкты используются для организации данных и уменьшения количества проводников, данных на блок-диаграмме. Последовательное использование конструктов во всем приложении помогает сделать его более простым для понимания и управления, и более эффективном с точки зрения использования памяти. Встроенные структуры данных характеризуются некоторыми особенностями, связанными с производительностью и использованием памяти, которые мы обсудим позже в этой главе.

Теорема 6.1 *Поскольку объем используемой памяти и частота обращений к ней являются принципиальными ограничениями производительности современных вычислительных устройств, то скорость работы приложения обратно пропорциональна к использованию памяти.*

Скорость работы приложения обратно пропорциональна использованию памяти – это общее правило для всего современного программного обеспечения. А именно – объем используемой памяти и частота обращений к ней являются принципиальными ограничениями производительности современных вычислительных устройств. Использование памяти приложением LabVIEW определяется его структурой данных и операциями, совершаемыми над данными. LabVIEW не накладывает ограничений на структуры данных, используемые в приложениях. Поэтому производительность приложения напрямую зависит от выбора структур данных и соответствующих операций, то есть от разработчика. В этой главе приведены правила выбора структур данных, которые позволят создать интуитивно понятное, надежное и эффективное приложение.

6.1. Методология разработки конструкций данных

Оцените структуры данных на стадии проектирования приложения. Опишите данные, необходимые для каждой основной подсистемы, включая графический интерфейс пользователя (GUI), сбор данных, подключение/отключение устройств, анализ, создание отчетов, ввод/вывод файлов, очереди баз данных, связь по сети и между различными приложениями. Изучите все основные источники данных приложений. Создание прототипа лицевой панели – очень полезная техника создания GUI, как уже обсуждалось в главе 2 «Приготовка к хорошему стилю». Кроме того, используйте прототипы лицевых панелей, чтобы определить данные, необходимые для каждого компонента приложения.

Когда начинается написание кода, структуры данных неявным образом определяются выбором элементов управления и индикаторов на лицевой панели и входами/выходами узлов на блок-диаграмме. Создание структур данных для ВП обычно происходит в 3 шага. Первый – выбор элементов управления и типов данных. Второй – настройка свойств выбранных элементов управления. Третий – группировка элементов управления в конструкты, если необходимо. В этой части мы рассмотрим некоторые общие правила, которые помогут нам в этом процессе. В последующих секциях обсуждаются правила, применимые к специфичным типам структур данных в LabVIEW.

6.1.1. Выбор элементов управления и типов данных

Правило 6.1 ***Выбирайте элементы управления, упрощающие работу с панелью***

Термин *простой* имеет несколько смыслов относительно элементов управления и структур данных в LabVIEW. *Простые структуры данных* хранят данные в смежных областях памяти. К таким структурам относятся все скалярные типы данных,

такие как логические, численные, строки, массивы и кластеры, содержащие только упомянутые выше типы данных. *Простые элементы управления* – это те, которые интуитивно понятны и которыми легко управлять, они представляют простые структуры данных. Простые элементы управления обладают свойствами, которые можно сконфигурировать так, чтобы помочь проверить достоверность программного или пользовательского ввода данных. Простые элементы очень надежны и эффективны, когда сконфигурированы правильно.

В какой-то степени типы элементов управления могут быть классифицированы в порядке простоты данных и операций с ними. Например, численные элементы очень понятны и могут быть настроены на точные значения из определенного диапазона и рационально хранятся в памяти согласно представлению. Строковые элементы могут содержать любые текстовые данные, без ограничений, и хранятся как блоки соседних байтов, по 1 байту на символ. Логические элементы максимально фиксированы и допускают значения только ИСТИНА и ЛОЖЬ и хранятся как целые байты. Поэтому из этих трех типов элементов управления логические являются самыми простым, за ним следуют численные и затем – строковые. Таблица 6.1 содержит полный список элементов управления LabVIEW, расположенных примерно по простоте данных и совершению операций над ними. Элементы вверху таблицы самые простые, начиная с логических и численных элементов, внизу же расположены потенциально самые сложные элементы. Заметьте, что список одновременно и перекрывающийся и субъективный. Например, простота численных элементов зависит от выбранной формы представления. Массивы и кластеры – это конструкты, и их простота зависит от содержания. Также строка использует столько же памяти, как и одномерный массив целых байтов такой же длины. Поэтому типы элементов управления примерно организованы по простоте данных. Поскольку массивы потенциально могут содержать более сложные данные, чем строки, то и строковый тип данных расположен как более простой, чем массив.

Таблица 6.1. Элементы управления LabVIEW и их описание, расположенные в порядке возрастания сложности

Логические переменные – кнопки, переключатели и световые индикаторы, которые используются для ввода и изображения логических (ИСТИННО/ЛОЖНО) значений

Кольцевая структура (Ring) – предоставляет возможность выбора из списка элементов (текст, картинки – Picture ring), каждому из которых соответствует численное значение

Перечень (Enumeration) – предоставляет список строк, каждой из которых ставится в соответствие численное значение. Похоже на кольцевую структуру, с той разницей, что численные значения и строки являются частями типа данных. Также используется обозначение enum

Вкладка (Tab) – состоит из многих страниц для элементов управления и графического интерфейса и селектора страниц-вкладок. Каждой вкладке ставится в соответствие текстовая метка и численное значение

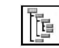
Численное значение (Numeric) – простой метод ввода скалярных численных значений. Можно сконфигурировать под множество различных типов как целых, так и дробных чисел

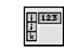
Таблица 6.1. Элементы управления LabVIEW и их описание, расположенные в порядке возрастания сложности (окончание)

Список (Listbox) – представляет собой список строк, связанных с численным значением (когда выбран один элемент из списка) или массивом значений (когда выбрано несколько элементов)

Комбинированный элемент управления (bombo box) – представляет собой список строк, которым в соответствие ставится другой список строк

Строка (String) – используется для ввода и отображения текстовых данных

 Дерево элементов (Tree) – список вложенных элементов с внутренней иерархией

 Массив – группа элементов данных одного типа. Может быть как одномерным, так и многомерным

Матрица – Двумерный массив действительных или комплексных чисел, используется для операций с матрицами, например в линейной алгебре

Таблица –двумерный массив строк, организованный в таблицу со строками и столбцами, ячейки которой содержат текстовые элементы

Кластер – группы элементов данных одного или различных типов

Универсальный тип данных (Variant) – используется для передачи или отображения универсального типа данных, состоящего из имени, типа данных, данных и атрибутов

Контейнер activeX – позволяет разместить элемент activeX на лицевой панели

Контейнер .Net – позволяет разместить элемент управления .Net на лицевой панели

Правило 6.2 Используйте эффективные с точки зрения использования памяти типы данных

Поскольку многие типы элементов управления можно настроить на работу со множеством типов данных, эффективность работы с памятью зависит от комбинации типа элемента управления и типа данных. Поэтому важно учитывать эффективность использования того или иного типа данных наряду с простотой элементов управления. Тип данных определяет размер памяти, выделенный для их хранения, и способ записи данных в память. Например, мы можем выстроить численные типы данных по способу представления данных. Так, целочисленное представление занимает меньше всего памяти и является самым эффективным, тогда как представление в форме числа с плавающей точкой занимает больше всего памяти и является наименее эффективным. Табл. 6.2 содержит перечень всех типов данных, расположенных в порядке эффективности использования ресурсов памяти. Как видно, меньше всего памяти занимает целое число представленное 8 битами (1 байт), а больше всего памяти использует представление числа с плавающей точкой расширенной точности (8 или 16 байт).

По аналогии с табл. 6.1 порядок элементов в табл. 6.2 – примерный и перекрывающийся. Например, Перечень из двух компонент целого числа с неопределенным знаком и набор строковых меток. Использование памяти для записи целого

числа с неопределенным знаком зависит от представления и может составлять 8, 16, 32 бита на число. Объем памяти, выделяемый под строковые метки, зависит от их величины и длины. Все эти свойства настраиваются разработчиком. Как и раньше, наиболее сложные типы данных расположены внизу таблицы, такие как Универсальный и Динамический типы данных, зависящие от атрибутов, которые они содержат.

Таблица 6.2. Типы данных в LabVIEW с описанием, приведенные в порядке возрастания объема использования памяти

Тип	Описание
TF	Логические переменные – сохраняет значение Истина/Ложь как целый байт
U8	8-битное целое число с неопределенным знаком – представляет неотрицательные целые числа в диапазоне от 0 до 255
I8	8-битное целое число с определенным знаком – представляет целые числа в диапазоне от –128 до +127
◂▸	Перечень – представляет выбор из списка текстовых элементов, которым соответствуют 8-, 16-, 32-битные целые
U16	1-битное целое число с неопределенным знаком – представляет неотрицательные целые числа в диапазоне от 0 до 65 535
I16	1-битное целое число с определенным знаком – представляет целые числа в диапазоне от –32 768 до +32 767
U32	32-битное целое число с неопределенным знаком – представляет неотрицательные целые числа в диапазоне от 0 до 4 294 967 295
I32	32-битное целое число с определенным знаком – представляет целые числа в диапазоне от –147 483 648 до +2 147 483 647
SGL	Число с плавающей точкой одинарной точности – использует 4 байта для представления дробных чисел согласно стандарту aNSI/IEEE 754-1985
	Refnum – уникальный идентификатор для таких объектов, как ВП, файл, элемент .Net, или сетевого подключения
I/O	I/O name – уникальный идентификатор для ресурсов ввода/вывода, таких как DAQmx и VISA, для обмена данными с устройствами
U64	64-битное целое число с неопределенным знаком – представляет неотрицательные целые числа в диапазоне от 0 до 18 446 744 073 709 551 615
I64	64-битное целое число с определенным знаком – представляет целые числа в диапазоне от –9 223 372 036 854 775 808 до +9 223 372 036 854 775 807
DBL	Число с плавающей точкой двойной точности – использует 8 байт для представления дробных чисел согласно стандарту aNSI/IEEE 754-1985
EXT	Число с плавающей точкой расширенной точности – использует от 8 до 16 байт для представления дробных чисел согласно стандарту aNSI/IEEE 754-1985
CSG	Комплексное число с плавающей точкой одинарной точности – дробное комплексное число, содержащее действительную и мнимые части, каждая представлена 4 байтами (SGL)
CDB	Комплексное число с плавающей точкой двойной точности – дробное комплексное число, содержащее действительную и мнимые части, каждая представлена 8 байтами (DBL)
CXT	Комплексное число с плавающей точкой расширенной точности – дробное комплексное число, содержащее действительную и мнимые части, каждая представлена от 10 до 16 байтами (EXT)

Таблица 6.2. Типы данных в LabVIEW с описанием, приведенные в порядке возрастания объема использования памяти (окончание)

Тип	Описание
	<64.64>-битная временная отметка – сохраняет абсолютное значение системного времени с очень большой точностью. На самом деле, сохраняет число секунд прошедших с 12:00 пятницы 1 января 1904 года по Гринвичу, используя 16 байт
[DBL]	Действительная матрица – двумерный массив чисел с плавающей точкой двойной точности, сохраняется как определение типа данных. Используется с математическим функциями, как в линейной алгебре
DBL	Комплексная матрица – двумерный массив чисел с плавающей точкой двойной точности, сохраняется как определение типа данных. Используется с математическим функциями, как в линейной алгебре
abc	Строка – последовательность aSbII кодов, записанных как одномерный массив байт. Может содержать как цифры, так и буквы, как печатные, так и непечатные
	Путь – содержит адрес файла или директории, использует синтаксис ОС
	Картинка – содержит набор инструкций и данные для отображения картинок в LabVIEW
0101	Цифровые данные (rigital data) – содержит данные цифрового сигнала
	rigital waveform – структура данных, содержащая начальное время (t0) как временную отметку, временной интервал между отсчетами (dt) как число с плавающей точкой удвоенной точности и цифровой волновой сигнал (Y) как цифровые данные (digital data)
	Waveform (Волновой сигнал) – структура данных, содержащая начальное время (t0) как временную отметку, временной интервал между отсчетами (dt) как число с плавающей точкой удвоенной точности, цифровой волновой сигнал (Y) как одномерный массив любого численного типа и атрибуты
	Динамический тип данных – представляет собой данные сигнала и его атрибуты как массив аналоговых волновых сигналов, используется только вместе с Express ВП
	Универсальный тип данных – кодирует имя, тип данных, сами данные и атрибуты в обобщенный тип данных. Используется с activeX, DataSocket и ВПП, требует фиксированного интерфейса для обработки множества данных

Выбирайте элементы управления и тип данных в соответствии с эффективностью использования памяти и легкостью работы. Табл. 6.3 представляет матрицу, содержащую элементы управления и поддерживаемые типы данных. Типы элементов управления расположены вверху по горизонтали, как если бы табл. 6.1 повернули боком. А типы данных расположены по вертикали, как в табл. 6.2. Ячейки таблицы отражают совместимость элементов управления и типов данных. Используйте табл. 6.3 для оптимизации выбора элементов управления и типов данных. Подумайте, какой тип данных вам необходим, и по таблице выберите соответствующий элемент управления. Начните в левого верхнего угла и оцените каждый элемент управления слева направо. Если ваш элемент управления имеет два противоположных положения, выбирайте Логический элемент. Если вам нужно задать диапазон чисел по выбору, сначала рассмотрите Перечень (enum), затем Кольцевую структуру (ring), а затем Число (numeric). Выбирайте первый

тип элементов управления, подходящий по параметрам. Затем просмотрите соответствующую колонку поддерживаемых типов данных, снизу вверх, пока не найдете первый тип данных, который обеспечивает нужную функциональность. В результате вы получите комбинацию элемента управления и типа данных, обеспечивающую простоту исполнения и наибольшую эффективность использования памяти.

Обратите внимание, что использование Правил 6.1 и Правил 6.2 вместе с табл. 6.3 помогает избежать создания некорректных данных в приложении. Например, вы никогда не выберите строковый элемент управления для данных, которые являются строго численными. И численный элемент управления является гораздо более простым и надежным в использовании, кроме того, память используется эффективнее. Также, если у вас есть дискретный набор текстовых (буквы и цифры) данных, используйте Перечень (enum) или Кольцевую структуру (ring), прежде чем взяться за строковые переменные. Как видно из табл. 6.3, Перечень и Кольцевая структура являются более простыми элементами, потому что сводят вводимые пользователем данные к ограниченному набору дискретных позиций, которые, в свою очередь, представлены целыми числами.

Наконец, заметьте, что Перечень и Условный тип данных появляются в каждой таблице и как элементы управления, и как тип данных. Это действительно так. Перечень – это специальный тип данных, состоящий из чисел и связанных с ними текстовых строк. Условный тип данных – это тип данных, который описывает сам себя, перекодируя любые данные LabVIEW в обобщенный формат. Аналогично, массивы и кластеры могут рассматриваться и как элементы управления, и как типы данных. Однако тип данных в массиве или кластере не определен до тех пор, пока они не заполнены соответствующими структурами данных. В отличие от Перечня и Условного типа данных, передача данных от терминалов на блок-диаграмме невозможна до тех пор, пока не определен тип данных в массиве или кластере. Иначе проводники данных выглядят разорванными. Поэтому, массивы и кластеры не становятся действительными типами данных, пока не заполнены.

Правило 6.3 Используйте те элементы управления и типы данных, которые облегчат создание согласованных структур во всем приложении

Согласно теореме 6.1, самое важное в выборе элементов управления и типов данных – последовательность. Различные типы данных требуют конвертирования: явно – через функции форматирования и конвертирования или автоматически – через точки или терминалы приведения типов данных. LabVIEW выделяет различные области памяти, чтобы хранить каждое представление данных. Поэтому использование различных типов данных приводит к дополнительному программированию, счету, выделению и потреблению памяти. Хорошие программисты LabVIEW стремятся оптимизировать производительность и использование

памяти в своих приложениях. Последовательные типы данных снижают усилия, требуемые для программирования, и, кроме того, повышают надежность и производительность.

Нужно учитывать сразу Правила 6.1, 6.2 и 6.3, когда разрабатываете структуры данных. Требуются элементы управления, которыми просто оперировать, типы данных, которые эффективно используют память, объединенные в непротиворечивые структуры данных. Как же понять, что мы учли все соображения? Тщательно изучите все требования к данным для каждого компонента или ВПП, и выберите те типы данных, которые проще, эффективнее и не противоречат друг другу. В случае если неизбежен компромисс между эффективностью (Правило 6.2) и непротиворечивостью (последовательностью) структуры данных (Правило 6.3), всегда выбирайте непротиворечивость (Правило 6.3). По теореме 6.1, время доступа к памяти является универсальной величиной. Чем больше операций с памятью совершают наши приложения, тем сильнее негативное влияние на производительность. Все операции конвертирования типов существенно сказываются на конечном результате. Однако большие типы данных, такие как DBL, (в отличие от I16) необязательно влияют на размер выделенной памяти. В самом деле, выделение памяти для DBL-массива из 4,096 элементов может привести к такому же времени обращения к памяти, что и выделение памяти для аналогичного I16-массива, тогда как выделение памяти под массив I16 и последующее конвертирование его в DBL-формат повлечет за собой множественное выделение памяти под одни и те же данные.

Рассмотрим пример на рис. 6.1. Драйвер осциллоскопа получает и обрабатывает волновой сигнал с помощью набора ВПП, как показано на рис. 6.1а. На нижнем уровне иерархии расположен ВП Get Raw I16 Waveform, показанный на рис. 6.1б. Этот ВП считывает исходные данные как бинарную строку, конвертирует данные в массив целых чисел (без указания знака), комбинирует два последовательных целых числа в одно 16-битное целое число (с указанием знака) и возвращает исходные данные как массив 16-битных целых чисел (с указанием знака). Поскольку обычно один отсчет описывается 2 байтами, то согласно табл. 6.3 1-битное целое число является наиболее простым и эффективным представлением таких данных.

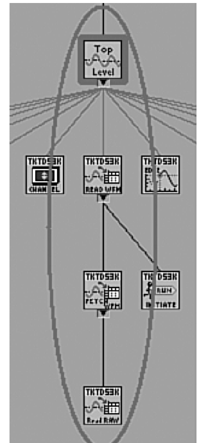
На рис. 6.1в ВП Fetch Waveform преобразует исходные данные, полученные от ВП Get Raw Waveform, в масштабированный массив с помощью нескольких арифметических функций. Поскольку факторы масштаба и отступа записаны в форме числа с плавающей точкой двойной точности, массив исходных данных необходимо перевести в такой же формат перед математическими операциями. Это отражает точка приведения типов данных на функции выделения. В этой точке создается новый буфер данных, расширенный до 8 байт на элемент массива, вдобавок к уже существующему. Таким образом, приведение типов данных увеличивает использование памяти в 4 раза помимо выделения памяти под новый буфер данных.

Затем отмасштабированный массив передается через ВП Read Waveform без изменений, как показано на рис. 6.1г. Наконец, ВП верхнего уровня, показанный

Таблица 6.3. Таблица совместимости элементов управления и типов данных в LabVIEW

		Логическое	Кольцевая структура	Перечень	Вкладка	Численное значение	Список	Комбинированный элемент управления	Строка	Дерево элементов	Массив	Матрица	Таблица	Кластер	Универсальный тип данных	Контейнер activeX	Контейнер .Net
Логические	TF	+								+			+	+			
8-битное целое число без знака	U8		+	+		+					+			+	+		
8-битное целое число со знаком	I8		+			+					+			+	+		
Перечень			+	+	+	+					+			+	+		
16-битное целое число без знака	U16		+	+		+					+			+	+		
16-битное целое число со знаком	I16		+			+					+			+	+		
32-битное целое число без знака	U32		+	+	+	+					+			+	+		
32-битное целое число со знаком	I32		+			+	+				+			+	+		
Число с плавающей точкой одинарной точности	SGL		+			+					+			+	+		
Ссылка (Refnum)											+			+	+	+	+
Имя ресурса (I/O name)	I/O										+			+	+		
64-битное целое число без знака	U64		+	+		+					+			+	+		
64-битное целое число со знаком	I64		+			+					+			+	+		
Число с плавающей точкой двойной точности	DBL		+			+					+	+		+	+		
Число с плавающей точкой расширенной точности	EXT		+			+					+			+	+		
Комплексное число с плавающей точкой одинарной точности	CSG					+					+			+	+		
Комплексное число с плавающей точкой двойной точности	CDB					+					+	+		+	+		
Комплексное число с плавающей точкой расширенной точности	CXT					+					+			+	+		
64-битный интервал времени											+			+	+		
Действительная матрица	[DBL]													+	+		
Комплексная матрица	DBL													+	+		
Строка	abc							+	+	+	+		+	+	+		
Путь											+			+	+		
Картинка											+			+	+		
Цифровые данные	0101										+			+	+		
Цифровая осциллограмма											+			+	+		
Осциллограмма (Waveform)											+			+	+		
Динамический тип данных											+			+	+		
Универсальный тип данных															+		

на рис. 6.1д, соединяет отмасштабированный массив, **Initial X** (Начальное значение X) и **X Increment** (Инкремент по X) и обновляет график. Функция соединения копирует массив в кластер, который она создает, что также требует выделения дополнительной памяти.



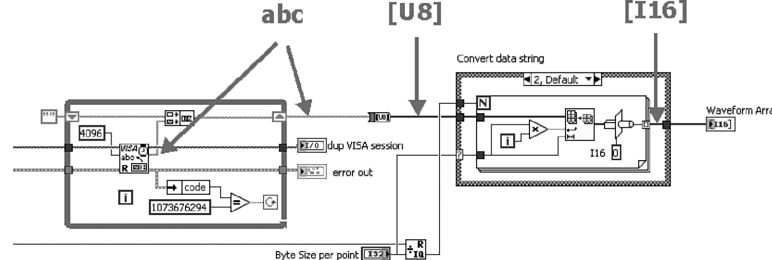
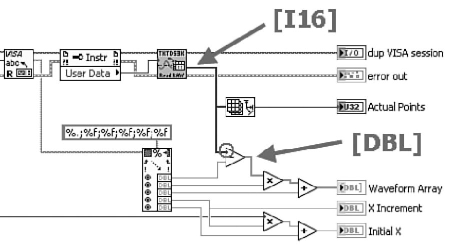


Рис. 6.1б. ВП Get Raw I16 Waveform возвращает массив 1-битных целых чисел с указанием знака. Это наиболее эффективный тип данных для хранения волнового сигнала в этом ВП

Рис. 6.1а. Секция окна иерархии для ВП верхнего уровня, которая обращается к драйверу инструмента и получает волновой сигнал с осциллоскопа. Драйвер инструмента использует вызов цепочки из трех ВПП, включая чтение, выборку и извлечение



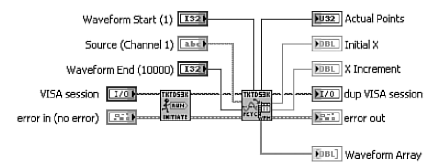


Рис. 6.1г. ВП Read Waveform передает отмасштабированный массив без изменений

Рис. 6.1в. ВП Fetch Waveform масштабирует исходный массив в массив чисел с плавающей точкой удвоенной точности. Арифметические операции с разными типами данных вызывают приведение типов данных, что влечет выделение нового буфера памяти

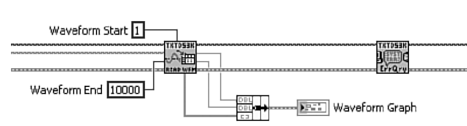


Рис. 6.1д. ВП высшего уровня объединяет масштабированный массив и другие компоненты волнового сигнала в кластер и обновляет график

Каждый ВП в этой цепочке обладает различными требованиями к данным, и элементы управления и типы данных были выбраны с тем, чтобы обеспечить максимальную простоту и эффективность для каждого ВП. Однако различные типы данных, используемые в этой цепочке, приводят к увеличению используемой памяти в 9 раз по сравнению с изначальным массивом 1-битных целых чисел. Более важно то, что в этой цепочке тип данных волнового сигнала конвертируется 5 раз, что добавляет необязательные буферы и снижает скорость выполнения приложения. Таким образом, использование самых простых и эффективных типов данных внутри каждого ВП *снижает* общую производительность приложения из-за несогласованных структур данных.

Рисунок 6.2 содержит тот же самый пример, за исключением того, что на протяжении всей цепочки вызываемых функций используются согласованные типы данных. На рис. 6.2а каждые 2 последовательных байта бинарной строки конвертируются непосредственно в массив чисел с плавающей точкой удвоенной точности. Заметьте, что при таком выборе типа данных требуется выделить в 4 раза больше памяти, чем на 1-битные целые числа, как на рис. 6.1а, но это улучшает согласованность цепочки вызываемых функций. Также обратите внимание, что промежуточное конвертирование бинарной строки в 8-битные целые числа было уничтожено, частично за счет выделения дополнительной памяти и вычислений, требуемых для перевода в DBL-формат. На рис. 6.2б исходные данные масштабируются с использованием арифметических операций, применяемых к согласованным типам данных, тем самым устраняется приведение типов и выделение дополнительной памяти в отличие от рис. 6.1б. Так же монтируется тип данных волнового сигнала. На рис. 6.2в волновой сигнал передается через ВП **Read Waveform** без изменений. Наконец, на рис. 6.2г волновой сигнал передается в ВП верхнего уровня без изменений, и график обновляется.

В этом примере согласованные типы данных значительно уменьшают использование памяти и уменьшают число выделенных буферов. Таким образом, улучшается общая производительность. Кроме того, волновой сигнал является удобной структурой данных, которая требует меньше терминалов и проводников данных

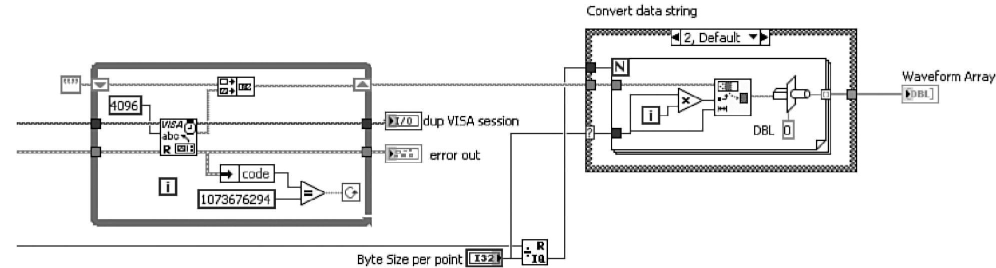


Рис. 6.2а. ВП Get Raw DBL Waveform возвращает массив чисел DBL, так достигается компромисс между эффективностью внутри этого ВП и согласованностью всей структуры данных. Так же устраняется промежуточное преобразование

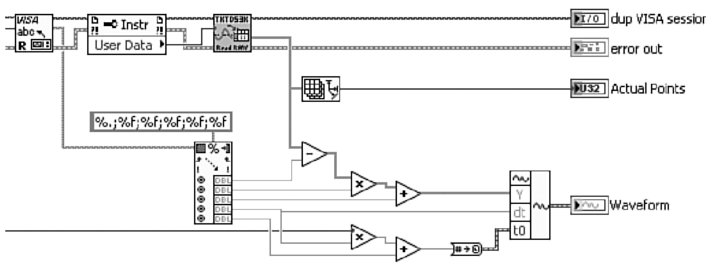


Рис. 6.2б. ВП Fetch Waveform осуществляет арифметические операции, используя похожие типы данных, избегая приведения типов данных и создавая структуру волнового сигнала

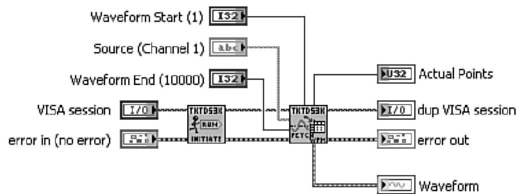


Рис. 6.2в. ВП Read Waveform передает волновой сигнал без изменений

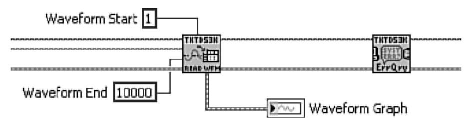


Рис. 6.2г. ВП верхнего уровня просто обновляет график. Необходимости в изменении данных нет

для множества ВП в цепочке. Когда структура волнового сигнала сформирована, разработчик может использовать ВП высокого уровня (расположенные на вкладке **Waveform**) для осуществления обычных операций с сигналами. Так, правила 6.1, 6.2 и 6.3 могут быть скомбинированы для улучшения общей производительности.

Следует отметить, что в некоторых приложениях исходный волновой сигнал предпочтительнее представить в наименьшем формате вместо формата DBL или волнового сигнала. На пример, если наша цель – получить и вывести исходные данные в файл, то нас в первую очередь будет интересовать наименьший размер файла и наибольшая скорость записи на диск. В этом случае оптимальным будет писать в файл бинарную строку исходных данных, полученную прямо с инструмента. Убедитесь, что записываете информацию о масштабе волнового сигнала, чтобы конвертировать данные в желаемые физические единицы.

6.1.2. Настройка свойств

После того как выбраны элементы управления, следующий шаг – настроить их свойства. Для булиновых кнопок обычно настраивается Механическое Действие и текстовые свойства. Для численных элементов управления настраивается формат, точность, диапазон. Для строк можно определить тип отображения, одну строку и поведение при переносе. Настраивайте внешний вид и значение по умолчанию для каждого элемента управления или создавайте собственные элементы управления, используя **Control Editor**.

Правило 6.4 Задайте соответствующее значение по умолчанию для каждого элемента управления

Чтобы помочь в развитии правильного использования ВП, настройте соответствующие значения по умолчанию для каждого элемента управления. Рассмотрите все обычные случаи использования элемента, определите типичное значение, которое может использовать элемент, и настройте его как значение по умолчанию, если возможно. ВП должен, по крайней мере, загружаться из памяти и работать правильно без изменения значений элементов управления. Требуемые входные данные, вроде refnum или сессий подключения инструментов, являются исключениями. Для ВПП следует отображать значение по умолчанию в скобках, в конце лейбла, так чтобы оно было видно с блок-диаграммы и в окне контекстной справки LabVIEW.

Правило 6.5 Вводите описания элементов управления

Как уже обсуждалось в главе 3, понятные и лаконичные метки помогают документировать элементы управления и индикаторы. Кроме этого, добавьте одно или два предложение к метке каждого элемента, где бы указывались его цель, значение по умолчанию, диапазон, если только элемент не является базовым. Документация элементов управления создает описание, видимое в окне контекстной справки LabVIEW. В главе 9 «Документация» описано, как создать документацию, содержащую описание всех элементов управления и их лейблы. Следует писать описания, даже когда это, казалось бы, не нужно, иначе можно сильно пожалеть, когда понадобится формальный документ.

Правило 6.6 Сохраняйте собственные элементы управления как строгие определения типов (strict type def., далее – тайпдеф)

Строгое определение типов – это элемент управления, который настроен и сохранен как **Strict Type def.** в окне **Control Editor** (Настройка элемента управле-

ния). Control Editor формирует CTL-файл, который сохраняет информацию о типе данных и свойствах тайпдефа. Это позволяет многим копиям поддерживать одинаковые свойства, в том числе внешний вид и поведение. Если затем свойства изменяются с помощью Control Editor, то изменения автоматически применяются ко всем копиям. Строгое определение типов в деталях обсуждается в разделе 3.4.2 «Ваш стиль». В общем, если у элемента управления есть одно свойство или более, которое было сконфигурировано и будет больше одной копии элемента, требующей таких же свойств, сохраняйте его как тайпдеф.

6.1.3. Создание конструктов данных

После того как элементы управления и их свойства определены, остается только создать конструкты данных. В основном это осуществляется путем группировки связанных элементов в массивы и кластеры и сохранении их как тайпдефов. Массивы – это наборы данных одного типа. Кластеры содержат множество данных различных типов. Массивы и кластеры представлены одним терминалом и проводником данных на блок-диаграмме. Они очень полезны для организации данных и уменьшения числа терминалов ВПП.

Правило 6.7 Создавайте массивы и кластеры, которые объединят связанные элементы

Рассмотрим ВП Torque Hysteresis, показанный на рис. 6.3. Сначала приложение позволяет пользователю ввести некоторую информацию о тестируемом элементе (UUT). Это приложение выстраивает в очередь базу данных и, если обнаруживает тестируемый элемент, позволяет пользователю ввести набор параметров, управляющих движением. На следующем шаге ВП запускает тест, состоящий из снятия профиля движения (момент вращения от угла поворота). После завершения теста ВП осуществляет статистический анализ данных и создает отчет. Наконец, ВП записывает данные в файл. Следуя советам из главы 4 «Блок-диаграмма», эти советы применяются при создании модульной диаграммы, содержащей отдельные ВПП для каждой задачи, как в этом примере: UUT Information Dialog (1), Motion Parameters Dialog (2), ВП Run Test (3), ВП Compute Statistics (4), ВП Generate Report (5) и ВП Save Data (6). Упрощенная версия диаграммы, содержащая эти шесть ВПП, приведена на рис. 6.3а в порядке введения структур данных. Данные создаются с помощью первых четырех ВПП и передаются последующим двум, как показано на рис. 6.3б.

Показанная на рис. 6.4а диаграмма ВП Torque Hysteresis построена без использования кластеров. Требуется много проводников данных и терминалов, чтобы передать данные от первых трех ВПП к последующим. Следующий ВПП, ВП Compute Statistics, создает шесть дополнительно вычисляемых значений, которые используются в ВП Generate Report и ВП Save Data. Однако следующие два ВПП не обладают достаточным количеством терминалов, чтобы принять эти дан-

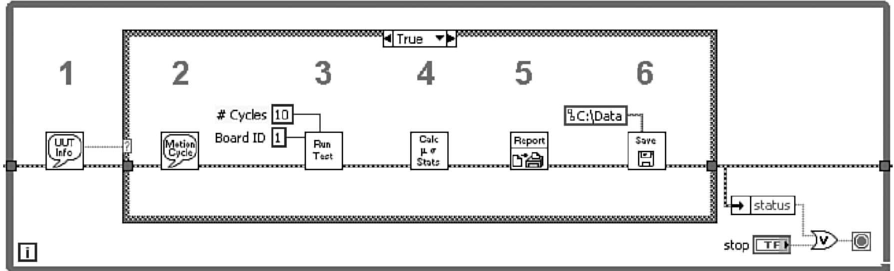


Рис. 6.3а. Блок-диаграмма ВП Torque Hysteresis

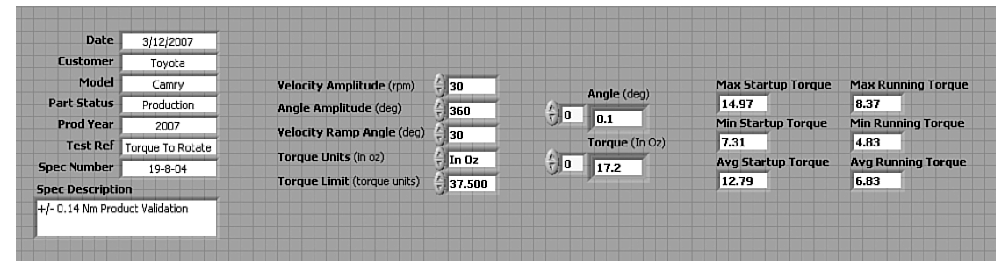


Рис. 6.3б. Данные, созданные с помощью первых четырех ВПП и переданные при помощи ВП Generate Report и ВП Save Data

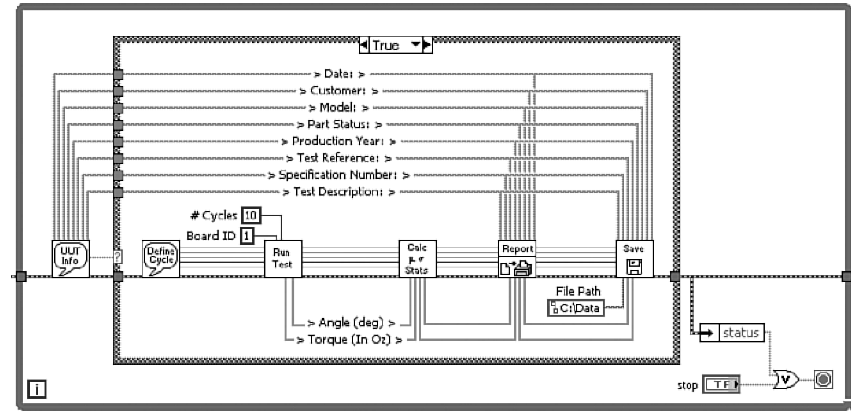


Рис. 6.4а. Блок-диаграмма ВП Torque Hysteresis, выполненная без использования кластеров. Плотность проводников данных и терминалов ВПП затрудняет техническую поддержку

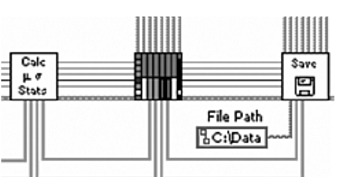


Рис. 6.4б. И хотя ВП Generate Report имеет максимальное число (28) терминалов на соединительной панели, этого недостаточно для еще 6 дополнительных значений, вычисленных в ВП Compute Statistics

ные по проводникам данных, несмотря на использование соединительной панели с максимальным числом (28) терминалов, как показано на рис. 6.4б. Вместо этого используются глобальные переменные для передачи данных между ВПП. Кроме того, техническая поддержка этого ВП – рутинная работа. Если добавить дополнительный параметр к информационному диалогу UUT, то потребуется добавить соответствующие элементы управления, проводники данных, терминалы к трем ВПП или задействовать дополнительные глобальные переменные.

Применяя Правило 6.7, сгруппируем данные с рис. 6.3б в кластеры, как показано на рис. 6.5а. Блок-диаграмма ВП Torque Hysteresis, выполненная с помощью кластеров, показана на рис. 6.5б. Восемь параметров информации о тестируемом элементе собраны в кластер **UUT Information**. Пять контрольных параметров движения хранятся в кластере **Motion Parameters**. Этот кластер передается в ВП Run Test. После завершения теста ВП сохраняет исходные данные, полученные

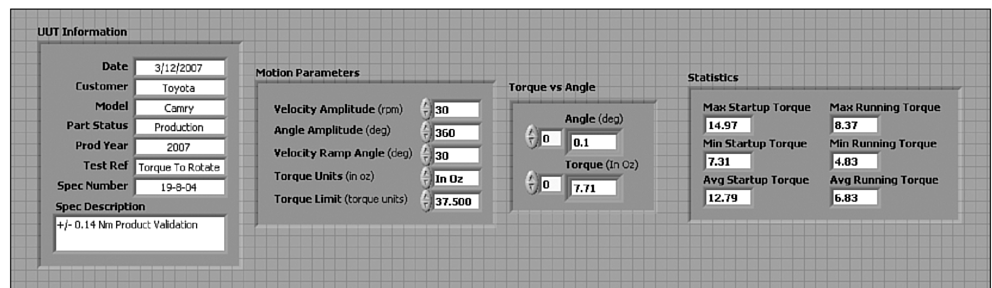


Рис. 6.5а. 21 параметр с рис. 6.3б сгруппирован в 4 кластера

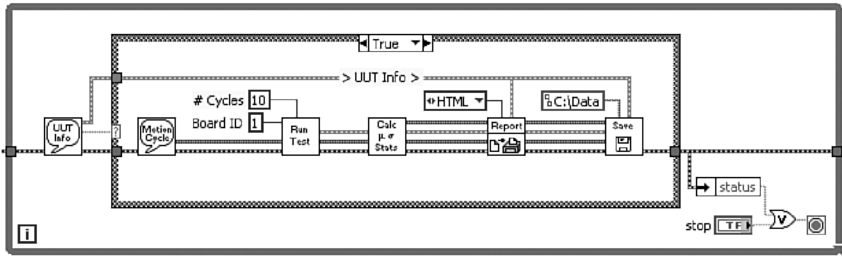


Рис. 6.5б. Блок-диаграмма ВП Torque Hysteresis, выполненная без использования кластеров. Диаграмма выглядит более аккуратной и упорядоченной, поддержка упрощена

в ходе тестирования, в кластер **Torque vs Angle**. Далее оба кластера (**Motion Parameters** и **Torque vs Angle**) передаются в ВП Compute Statistics. Вычисленные результаты записываются в кластер **Statistics**. Так же ВП Compute Statistics передает кластеры **Motion Parameters** и **Torque vs Angle** без изменений в ВП Generate Report. Этот ВП получает кластер **UUT Information** от соответствующего диалогового окна и создает отчет. Наконец, все кластеры передаются в ВП Save Data, где они записываются на диск.

В этом примере 21 параметр сгруппирован логичным образом в 4 кластера. Все данные передаются между ВПП с использованием терминалов и проводников данных. Все ВПП имеют стандартную 4×2×2×4 соединительную панель. Блок-диаграмма выглядит чистой и упорядоченной благодаря использованию кластеров. Каждый кластер сохраняется как тайпдеф для простоты технической поддержки. Добавлять и удалять параметры легко – просто добавляя или удаляя элементы управления в тайпдефе.

В следующих разделах содержатся советы по использованию простых типов данных, простых и сложных конструктов данных наряду с еще большим количеством примеров.

6.2. Простые типы данных

Простые типы данных – это часть фундаментальных типов данных в LabVIEW, которые хранятся в последовательных ячейках памяти. К таким типам данных относятся логические переменные, числа, строки, путь (к файлу), картинки и несколько специфичных типов численных данных. В этой секции мы рассмотрим правила стиля относительно простых типов данных.

6.2.1. Логические переменные

Логические переменные являются простейшим типом элементов управления как в смысле операций над ними, так и в смысле простоты данных – эти переменные могут принимать только 2 значения: TRUE (ИСТИНА) и FALSE (ЛОЖЬ). На самом деле логические данные хранятся как целый байт. Поэтому и, в самом деле, нет ничего более эффективного, чем логическая переменная, как наименьшее представление целого в терминах занимаемой памяти. Однако логические элементы управления очень наглядны. Для них существует два различных состояния, напоминающих объекты нашей повседневной жизни, такие как кнопки, переключатели, тумблеры, светодиоды и т.д.

Правило 6.8 *Используйте логические переменные, если два состояния логически противоположны*

Наиболее важное правило, касающееся логических переменных, состоит в том, чтобы использовать их для представления параметров, которые имеют точно два

логически противоположных значения – например, «Включено» или «Выключено», «Да» или «Нет», «Открыто» или «Закрыто», «Стой» или «Иди». Если два состояния не являются противоположными, используйте текстовую кольцевую структуру или перечень элементов.

Правило 6.9 Присваивайте имена, которые идентифицируют поведение значений TRUE и FALSE

Когда состояния параметра, в самом деле, противоположны, становится легко присвоить элементу имя, которое бы однозначно определяло его поведение в соответствии со значением TRUE/FALSE. Например, если у нас есть элемент с именем **Valve State** (Состояние клапана), то его поведение непонятно. Вместо этого можно установить имя элемента **Close Valve** (Клапан закрыт). В этом случае мы можем определить из имени элемента, что клапан закрыт, когда значение – TRUE, и открыт, когда – FALSE. Согласно правилу 3.23, всегда вводите значение по умолчанию для ВПП. Это устранит неясность. В последующих примерах у нас элемент называется **Close Valve** (Клапан закрыт) (F = Open).

Правило 6.10 Используйте кнопки управления для действий, ползунковый переключатель – для установок параметров

Используйте кнопки управления для представления логических переменных, которые влекут за собой немедленные действия в графическом интерфейсе ВП, такие как **Run** (Запуск), **Cancel** (Отмена), **Quit** (Выход) и **Close Valve** (Закрыть клапан). Начните с простых кнопок **OK**, **Cancel** (Отмена) или **Stop** (Стоп) на вкладке логических переменных и настройте их под свои нужды. Вы можете изменять текст или размер, шрифт надписи и цвет. Не стоит использовать кнопки управления для случаев, которые не влекут за собой немедленных действий, таких как настройка свойств и т.д. Лучше всего использовать такие элементы управления, которые будут наиболее понятны пользователю. NI рекомендует использовать вертикальные ползунковые переключатели для всех логических элементов управления, используемых внутри драйвера инструмента. В большинстве приложений можно обойтись всего двумя типами Логических элементов: кнопками управления для действий на панели графического интерфейса ВП и вертикальными ползунковыми переключателями для параметров конфигурации.

Правило 6.11 Отмечайте состояния TRUE/FALSE для ползунковых переключателей и тумблера

Всегда помечайте состояния TRUE/FALSE для ползунковых переключателей и тумблеров именами, описывающими поведение переключателя в каждом состоянии, метки расположите соответственно физическим положениям переключателей. Для этого в меню быстрого вызова элемента управления выберите **Advanced** ⇒ **Customize**, чтобы открыть окно настройки. Разместите метки по соседству с каждым положением переключателя и введите текст, описывающий его поведение в состояниях TRUE/FALSE. Настройте шрифт, положение и ориентацию меток так, чтобы они были вровень с положениями TRUE/FALSE переключателя. Закройте окно настройки и примените сделанные изменения. Таким образом, вы сделаете метку частью элемента и не сможете случайно сдвинуть ее или удалить в процессе работы с приложением. Другой, более быстрый, способ состоит в том, чтобы сделать видимым текст элемента, для этого выберите **Visible Items** ⇒ **Boolean Text** в меню быстрого вызова. Однако такой способ показывает только одно состояние за раз и не является таким наглядным, как присвоение стационарных меток состояний TRUE/FALSE.

На рис. 6.6 изображен вертикальный тумблер и кнопка управления для клапана, сконфигурированная по трем разным схемам описания. Кнопка управления содержит символ клапана. На рис. 6.6а метки элемента **Valve State** неоднозначны в терминах TRUE/FALSE. А на рис. 6.6б метки элемента **Close Valve** четко отражают его поведение в состояниях TRUE/FALSE. Кроме того, вертикальный тумблер содержит видимый логический текст со строками **ON**/**OFF**. На рис. 6.6в этот текст заменен на более наглядный **Closed/Open** (Открыто/Закрыто), и дополнительно значение по умолчанию для каждого элемента было указано в скобках. Это дополнительный текст отвлекает на лицевой панели, но очень полезен как описание в окне контекстной справки при создании ВПП. Согласно Правилу 6.10, кнопки предпочтительнее использовать для немедленных действий, таких как открытие и закрытие важного клапана. Однако правильное описание делает эти два типа элементов управления практически эквивалентными.

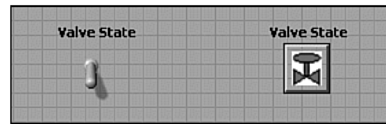


Рис. 6.6а. Два различных элемента управления клапаном: вертикальный тумблер и кнопка с символом клапана. Состояние клапана в этом примере не однозначно

*Рис. 6.6б. Собственна метка **Close Valve** четко определяет состояние TRUE/FALSE. Вертикальный тумблер также снабжен меткой состояний **OFF\ON**. С точки зрения графического интерфейса – кнопка предпочтительнее*

*Рис. 6.6в. Положения TRUE/FALSE вертикального тумблера названы соответственно: **Closed** (Закрыто) и **Open** (Открыто). В скобках указано значение по умолчанию для каждого элемента*

Правило 6.12 Избегайте использования кнопок и переключателей как индикаторов и светодиодов – как элементов управления

Объекты лицевой панели в LabVIEW обладают полезным свойством – любой элемент управления можно превратить в индикатор, и наоборот. Но в некоторых случаях это не имеет смысла. Например, кнопки и переключатели не стоит использовать как индикаторы, а светодиоды как элементы управления. Такие обратные присвоения непонятны и могут смутить людей, таким образом, нарушается Правило 6.1.

Логические элементы, которые являются частью кластера, поддерживают один и тот же внешний вид, когда кластер используется и как индикатор, и как элемент управления. В этом случае сложно не нарушить Правило 6.12, потому что согласно Правилу 6.3 требуются согласованные структуры данных. Существует два выхода. Первый – кнопки управления можно настроить так, чтобы нейтрализовать их внешний вид, как в случае с кластером ошибок, второй – необходимо создавать различные кластеры для элементов управления, содержащие кнопки и переключатели, и для индикаторов, содержащие светодиоды. В этом случае появится точка приведения типов, если кластеры сохранены как тайпдеф и соединены вместе. А потому как кластеры часто сохраняют как тайпдеф, этого подхода следует избегать, так как нарушается Правило 6.3.

6.2.2. Численные элементы

Существует две категории численных типов данных: целые числа и числа с плавающей точкой (необходимы для представления дробных чисел).

Правило 6.13 Используйте представление I32 для целых чисел и DBL для чисел с плавающей точкой

В разделе 6.1.1 «Выбор элементов управления и типов данных» обсуждается важность эффективности, простоты и согласованности типов данных. Согласно Правилу 6.3 важно поддерживать согласованность структур данных во всем приложении. В большинстве ситуаций используйте I32-представление целых чисел и DBL для чисел с плавающей точкой. Все физические единицы, описывающие напряжение, силу, частоту, длину, угол и т.д., должны быть DBL.

Численные функции LabVIEW могут принимать и оперировать с любым типом численных данных. Способность функции или терминала ВП воспринимать данные более чем одного типа называется полиморфизмом. Полиморфные функции и ВП адаптируются к входным данным вместо того, чтобы разрывать проводник данных или принудительно выполнять приведение типов на входном терминале. Более важно то, что функция завершает операцию корректно. Большинство встроенных функций LabVIEW, ВП и конструктов, которые не являются полиморфными, используют I32 для целых и DBL для чисел с плавающей точкой. Например, счетчики итераций в циклах и терминалы индексов в функциях по работе с массивами и строками используют I32. DAQmx и большинство драйверов инструментов возвращают DBL или волновые сигналы, состоящие из массивов DBL. Аналогично и ВП, используемые при анализе, оперируют DBL. Поэтому использование I32 для целых чисел и DBL для чисел с плавающей точкой помогает поддерживать согласованные типы данных во всем приложении и избегать ненужного приведения типов и конвертирования.

Для полноты картины рассмотрим два исключения из этого правила: цифровой сигнал I\O и традиционные ВП сбора данных. Чтение или запись цифрового сигнала с устройства сбора данных обычно требует 1 или 4 байта целых чисел без указания знака (U8, U32). Также аналоговые DAQ ВП используют 4 байта числа одинарной точности (SGL), если настроены на чтение или запись масштабированных данных вместо волнового сигнала. Однако DAQmx ВП и драйвер функционально и стилистически «старше», чем традиционные DAQ ВП и драйвер. И я крайне рекомендую обновить все приложения, которые используют традиционный DAQ.

Возможно, вы, как и я, задаетесь вопросом, почему LabVIEW использует I32 для величин вроде счетчика итераций цикла, вместо U32. Изобретатели LabVIEW настаивают на том, что бы I32 и DBL были основными форматами данных для целых чисел и чисел с плавающей точкой соответственно. Поэтому в LabVIEW используются I32 и DBL.

Правило 6.14 Используйте автоматическое форматирование, если не требуется специальный формат

По умолчанию LabVIEW форматирует числа с плавающей точкой внутри численного элемента управления или индикатора автоматически. А именно, LabVIEW отображает соответствующее число знаков после запятой и переключается между десятичным и экспоненциальным отображением, как на ручном калькуляторе. Иногда бывает необходимо отобразить определенное число знаков после запятой, чтобы показать точность измерений или какого-либо параметра. А без веской причины лучше использовать автоформатирование.

Правило 6.15 Показывайте основание системы исчисления для данных в восьмеричной, шестнадцатеричной или двоичной системах

Численные элементы управления в целом представлении по умолчанию форматируются как десятеричные. Но также можно выбрать восьмеричное, шестнадцатеричное или двоичное представление на вкладке **Format and Precision** меню **Numeric Properties**. Выберите **Advanced Editing Mode**, чтобы сделать видимыми коды численного форматирования (**Numeric Format Codes**). Когда вы отображаете данные в таких менее популярных форматах, всегда делайте видимым основание системы исчисления. Это делается с помощью команды контекстного меню

Visible items ⇒ Radix. Если основание системы не видно, обычно полагают, что данные представлены в десятичной системе.

На рис. 6.7 показаны 4 разных элемента управления, которые могут быть использованы для записи цифрового сигнала через 8-битный цифровой порт ввода/вывода. К ним относится массив логических переменных и три численных элемента управления, настроенных на представление в двоичной, восьмеричной и десятеричной системах. И хотя каждый элемент отражает одно и то же значение, различные форматы было бы сложно интерпретировать без видимого основания системы исчисления.

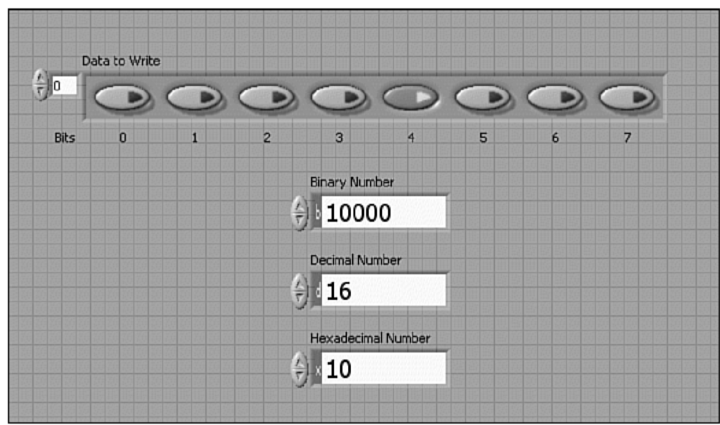


Рис. 6.7. Четыре различных элемента управления содержат цифровой сигнал для записи по 8-битному порту ввода/вывода. К ним относится массив логических переменных и три численных элемента управления, настроенных на представление в двоичной, восьмеричной, и десятеричной системах. Важно сделать видимым основание системы исчисления, если элемент отображает данные не в десятеричной системе

6.2.3. Специальные численные данные

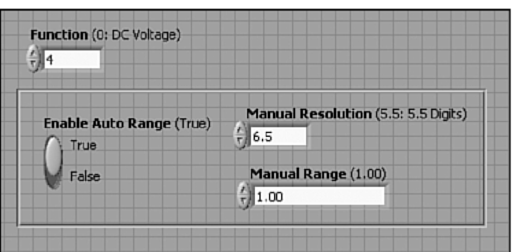
Четыре численных типа данных обладают специальными свойствами, к этим типам относятся: временная отметка, refnum (ссылка) и имя ввода/вывода, кольцевая структура, перечень. Тип данных «временная отметка» хранит абсолютное время и выделяет 8 байт на секунды и доли секунды. Поэтому временная отметка очень точна. **Refnum** и имя **I/O** (Ввод/Вывод) – это ссылки на специальный экземпляр открытого ресурса, как-то: файл, устройство, сетевое соединение, изображение, приложение LabVIEW или ВП. Они действуют как указатель на структуру данных, описывающих ресурс. Если назначение двух типов ссылок похоже, то элементы управления существенно отличаются. А именно, имя ввода/вывода отображается в интуитивно понятном формате, а refnum – нет. Имя VISA, IВП, DAQmx сессий, например, содержит информацию об оборудовании, с которым

они работают. Refnum activeX и .NET компонентов содержит только информацию о типе данных. Поскольку каждый тип ресурсов имеет свои требования, реально используемый тип данных, используемая память, структура данных различны для каждого. Расположение refnum и имени ввода/вывода в табл. 6.2 и 6.3, в которых типы данных выстроены по эффективности использования памяти, касается только части самой ссылки этой структуры данных. Важно отметить, что речи о самом ресурсе не идет, только о ссылке.

Кольцевая структура и перечень ставят в соответствие выбранным текстовым элементам числа. Они очень эффективны для представления дискретного набора элементов, которые наглядно описаны при помощи текстовых меток на лицевой панели, но числа гораздо более функциональны на блок-диаграмме. Перечень всегда представлен набором последовательных целых чисел, начиная с 0, который ставится в соответствие текстовым элементам. Кольцевая структура может иметь любое численное представление, и текстовым элементам можно поставить в соответствие любые числа в диапазоне, допускаемом в выбранном представлении, в том числе непоследовательные, отрицательные или дробные. Преимущество использования перечней и кольцевых структур состоит в том, что текстовое описание элементов может быть более полным и понятным, нежели числа и выбор элемента из списка. Эти элементы играют важную роль в стильном программировании, так как обеспечивают читаемость как лицевой панели, так и блок-диаграммы.

На блок-диаграмме константы типа перечень и список могут отображать текстовые метки вместо или вместе с численными значениями, что они представляют. Создайте константу из меню быстрого доступа терминала (**Create ⇒ Constant**). Константа содержит список элементов перечня, как и элемент управления на лицевой панели. Вы можете выбрать любой из текстовых элементов и показать/спрятать его. Дополнительно, когда перечень связан проводником данных с селектором структуры Base, выбранные элементы перечня появляются в области селектора. Поэтому перечень используется при создании конечного автомата, более подробно это обсуждается в главе 8 «Схемы расположения».

На рис. 6.8 приведен пример драйвера ВПП, который настраивает цифровой мультиметр. На рис. 6.8а цифровые элементы управления используются для того, чтобы сконфигурировать параметры **Function** и **Manual Resolution** на лицевой панели ВПП. Эти параметры передаются через терминалы на соединительной панели ВПП. Вызов ВПП показан на рисунке справа, используются численные константы, чтобы настроить эти параметры. Значение 4 для параметра **Function** – бессмысленно, а значение 6.5 для **Manual Resolution** рискованно. Диапазон возможных значений и их смысл не ясны. На рис. 6.8б используется кольцевая структура вместо численных элементов управления. При вызове ВПП используются те значения, которые передаются в соответствии с выбранными позициями в меню. Эти константы обладают дискретным набором текстовых меток, и каждая позиция описана и ясна для пользователя. Следовательно, кольцевая текстовая структура функциональнее, понятнее и надежнее, чем численные элементы управления.



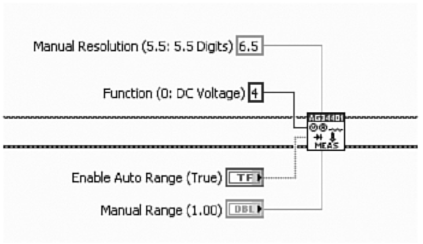
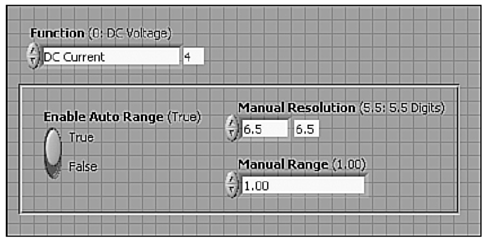


Рис. 6.8а. Лицевая панель драйвера ВПП с числовыми элементами управления функцией измерения и разрешением. При вызове ВПП используются численные константы для определения параметров



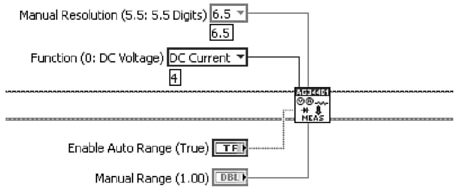


Рис. 6.8б. Лицевая панель драйвера ВПП, содержащая текстовую кольцевую структуру, вместо численных элементов управления

Правило 6.16 Используйте как можно больше перечни во всех приложениях

Текстовые элементы перечней и кольцевых структур формируют очень важный источник документации на лицевой панели и блок-диаграмме, который встроен в эти элементы управления. Я рекомендую использовать перечни из-за их максимальной читабельности. Однако существует несколько исключений, когда следует использовать кольцевую структуру. Используйте кольцевую структуру, когда текстовые метки соответствуют непоследовательным, отрицательным или дробным значениям или когда их локализация представляет проблему. Также следует использовать кольцевую структуру при создании драйвера или набора инструментов, которые планируете распространять на других языках. Текст в перечне единственным образом определяет данные на родном языке той версии LabVIEW, в которой был создан. Сами функции перечня портированы на локализованные версии, но текст перечня не переводится. Поэтому NI рекомендует в этом случае использовать кольцевую структуру и логические переменные, нежели перечни. Текст внутри кольцевой структуры изящно переносится на язык локализации, потому что текст символически соответствует числам. Наконец, свойства строк кольцевой структуры считываются и записываются программно,

но их невозможно записать для перечня. Суммарное сравнение кольцевой структуры и списка приведено в табл. 6.4.

Таблица 6.4. Сравнение кольцевой структуры и перечня

Поведение типа данных	Кольцевая структура (ring)	Перечень (enum)
Представление целых (без знака)	✓	✓
Представление целых (со знаком)	✓	
Представление чисел с плавающей точкой	✓	
Непоследовательные значения	✓	
Текстовые элементы управления, константы, индикаторы	✓	✓
Селектор текстового меню		✓
Возможность настройки свойств строк	✓	
Строки локализуются	✓	

Правило 6.17 Сохраняйте перечни как тайпдеф

Во время разработки приложения часто приходится удалять и добавлять элементы в перечень. Однако копии ваших элементов не будут поддерживать те же объекты, что и оригинальный элемент управления, если только он не был сохранен как тайпдеф. Этот момент легко проглядеть, и он является частой причиной неправильного поведения приложения. Поэтому следует всегда сохранять перечни как тайпдеф. Это можно сделать следующим образом. Выберите **Advanced** ⇒ **Bustomize** из меню быстрого доступа, затем выберите **Type Def.** или **Strict Type Def.** в поле **Type Def. Status** и сохраните элемент как вTL файл. Согласно Правилу 6.6 следует выбрать **Strict Type Def.**, если у элемента есть специальные свойства и дополнительно текстовые метки; во всех других случаях лучше использовать **Type Def.** Обратите внимание, что кольцевую структуру необходимо сохранять как строгий тайпдеф.

6.2.4. Строка, путь и изображение

Строка, путь и изображение – типы данных переменной длины, хранящиеся в последовательных областях памяти. Строки предоставляют огромные возможности передачи и ввода текстовых данных. Однако для них нет фильтров или встроенных методов ограничения формата, числа знаков, регистра или написания данных. Поэтому они предоставляют больше всего возможностей для ошибки.

Правило 6.18 Избегайте строковых элементов управления на лицевой панели

Избегайте использования строковых элементов управления на лицевой панели, кроме случаев, когда требуются данные в свободной форме. Как разработчики,

мы никогда не должны доверять пользователю ввод данных в особой манере, и чем больше ошибочных решений мы предоставляем пользователю, тем менее устойчиво наше программное обеспечение к ошибкам. А согласно Теореме 3.1, надежность – задача разработчика.

Существует два важных применения строковых элементов управления и индикаторов – это отображение данных с устройств и описаний. Если оператору требуется вводить комментарии в свободной форме, к описанию единичного теста или причины превышения уровня тревоги строковые элементы подходят. Но вместо того чтобы создавать строковый элемент на лицевой панели, создайте диалоговый ВП, который даст возможность ввести описание и закроется сразу же после этого.

Строковые индикаторы бесценны при устранении неполадок во взаимодействии с приборами. Управление приборами происходит на низком уровне с использованием функций VISA Write и VISA Read. Эти функции передают команды и отображают ответы приборов как строки, обычно загадочные. Однако во время разработки ПО или настройки приборов бывает необходимо увидеть эти низкоуровневые сообщения в строковых индикаторах. В самом деле, строковые индикаторы обладают рядом полезных режимов отображений данных, включая отображение непечатаемых символов и шестнадцатеричное представление. В режиме отображения непечатаемых символов (режим \ – **backslash**) мы можем видеть такие символы, как табулятор, перевод и заполнение строки в виде кодов: \t, \r, \n. В шестнадцатеричном режиме мы можем видеть бинарные данные в шестнадцатеричном формате.

Правило 6.19 Используйте перечень, кольцевые структуры и указатели пути вместо строковых элементов управления везде, где возможно

*Правило 6.20 Держите кнопку **Browse** (Обзор) видимой для указателей пути на лицевой панели*

Используйте табл. 6.1, чтобы найти простейшую замену строковым элементам. Например, перечни и кольцевые структуры обеспечивают дискретный набор текстовых элементов, как описано выше. Всегда используйте указатели пути для ввода или программирования текстовых данных, представляющих собой путь к файлу. LabVIEW форматирует путь, используя стандартный синтаксис операционной системы. Тем самым поддерживается кроссплатформенность. Кнопка **Browse** (Обзор) позволяет пользователю легко выбрать желаемый файл или директорию, поэтому она должна быть видимой на лицевой панели. Настройте **Browse Options** (Опции обзора), включая **Selection Mode**, чтобы ограничить выбор пользователя и проверить данные.

Тип данных изображения хранит последовательность операционных кодов и данных об изображении. Операционные коды – это низкоуровневые инструкции, которые говорят вPU, что рисовать. Существует около 50 операционных кодов, соответствующих множеству геометрических форм и настройкам поразрядной карты изображения. Для каждого операционного кода существует свой операнд, который включает координаты, цвет, и другие специфичные данные об операциях. Операционные коды и сами данные, сохраняемые типом данных изображения, доступны разработчику. Используйте соответствующие ВП с палитры функций **Picture** для создания изображения. Поскольку информация об изображении хранится в последовательных областях памяти, она относительно проста и похожа на строковые и текстовые массивы.

6.3. Конструкты данных

Конструкты данных – это собрание из одного или более фундаментального типа данных, используемого для создания нового типа данных. К ним относятся как определяемые разработчиком структуры (массивы, кластеры, переменные, очереди, условный тип данных), так и встроенные типы данных (матрицы, кластер ошибок, волновой сигнал, динамическая переменная). Массивы и кластеры предоставляют огромные возможности и являются, по сути, основным способом организации данных в приложении. В следующем разделе представлены правила оформления конструктов данных, особое внимание уделяется массивам и кластерам.

6.3.1. Простые массивы и кластеры

Массивы и кластеры состоят из контейнеров, внутри которых можно разместить любые данные или конструкты. Сложность массивов и кластеров целиком зависит от содержания. Они могут быть простыми, как скалярная логическая переменная, или сложными, как вложенные массивы и кластеры. В этом разделе обсуждаются простые массивы и кластеры.

Массивы хранят наборы данных, которые имеют множество значений, но все одного типа. В памяти массивы записаны как заголовок, содержащий длину каждого измерения массива в виде 4-байтового целого числа, затем записываются данные, содержащиеся в массиве. Массивы, содержащие один из простых типов данных, описаны в разделе 6.2 «Простые типы данных», хранят заголовок и сами данные в последовательных областях памяти.

Правило 6.21 Используйте массивы для данных с множеством значений; используйте кластеры для группировки различных типов данных

Большинство считает массивы собранием связанных данных, похожим на кластеры, с той разницей, что элементы – одного типа. И это верно. Однако существует очень важное различие между массивами и кластерами помимо разрешенных типов данных. Все элементы массива обладают одними свойствами, тогда как каждый элемент кластера может обладать своими уникальными свойствами.

У массива одна собственная метка, размер, цвет, шрифт, описание, единица измерения и другие свойства, одинаковые для всех элементов. Таким образом, все элементы массива абсолютно одинаковы по своим свойствам и различаются только индексом. В результате этого различия элементы массива связаны друг с другом сильнее, чем элементы кластера. Массивы используются для хранения векторных или многозначных величин, тогда как кластеры применяются для группирования множества связанных, но автономных элементов данных.

Примерами массивов могут служить точки отсчета волнового сигнала и коэффициенты полинома. Эти элементы можно рассматривать как одиночные объекты, содержащие множество элементов. Индекс элемента полностью характеризует элемент в массиве как точку волнового сигнала или коэффициент полинома.

К кластерам относятся входные (**error in**) и выходные (**error out**) кластеры ошибок. Эти кластеры содержат логическую переменную, определяющую статус, численную переменную, соответствующую коду, и строку, относящуюся к источнику. Поскольку элементы связаны, то они логически группируются в кластер. А так как типы данных элементов различны, то массив в данном случае не подходит. Но если мы на мгновение представим, что все элементы – строки, то использование массива стало бы возможным. Однако и тогда индекс элемента был бы недостаточен для идентификации элементов. Вместо этого различным элементам управления требуются независимые метки. Итак, нам нужен кластер.

Правило 6.22 Используйте массивы для хранения большого объема данных или наборов данных динамически изменяющейся длины

Другое различие между массивами и кластерами состоит в том, что число элементов массива может быть определено программно и изменено так часто, как необходимо, тогда как число элементов кластера фиксировано. Также может быть очень невыгодно с точки зрения времени и места создавать кластеры с очень большим числом элементов. Поэтому, когда число элементов велико или может меняться во время работы программы и все элементы одного типа, используются массивы, а не кластеры.

Давайте снова обратимся к ВП Torque Hysteresis, который мы рассматривали в разделе 6.1.3 «Создание конструктов данных». Кластеры применяются так, как показано на рис. 6.5. На первом шаге измерений в диалоговом окне пользователь вводит информацию, касающуюся тестируемого образца, включая имя заказчика, модель, год выпуска и др. Эта информация сохраняется в кластер **UUT Information**. Здесь все данные связаны и все одного типа – строки. Кластер выбран из-за того, что каждый элемент нуждается в независимой текстовой метке и в описании.

Затем приложение позволяет оператору ввести некоторые параметры движения: скорость, угол, максимальны момент вращения. Данные сохраняются в отдельный массив **Motion Parameters**. Обратите внимание, что кластер **Motion Parameters** слабо связан с кластером **UUT Information**. Точнее, параметры движения выбраны так, чтобы отражать физические характеристики UUT. Однако это не строгое требование. На рис. 6.5 **UUT Information** и **Motion Parameters** хранятся в разных кластерах. Если бы приложение было больше и более сложным, то мы могли бы объединить элементы этих двух кластеров в один, чтобы минимизировать количество терминалов ВПП и проводников данных. Большой кластер можно было бы переименовать, скажем, в **Config Data**.

После этого запускается тест, и данные об угле и моменте вращения возвращаются как кластер, содержащий два различных одномерных массива. Массивы используются потому, что данные представляют собой волновой сигнал и собираются с двух различных входных каналов. Обратите внимание, что вместо кластера массивов можно было выбрать двумерный массив. Если бы данные возвращались собирающими их ВП как двумерный массив, то для того, чтобы не нарушить Правило 6.3, мы бы продолжили пользоваться двумерным массивом. Однако в этом случае данные собираются двумя разными ВПП и внутри ВП Run Test и передаются как два различных одномерных массива. Соединение их в кластер помогает сохранить их различную структуру и делает данные напрямую совместимыми с графическим индикатором XY Graph. Таким образом, кластер одномерных массивов поддерживает согласованность типов данных во всем приложении, тем самым удовлетворяется Правило 6.3.

Наконец, данные обрабатываются, вычисляется статистика, и данные сохраняются в кластер **Statistics**. Все элементы этого кластера имеют одинаковый формат, DBL, но кластер используется для того, чтобы сохранить их уникальные метки.

Правило 6.23 Вводите описания для ячеек массивов и кластеров и элементов управления

Правило 6.24 Используйте инструменты выравнивания, чтобы кластеры выглядели компактно и аккуратно

Массивы и кластеры зачастую формируют инфраструктуру данных приложения. Когда они определены, то могут распространиться на все приложение. Поэтому важно, чтобы они были согласованы и хорошо документированы. Как уже обсуждалось в главе 3, правильная документация должна быть лаконичной, понятной (единицы для физических величин указаны в скобках). Добавьте также документацию для каждого элемента управления, ячейки массива или кластера. Используйте инструменты автоформатирования, чтобы компактно расположить элементы в кластере. Я выяснил, что некоторые кластеры могут заметно увеличиться в числе элементов по мере развития приложения, так что я предпочитаю сделать их компактными, чтобы они занимали как можно меньше места на лицевой панели. Сюда относится и использование настроек по умолчанию (размер, шрифт, расположение, сжатие) элементов кластера. Расположите элементы в компактной вертикальной форме, сверху вниз, согласно порядковому номеру в кластере, для этого выберите **Autosizing** ⇒ **Arrange Vertically** из контекстного

меню кластера. Или же выберите набор элементов кластера и расположите их в сжатой вертикальной или горизонтальной форме, выбрав **Vertical Compress** или **Horizontal Compress** из меню инструментов **Distribute Objects**. Это поможет избежать заполнения лицевой панели большими кластерами.

На рис. 6.9 показан кластер **Motion Parameters**, который используется в ВП Torque Hysteresis. Элементы управления и описания выровнены независимо по правому (**Right Edge**) и левому (**Left Edge**) краям соответствующими инструментами. Элементы управления также разделены в пространстве и сжаты по вертикали. Каждый элемент управления обладает описанием в окне контекстной справки, как например элемент **Velocity Amplitude**.

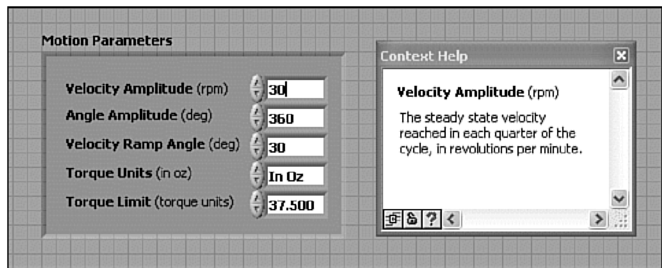


Рис. 6.9. Кластер ***Motion Parameters*** *из ВП Torque Hysteresis, в окне контекстной справки отображается описание элемента* ***Velocity Amplitude****. Кластер выглядит компактно, аккуратно и хорошо документирован*

Правило 6.25 Сохраняйте все кластеры как тайпдеф

Это правило сложно переоценить. Кластеры часто формируют структуру всего приложения и являются предметом частых изменений. Сохраняйте все кластеры как строгий тайпдеф и применяйте тайпдеф для всех копий кластера. Это гарантирует то, что все изменения, которые произойдут с тайпдефом, будут применены ко всем копиям кластера в каждом ВП. Это в свою очередь гарантирует согласованность конфигураций кластеров в приложении, существенно упрощая задачу технической поддержки. Согласно Правилу 6.6 используйте **Strict Type Def.**, если у элемента есть дополнительные свойства вдобавок к стандартным, иначе используйте **Type Def**. Лично я предпочитаю **Strict Type Def.** потому, что мне может потребоваться изменить тайпдеф после того, как я создал множество его копий. Так я смогу изменить внешний вид кластера или элемента, и все изменения будут применены ко всем его копиям. Клонируйте копию тайпдефа, выбрав **Select Control** из палитры **Controls**, или перетащите его из окна проводника проекта. Если вам нужна константа, совместимая с кластером, выберите **Create ⇒ Constant** из меню быстрого доступа терминала тайпдефа. Созданные таким образом константы совместимы с тайпдефами.

Правило 6.26 Всегда используйте Bundle By Name и Unbundle By Name

На блок-диаграмме используйте функции Bundle By Name (Соединить по имени) и Unbundle By Name (Разъединить по имени), чтобы получить доступ к элементам кластера. Эти функции предоставляют больше возможностей по сравнению с функциями Bundle и ***Unbundle***. Вы можете получить доступ к любому элементу кластера в любом порядке, используя функции Bundle By Name и Unbundle By Name; кроме того, эти функции продолжат работать, если ваш кластер изменится, если только не изменятся сами элементы, с которыми работают функции. Это правило хорошо работает с Правилом 6.25. Самое важное – метки единственным образом идентифицируют элементы кластера и предоставляют отличную документацию на диаграмме. Поскольку функции Bundle By Name и Unbundle By Name имеют размер самой длинной метки, сохраняйте метки элементов лаконичными и понятными.

Правило 6.27 Избегайте использования кластеров для интерактивных элементов управления с диалоговыми ВП

Обратите внимание, что кластеры ВП Torque Hysteresis, показанные на рис. 6.5а, не рассчитаны на интерактивное поведение. Сжатие, шрифты по умолчанию и прочие свойства делают их менее подходящими для графического интерфейса вроде диалога. Вдобавок, есть некоторые аспекты, связанные с клавишей табуляции. Индивидуальные элементы управления, которые не являются частью кластера, по умолчанию настроены на простую навигацию клавишей табуляции. Порядок табуляции для элементов на лицевой панели вы можете выбрать указать, выбрав **Edit ⇒ Set Tabbing Order**. Между элементами кластерам также можно переключаться клавишей табуляции, после того как один из элементов кластера выбран активным. И пока не будет выбран элемент вне кластера, табуляция будет происходить только между элементами кластера. Таким образом, осуществление навигации клавишей табуляции требует дополнительного программирования, что не нужно для элементов управления на лицевой панели.

Рисунок 6.10 иллюстрирует: кластер можно использовать вместе с диалоговым окном, которое получает информацию о тестируемом образце от пользователя. Видимая часть панели содержит строковые элементы управления, которые были настроены для интерактивного взаимодействия, как показано на рис. 6.10а. На самом деле кластер **UUT Information** сдвинут в невидимую боковую часть панели, как показано на рис. 6.10б, вместе с индикатором кластера ошибок **Cancelled?**. Табуляция для не интерактивных элементов запрещена. Это сделано следующим образом: **Advanced ⇒ Key Navigation ⇒ Tab Behavior ⇒ Skip this control when tabbing**. Обратите внимание, что кластер куда более компактен, чем соответствующие элементы управления. В этом примере элементы на лицевой панели настроены для работы в качестве графического интерфейса пользователя, а кластер оптимизирован для ВПП.

На блок-диаграмме диалоговое окно **UUT Information** соединяет данные от интерактивных элементов управления в кластер по нажатию кнопки **OK**. На рис. 6.10в мы видим, что терминалы функции Bundle непонятны. Невозможно понять, какой терминал какому элементу кластера соответствует, не зная порядка элементов в кластере. Более того, любые изменения в кластере добавят проблем разработчику при таком подходе. Например, порядок элементов в кластере может измениться после добавления или удаления, или переименования строковых элементов управления, при этом число элементов и тип данных могут остаться неизменными. В результате будет такая же конфигурация, как на рис. 6.10в, только с неправильными выходными данными. Рис. 6.10г иллюстрирует правильное использование функции Bundle by Name для соединения данных в тайпдеф. Как видно, терминалы функции Bundle by Name четко помечены. Также эта функция не чувствительна к порядку элементов в кластере. Если добавить/удалить или переименовать элемент кластера, то функции Bundle By Name и Unbundle By Name перестанут работать, если изменения повлияют на элементы, с которыми они работают.

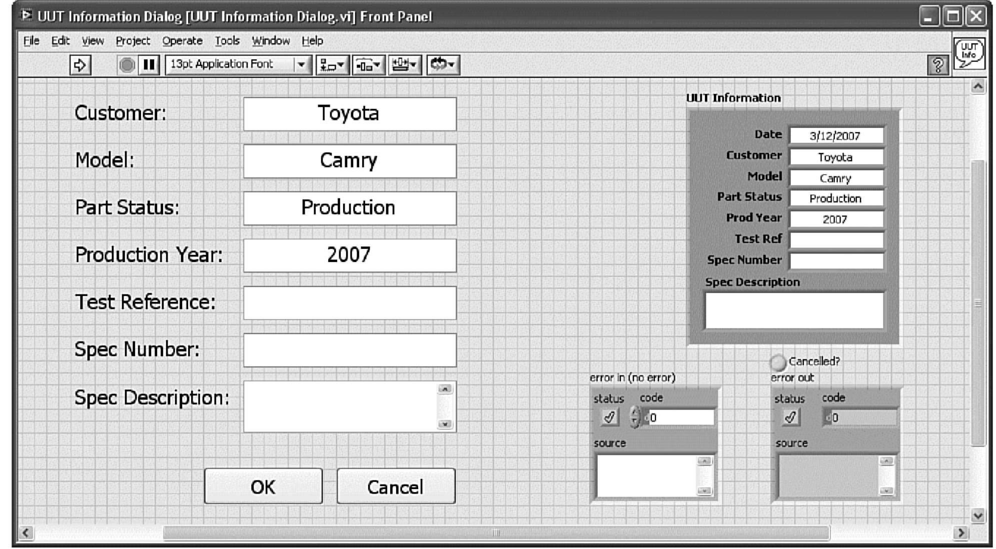


*Рис. 6.10а. Лицевая панель **UUT Information Dialog** содержит строковые элементы управления, настроенные для интерактивного взаимодействия*

*Рис. 6.10б. Кластер **UUT Information** и другие элементы управления сдвинуты из видимой части экрана*

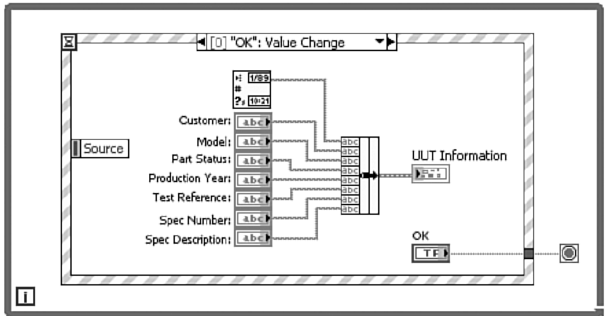


Рис. 6.10в. Строковые элементы соединены в кластер. Терминалы функции Bundle неразличимы

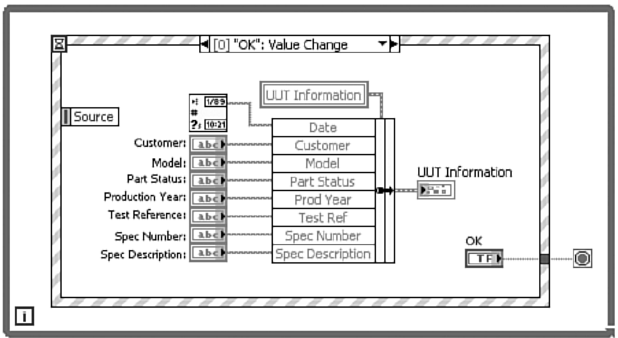


Рис. 6.10г. Функция Bundle By Name дает метки терминалам и увеличивает надежность приложения

6.3.2. Специальные конструкты данных

Несколько специальных конструктов данных включают в себя матрицы, кластер ошибок, волновой сигнал, динамический тип данных и универсальный тип данных. Первые четыре – это варианты массивов и кластеров, которые предопределены в LabVIEW и служат специальным целям. *Действительные* и *мнимые* матрицы – это просто двумерные массивы, содержащие числа в DBL-формате, и комплексные числа соответственно. Эти массивы были настроены и сохранены как тайпдеф. Они широко используются в ВП, принадлежащих математической библиотеке, особенно в линейной алгебре. Так же как кластеры ошибок, которые широко используются для устранения ошибок, являются просто кластерами.

Осциллограмма (waveform data type – WDT) как тип данных можно рассматривать в качестве специального типа кластера, состоящего из трех элементов плюс атрибуты. К этим элементам относятся: отсчет времени, $\mathbf{t_0}$; интервал времени между точками, **dt**; и одномерный массив чисел, соответствующих аналоговому или цифровому сигналу **Y**. Атрибуты могут содержать любое количество определенных разработчиком пар «имя–значение», например, имя и или описание волнового сигнала. Кроме существования атрибутов, WDT имеет принципиально отличную семантику в том, что касается математических операций, то есть имеет свою палитру функций; стандартные функции по работе с кластерами не поддерживают WDT.

Динамический тип данных (dinamic data type) – это универсальный тип данных, используемый в экспресс ВП. Он был разработан для неопытных разработчиков как очень гибкий тип данных, способный хранить и обрабатывать множество типов данных, ничего о них не зная. Вы можете подсоединить динамический тип данных к любому входному терминалу или индикатору, который принимает логические, численные данные или данные волнового сигнала. LabVIEW автоматически вставит необходимую функцию конвертирования. В памяти динамический тип данных представлен как массив аналоговых DBL волновых сигналов.

Универсальный тип данных (variant) кодирует имя данных, тип данных, сами данные и атрибуты или информацию о данных в обобщенный формат. Можно рассматривать универсальный тип данных как упаковщик, конвертирующий любые данные в новый формат, который описывается в универсальной манере. В LabVIEW универсальный тип данных совместим со стандартным, используемым в Microsoft COM и .NET-технологиях. Универсальный тип данных используется в протоколе DataSocket и ActiveX. Кроме того, универсальный тип данных обычно используется как обобщенный тип данных для передачи динамических, определенных во время работы приложения, данных. Это позволяет источнику (серверу) данных передавать клиенту любой тип данных. Интерфейс функций, таких как терминалы соединительной панели ВП, поддерживает универсальный тип данных вне зависимости от его содержания. Это позволяет проверять клиент и сервер, не влияя на данные.

Недостатками условного типа данных являются его малая эффективность, в смысле использования ресурсов памяти и процессора по сравнению с традиционными фиксированными типами данных, и необходимость программного кодирования/декодирования данных в приложении. В LabVIEW универсальный тип данных вводится с помощью функций To Variant и Set Variant attribute.

6.3.3. Вложенные структуры данных

Как уже отмечалось ранее, массивы и кластеры – это невероятно гибкие структуры данных. Например, нет практических ограничений на размерность массива, хотя обычно встречаются трех- и менее мерные массивы. Также вы можете создавать массивы, содержащие кластеры в качестве элементов, кластеры же сами могут содержать массивы, и так до бесконечности. Аналогично вы можете создать кластеры, содержащие комбинации простых типов данных, массивов и кластеров, которые в свою очередь могут содержать массивы и кластеры. Единственное ограничение состоит в том, что массивы не могут напрямую содержать другие массивы, если последние не соединены в кластер. В любом случае, не существует принципиальных ограничений на число слоев массивов и кластеров внутри массивов и кластеров.

Конструкты данных, содержащие множество слоев массивов и кластеров, называются вложенными структурами и считаются сложными по двум причинам. Во-первых, они затрудняют разработчику работу с блок-диаграммой. Например, изменение одного элемента может потребовать множества вызовов функций Index array и Unbundle By Name, за которыми последует множество вызовов функций Bundle By Name и Replace array, чтобы переделать структуру данных. Эти функции работы с кластерами и массивами часто перемешиваются с вложенными циклами, используемыми для индексации элементов каждого слоя массива. Все это становится сложным и запутанным уже после первых двух слоев.

Вторая причина, по которой вложенные структуры относятся к сложным структурам данных, касается того, как LabVIEW хранит их в памяти. Кластеры записываются как заголовок, в котором описан порядок и тип данных элементов,

вместе со значениями простых типов данных в последовательных областях памяти. Данные из массивов, строк, указателей пути, вложенных кластеров не хранятся внутри кластера. В кластере хранится указатель, в котором содержится информация о том, в каких областях памяти хранятся данные из массивов, и т.д. Следовательно, вложенные структуры данных хранятся как сеть указателей на данные в памяти конкретного компьютера. Чем больше слоев, тем сложнее эта сеть.

Поскольку LabVIEW управляет памятью автоматически, описанная сложная сеть для вложенных структур ясна разработчику. Однако разработчик должен позаботиться об эффективности использования памяти. Каждая ветвь вложенной структуры данных требует дополнительных операций по выделению и манипулированию памятью. Во-первых, большинство вызываемых функций Bundle\ Unbundle By Name, Index array и т.д. создают копию данных в памяти на выходе. По мере роста размера данных на каком-либо слое массива растет и буфер, и может потребоваться его перемещение или выделение дополнительной памяти. Предыдущие буферы памяти фрагментируют свои блоки памяти, когда выделяются новые области памяти. Таким образом, вложенные структуры используют память гораздо менее эффективно, чем простые структуры. Согласно теореме 6.1 время обращения к памяти является основным ограничением в современных вычислительных устройствах. Часто использование вложенных структур данных может привести к снижению производительности приложения.

Правило 6.28 Упорядочивайте сложные данные, используя вложенные структуры данных

Правило 6.29 Избегайте манипулирования вложенными структурами во время критических операций

Несмотря на страшно звучащие побочные эффекты, существует несколько практических применений вложенным структурам с ограничениями на число слоев и размер данных. Вложенные данные помогают ввести иерархию данных. Если вы работаете с современным ПК, в вашем распоряжении, вероятно, большие ресурсы памяти, и при правильной организации вложенных структур вы может минимизировать их влияние на производительность. Попробуйте использовать вложенные структуры данных в своем приложении там, где производительность не так критична.

Например, вложенные структуры данных могут быть использованы для организации мешанины данных конфигурации, исходных полученных и обработанных данных. Данные конфигурации могут быть получены из разных мест и соединены/разъединены в соответствии с производимыми операциями. Следует избегать обращения к вложенным структурам данных во время выполнения важных задач, таких как высокоскоростной сбор данных, детерминированный контроль или оперативная обработка информации. Никогда не соединяйте/разъединяйте последовательно собранные данные во вложенные структуры «на лету», во время сбора данных. По мере увеличения размера данных LabVIEW может выделить новую область памяти, что приведет к задержке. Задержки влияют на надежность задач, в которых важно время выполнения, они могут привести к потере или необратимому искажению данных.

Правило 6.30 Ограничивайте размер массивов, указывая максимальную длину

Другой способ избежать задержек, связанных с выделением памяти, состоит в том, чтобы заранее указать максимальный размер всех массивов, включая массивы на слоях вложенных структур. Все операции, следующие за инициализацией массива, можно выполнить с помощью функций Index array и Replace array Subset. В таком случае структуры данных обладают постоянным размером, что делает управление памятью более предсказуемым.

Используйте необычное начальное значение, чтобы различать правильные данные и данные инициализации. Константа NaN (Not a Number) особенно полезна для инициализации массивов чисел с плавающей точкой. Константу NaN легко определить и найти при использовании функций работы с массивами и ее нельзя спутать с измеренными данными. Кроме того, все графические индикаторы в LabVIEW не будут отображать точки, содержащие NaN.

На рис. 6.11 представлена альтернативная реализация ВП Torque Hysteresis, в которой использованы вложенные структуры. Элементы управления, входящие во вложенную структуру, представлены на рис. 6.11а. Кластеры **UUT Information** и **Motion Parameters** с рис. 6.5а скомбинированы в новый кластер вместе с массивом кластеров, который объединяет кластеры **Torque vs angle** и **Statistics**. Следующий массив сохраняет данные множества прогонов теста, повторно используя те же кластеры **UUT Information** и **Motion Parameters** для каждого прогона. Кроме того, больший кластер является частью массива, хранящего данные от многих UUT. Эта структура данных содержит пять вложений в следующем порядке:

1. Кластеры **Torque vs angle** и **Statistics**.
2. Вложенный кластер, содержащий эти кластеры с данными одного прогона.
3. Массив **Multiple Run Data**, состоящий из массива кластеров одного прогона.
4. Кластер, объединяющий **Multiple Run Data** с **UUT Information** и **Motion Parameters**.
5. Массив **Multiple UUT Data**, состоящий из массива данных о многих **UUT**.

На рис. 6.11б представлена улучшенная диаграмма, использующая преимущества вложенной структуры данных. Теперь в ней появилось два дополнительных цикла. Внутренний цикл For позволяет делать много прогонов теста одного UUT, соответствующая статистика вычисляется после каждого прогона. Средний цикл While позволяет тестировать множество UUT последовательно. Сдвиговые регистры и проводники данных используются для того, чтобы распространять вложенную структуру данных на каждый цикл. Внутри ВПП данные о каждом UUT и каждом прогоне теста добавляются в структуру данных. Приложение записыва-

ет данные теста для каждого UUT, но сохраняет данные в структуре до конца измерений, затем создает отчет, в котором суммирует данные для многих UUT. В этом примере вложенная структура данных снижает количество проводников данных на блок-диаграмме, объединяя множество структур данных в одну, и добавляет возможность проводить множество испытаний одного UUT, сохраняя данные от каждого теста для многих UUT.

На рис. 6.11в проиллюстрирован один из возможных подходов реализации ВП Run Test, в котором полученные данные объединены во вложенную структуру. Однако этот подход нельзя рекомендовать потому, что размер массивов постоянно увеличивается (функция Build Array) и манипуляции с кластером **Torque vs Angle** проходят в цикле, чувствительном к времени исполнения. На выходе каждой функции Build Array массив имеет размер, отличный от того, что на входе, это заставляет LabVIEW периодически выделять новые области памяти. Задержки, вызванные этими операциями с памятью, могут вызвать недопустимые флуктуации во время сбора данных. Функции Unbundle и Bundle By Name используются в каждой итерации цикла While и приводят к дополнительным, не нужным операциям внутри цикла. Поскольку в этом цикле собираются данные, это критическая точка приложения, в ней манипуляции данных должны быть сведены к минимуму.

На рис. 6.11г изображена диаграмма ВП Populate Data Structure – процедура, инициализирующая часть **Motion Run Data** всей структуры данных. Она создает вложенный кластер **Torque vs angle**, заполненный константами NaN. Максимальное число точек, описывающих момент вращения и угол, которое можно измерить в ходе теста, вычисляется по параметрам **Angle Amplitude (deg)** и **Max Rate (Sa/deg).** Создается массив NaN указанной длины и присваивается элементам **Torque (In Oz)** и **Angle(deg)** кластера **Torque vs Angel**. Затем этот кластер становится соответствующим элементом кластера, описывающего единичный прогон теста. Массив элементов, описывающих один прогон теста, инициализируется, основываясь на значении **Number of Runs**, и формирует массив **Multiple Run Data**. Массив **Multiple Run Data** становится соответствующим элементом одного кластера UUT. Один кластер UUT заменяет элемент массива **Multiple UUT Data** согласно номеру **UUT Number**. Инициализация вложенной структуры окончена. Заполненный ВП Data Structure вставляется в блок-диаграмму сразу перед циклом For, как показано на рис. 6.11д.

На рис. 6.11д показан надежный и эффективный метод реализации ВП Run Test с использованием инициализированной вложенной структуры. Операции с данными в чувствительном ко времени цикла сбора данных минимизированы до операций замены значений массивов, описывающих момент вращения и угол, на измеренные значения функцией **Replace Array Subset**. Поскольку эти массивы содержат данные простого типа и постоянного размера, то не создается дополнительных буферов памяти и не возникают непредсказуемые задержки во время сбора данных. Аналогично, функция Replace Array Subset используется для того, чтобы добавить данные об одном испытании в массив **Multiple Run Data** и для того, чтобы добавить кластер UUT в массив **Multiple UUT Data**. Операция замены всегда происходит вовремя и без перемещения или создания копий входных

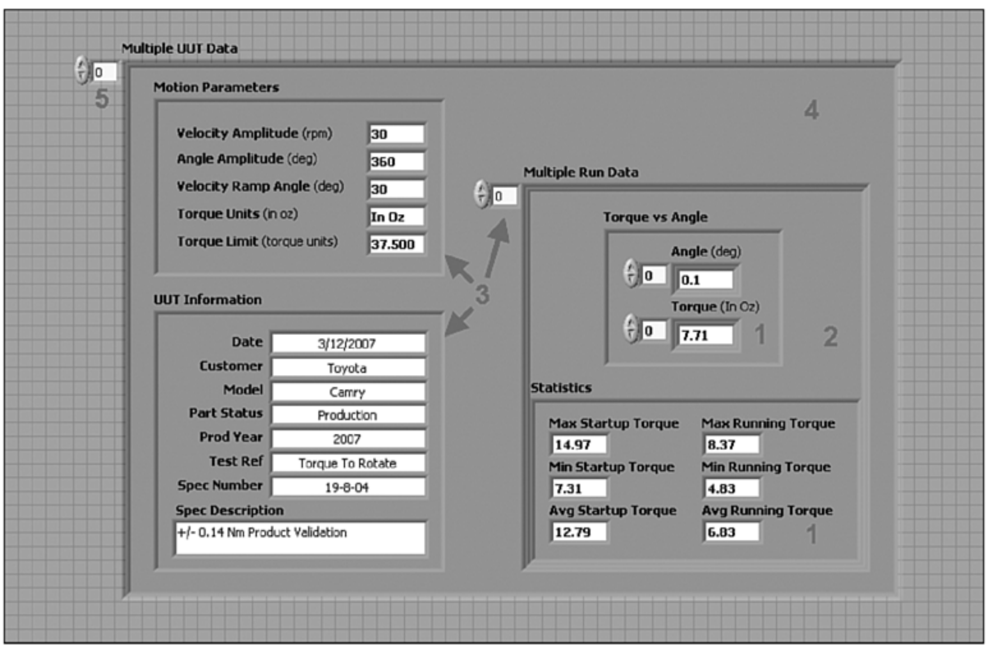


Рис. 6.11а. Альтернативная структура данных для ВП Torque Hysteresis имеет пять вложенных слоев

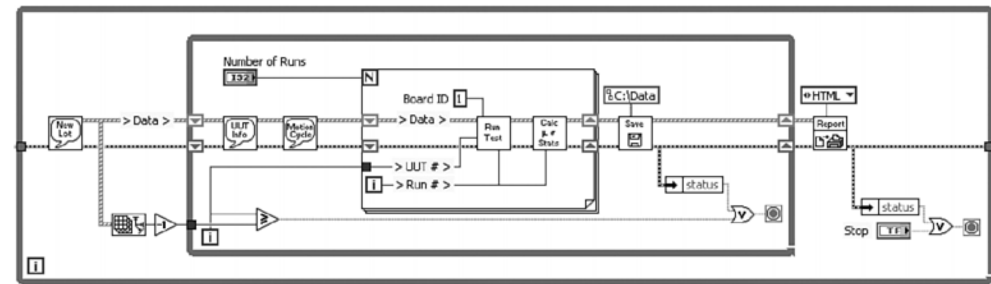


Рис. 6.11б. Вложенная структура данных используется во всех ВПП на блок-диаграмме верхнего уровня. Использование вложенной структуры данных позволяет сократить число проводников данных, тем самым упростив блок-диаграмму

данных. Это возможно потому, что функция не изменяет размер массива. После завершения испытаний процедура обработки просто ищет значения NaN и изменяет длину массива на правильную. Процедура обработки должна осуществлять уже после того, как завершены все чувствительные ко времени реализации процедуры, и производительность не играет решающей роли.

Внимательные читатели могли заменить точки приведения типов на входных терминалах ВП, контролирующих движение на рис. 6.11в и 6.11е. Эти точки при-

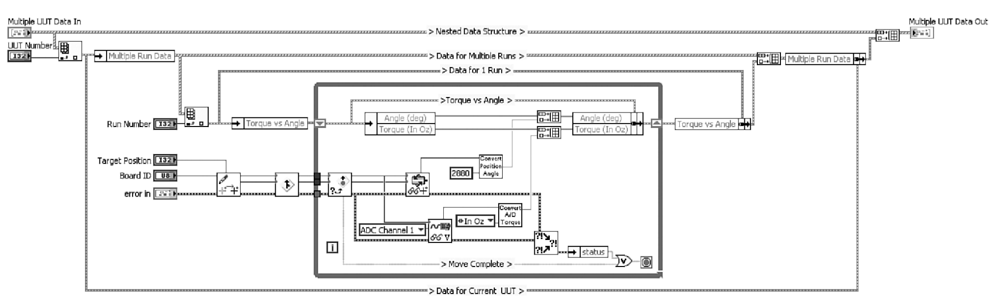


Рис. 6.11в. Один из вариантов ВП Run Test использует функции Unbundle by Name, Bundle by Name и Build Array для того, чтобы соединить полученные данные и кластер **Torque vs angle**. Такая реализация не эффективна, так как функция Build Array постоянно изменяет размер массива, что приводит к задержке, связанной с выделением памяти во время сбора данных

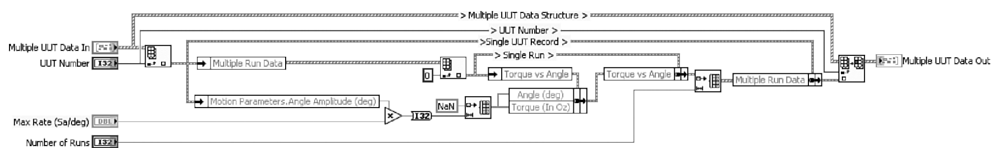


Рис. 6.11г. ВП Populate Data Structure – это ВПП, который инициализирует массив **Multiple Run Data** вложенной структуры данными **Torque vs Angle**, содержащими константу **NaN**

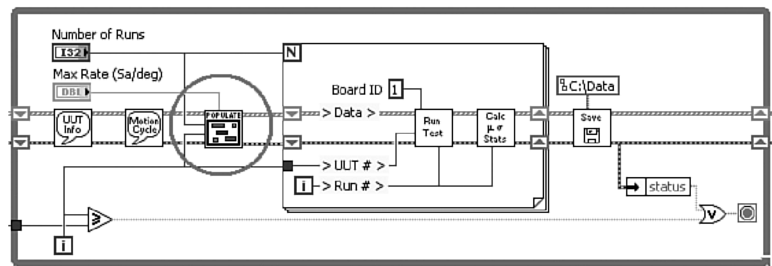


Рис. 6.11д. ВП Populate Data Structure вставляется в ВП верхнего уровня непосредственно перед циклом For, который осуществляет множественные испытания

ведения типов объясняются различными типами данных внутри ВП-драйверов. Входные и выходные терминалы, которые предполагается соединять между ВП-драйверов, состоят из нескольких различных тайпдефов для элементов управления и индикаторов. Соединение разных тайпдефов приводит к появлению точек приведения типов, то есть ВП-драйверы нарушают Правило 6.3. Однако приведение типов осуществляется над скалярными данными и представляет собой тривиальные операции, что не приводит к задержкам.

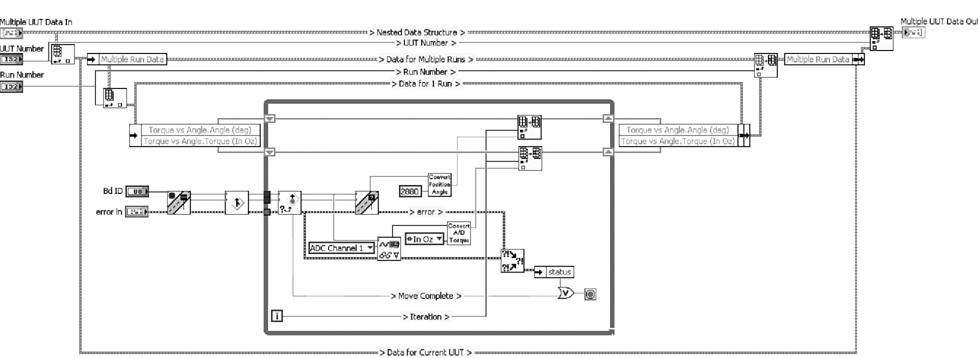


Рис. 6.11е. Улучшена надежность и производительность внутри ВП Run Test путем замены элементов данных на каждом уровне ранее инициализированной вложенной структуры функцией Replace array Subset. Структура данных постоянного размера, что позволяет избежать дополнительного выделения памяти

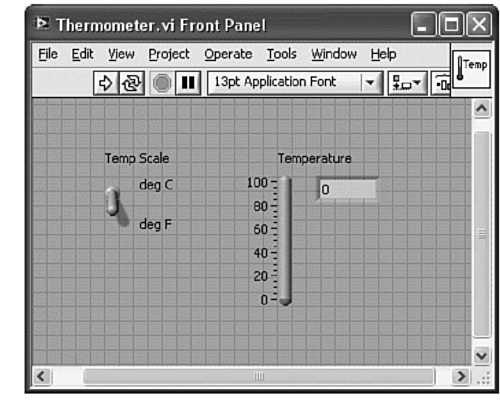
Я бы хотел добавить одно последнее замечание, касающееся вложенных структур данных. Некоторые источники рекомендуют вообще избегать вложенных структур данных. Я не согласен с этой рекомендацией. Как видно из предыдущего примера, вложенные структуры данных могут быть эффективно использованы для организации данных, улучшения функциональности, уменьшения числа проводников данных и упрощения технической поддержки. Если использовать вложенные структуры правильно, то плюсы в организации и функциональности могут превзойти минусы в производительности. В разделе 6.4 «Примеры» приведено еще больше примеров.

6.4. Примеры

В этом разделе представлено еще несколько примеров правильного использования структур данных.

6.4.1. ВП Thermometer

ВП Thermometer, показанный на рис. 6.12а, снимает данные о напряжении с термистора и переводит их в инженерные единицы. Вы могли узнать в ней классическое упражнение LabVIEW по созданию ВПП. В нем используется вертикальный тумблер **Temp Scale** для выбора единиц измерения (градусы по Фаренгейту и градусы по Цельсию). И хотя состояний всего два, использование логического элемента управления не оправдано, так как эти состояния не являются логически противоположными (Правило 6.8). Кроме того, на блок-диаграмме вычисляются обе единицы измерения, вне зависимости от выбора пользователя. На рис. 6.12б логическая переменная заменена перечнем. Использование перечня также дает возможность в будущем добавить и другие элементы. На блок-диаграмме теперь находится структура варианта, теперь вычисляются только те единицы, которые



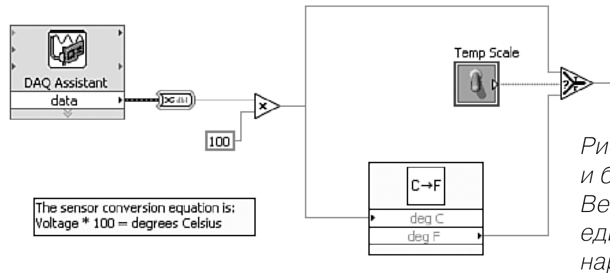


Рис. 6.12а. Лицевая панель и блок-диаграмма ВП Thermometer. Вертикальный тумблер выбора единиц измерения температуры нарушает Правило 6.8, так как два его состояния не являются логически противоположными

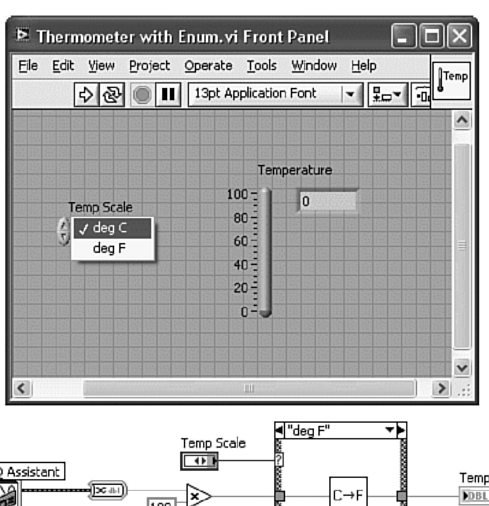


Рис. 6.12б. ВП Thermometer with Enum использует перечень для выбора единиц измерения температуры. Блок-диаграмма более эффективна, так как конвертирование в градусы по Фаренгейту не происходит, если не выбрано соответствующее положение

указаны переключателем. Помимо этого, элементы управления и индикаторы выглядят теперь как терминалы, а экспресс-ВП – как иконки. ВПП обеих версий представлены на рис. 6.12в. Мы можем видеть, что логический вход ВП Thermometer воспринимается неоднозначно на блок-диаграмме, вызывающей ВП. Очевидно, что перечень в данном случае гораздо лучше.

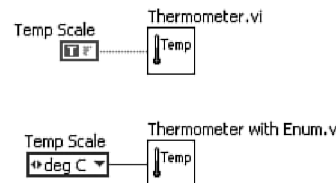


Рис. 6.12в. ВПП для ВП Thermometer и ВП Thermometer with Enum демонстрируют, что логическая переменная воспринимается неоднозначно, а перечень – нет

6.4.2. Вариант OpenG

Инструменты OpenG (3) включают расширенный набор ВП, которые используют универсальный тип данных. Многие из этих ВП рассчитаны на работу со многими типами данных LabVIEW как полиморфные ВП. Тип данных LabVIEW конвертируется в универсальный тип данных, который содержит информацию о типе данных, имени элемента управления и сами данные. Эта информация используется для повышения гибкости данных, которые можно передать ВПП через соединительную панель, и операций, которые можно осуществлять с данными внутри ВПП. Инструменты OpenG используют эту идею для расширения возможностей библиотек LabVIEW.

Когда тип данных LabVIEW подключается по проводнику данных непосредственно к входному терминалу универсального типа данных, то на входном терминале появляется точка приведения типов, и превращение в универсальный тип данных происходит автоматически. Однако непохожие типы данных по Правилу 6.3 соединять нежелательно. В идеале для лучшего стиля стоит избегать приведения типов. Но порой конвертирование типа данных LabVIEW в универсальный тип данных неизбежно. Как вариант, данные можно перевести в универсальный тип данных, используя функцию To Variant, перед тем как соединять с входным терминалом ВП OpenG. Этот метод позволяет заменить точку приведения типов явной функцией. И хотя два метода функционально эквивалентны, явное конвертирование показывает, что оно неслучайно.

Например, на рис. 6.13 ВП OpenG Empty Array используется для того, чтобы определить, является ли численный массив на ее входном терминале пустым. В окне контекстной справки виден фиолетовый проводник данных, подходящий к входному терминалу, так отображается универсальный тип данных. На иллюстрации расположены три функционально эквивалентные схемы. На верхней схеме

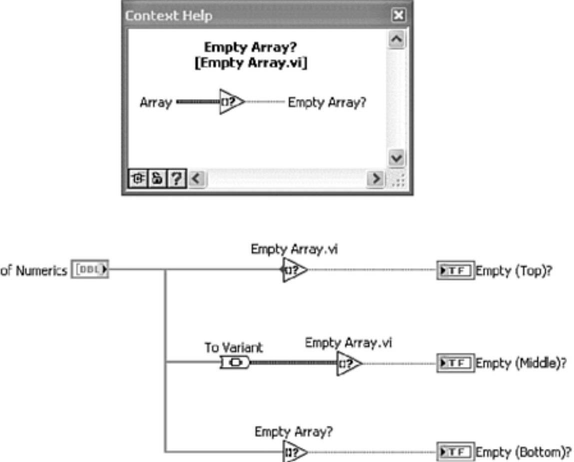


Рис. 6.13. На диаграмме расположены три способа определить, пустой ли массив на входе. На верхней схеме есть точка приведения типов на входном терминале ВП Empty Array. На средней схеме точка приведения типов устранена путем явного конвертирования типа данных. На нижней схеме используется полиморфная функция LabVIEW для согласования типов данных

массив подается напрямую в ВП Empty Array, и на входном терминале появляется точка приведения типов данных. На средней схеме добавлена функция To Variant, которая устраняет точку приведения типов данных. На третьей схеме используется функция LabVIEW Empty Array? для той же цели. Функция LabVIEW полиморфна и не требует приведения типов данных. С точки зрения производительности верхние две схемы идентичны. Средняя схема несколько предпочтительнее верхней, так как показывает, что приведение типов производится осознанно, и устраняет точку приведения типов. Нижний способ самый эффективный, так как полиморфная функция использует родные типы данных LabVIEW и приведение типов не нужно.

ВП OpenG Variant Configuration File в полной мере показывает мощь и гибкость универсального типа данных. Этот ВП считывает и пишет данные в текстовые файлы в стандартном для INI-файлов формате. Для чтения или записи множества значений при одном вызове ВПП используются кластеры, конвертированные в универсальный тип данных. Рис. 6.14 иллюстрирует использование ВП Write Section Cluster. Кластер **Motion Parameters** конвертируется в универсальный тип данных и соединяется с входным терминалом **Section Formatted Cluster**. ВП Write Section Cluster использует имя данных, тип и сами данные, которые получает из универсального типа данных для создания секции INI-файла. Элементы кластера соответствуют значениям секции, а имя кластера – имени секции, когда терминал секции не соединен. На рис. 6.14а показан исходный код, в котором пять значений записываются за один вызов ВП Write Section Cluster, и на рис. 6.14б показана результирующая секция INI-файла. Как показано на рис. 6.14в, вложенный клас-

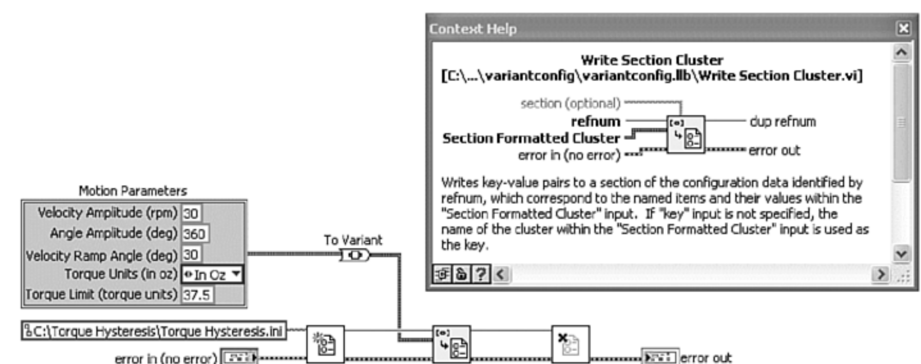
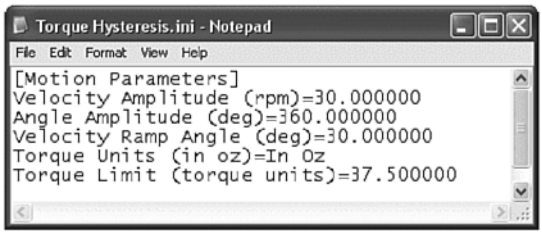


*Рис. 6.14а. Кластер **Motion Parameters** конвертируется в универсальный тип данных и записывается в секцию INI-файла с использованием ВП Write Section Cluster*


Torque Hysteresis.ini - Notepad

File Edit Format View Help

```
[Motion Parameters]
Velocity Amplitude (rpm)=30.000000
Angle Amplitude (deg)=360.000000
Velocity Ramp Angle (deg)=30.000000
Torque Units (in oz)=In Oz
Torque Limit (torque units)=37.500000
```

*Рис. 6.14б. Секция **Motion Parameters** INI-файла как результат операции на рис. 6.14а*

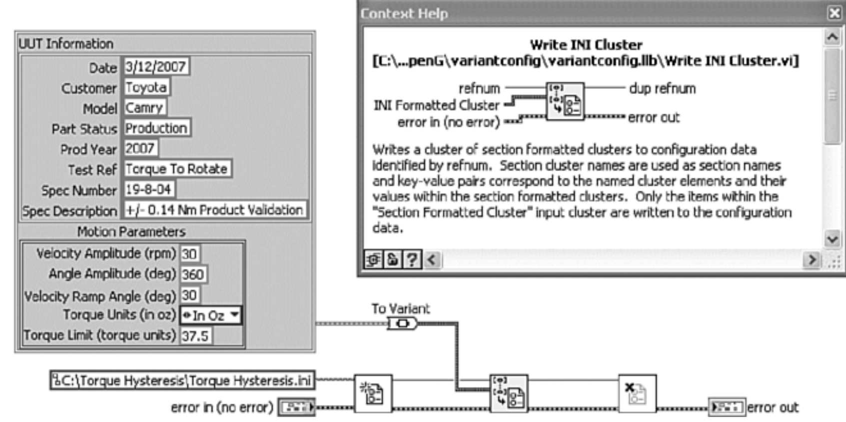


Рис. 6.14в. Вложенный кластер заполняет секции INI-файла, используя ВП Write INI Cluster

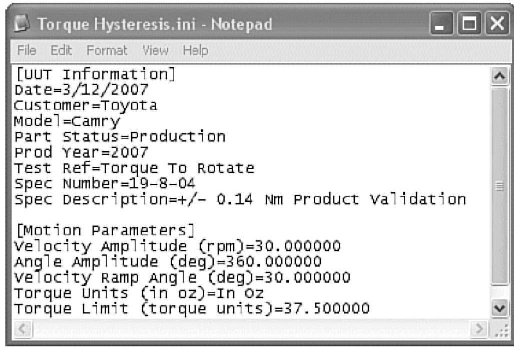


Рис. 6.14г. INI-файл, сформированный на рис. 6.14в, содержит много секций

тер, содержащий кластеры для каждой секции, соединен с входным терминалом **INI Formatted Cluster**. Конечный INI-файл показан на рис. 6.14г. В этих примерах универсальный тип данных используется для минимизации усилий разработчика.

6.4.3. Случайные данные

На рис. 6.15а содержится структура данных, собранная довольно случайным образом. Она состоит из массива кластеров, состоящих из кластера и трех массивов кластеров. Эта структура кажется растрепанной потому, что нарушает несколько правил, касающихся лицевой панели, изложенных в главе 3. Альтернативный вариант, представленный на рис. 6.15б, содержит несколько улучшений, включая ясные метки с именами, упорядоченные и сжатые элементы управления и полужирные метки с единицами в скобках. Кроме того, эти структуры данных содержат больше элементов с меньшим числом вложений. Множество двумерных массивов кластеров на рис. 6.15б, в общем, более эффективны с точки зрения использования памяти, чем вложенная структура на рис. 6.15а. Однако в альтернативном варианте в четыре раза возросло число проводников данных и терминалов ВПП. Лучшим вариантом, вероятно, было бы сохранить единственную структуру данных, как на рис. 6.15а с организованностью и аккуратностью рис. 6.15б.

6.4.4. ВП Centrifuge DAQ

Этот ВП уже обсуждался в главах 3 и 4, данное приложение осуществляет динамические измерения образцов почвы и центрифуге. Лицевая панель изображена на рис. 6.16а и позволяет пользователю настроить большое множество элементов. Оборудование поддерживает до 6 SCXI-шасси, содержащих любые комбинации реле, акселерометров, LVDT, динамометров, термопар. Любое число этих элементов можно расположить в любые слоты любого шасси. Для каждого типа подключаемых модулей существует различный набор программируемых параметров, та-

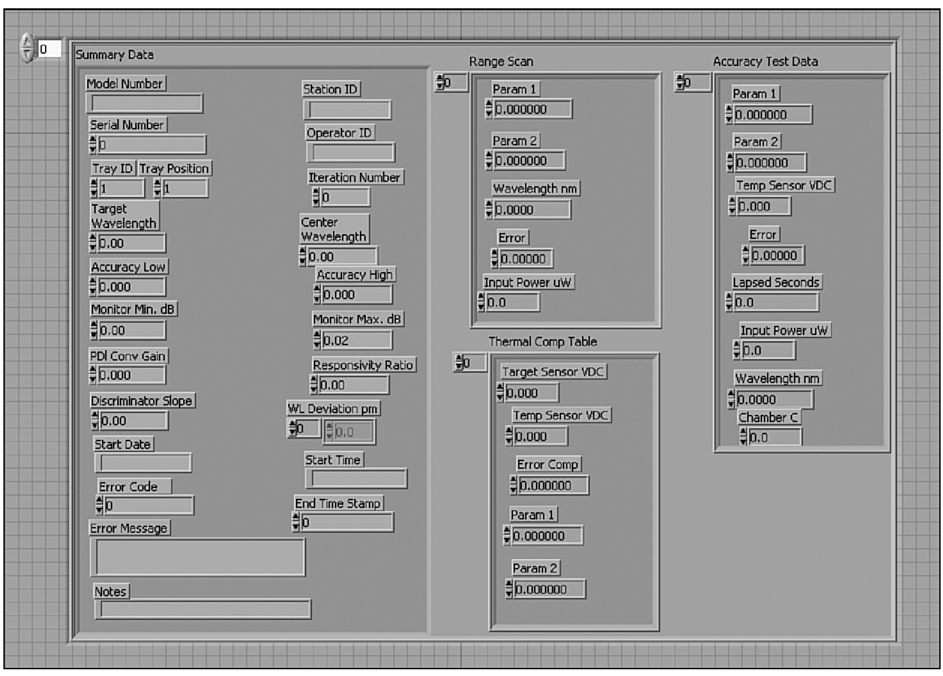


Рис. 6.15а. Вложенная структура данных со случайно расположенными элементами управления

ких как напряжение возбуждения, коэффициент усиления сигнала, фильтрация, масштаб, инженерные единицы. Эти параметры настраиваются с помощью ряда вложенных структур. Когда пользователь выбирает тип сенсора в списке **Sensor Selection** слева, становится доступной одна из семи структур. Каждая структура данных представляет собой массив кластеров, содержащих по одному элементу на каждый SCXI-модуль. Это позволяет настраивать каждый модуль независимо. На рис. 6.16б показана структура данных, используемая для настройки динамометров. Ее внешний вид программно контролируется узлом свойств. На лицевой панели элемент управления индексом массива обычно не виден, и задний фон элемента прозрачный.

На рис. 6.16в показан кусок кода настройки приложения. Множество вложенных структур данных, содержащих данные конфигурации для каждого типа модулей, соединены в другую более сложную структуру. Основная цель этой большой структуры – уменьшить число терминалов и связанных с ними проводников данных для следующего ВПП. Вложение этой структуры данных могло бы стать проблемой, если бы к ней осуществлялся доступ во время процедуры, чувствительной ко времени реализации. Однако эта структура данных содержит только данные конфигурации, доступ к которым осуществляется только перед началом процедуры сбора данных. Поэтому сложность этой структуры данных не влияет на производительность (в плане основной задачи) приложения.

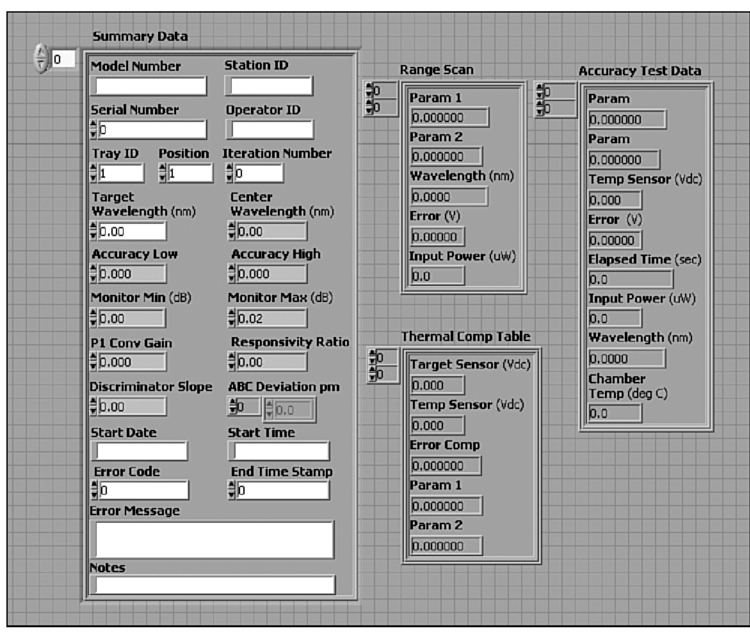


Рис. 6.15б. Вложенная структура данных разделена на четыре, что привело к уменьшению числа вложенных слоев, соблюдение правил оформления лицевой панели привело к невероятному улучшению внешнего вида лицевой панели

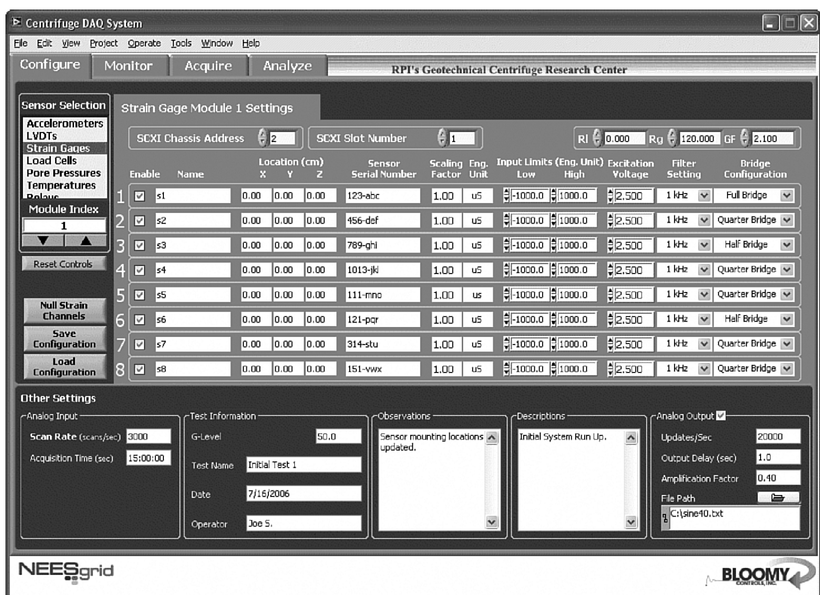


Рис. 6.16а. Лицевая панель для системы сбора данных с центрифуги

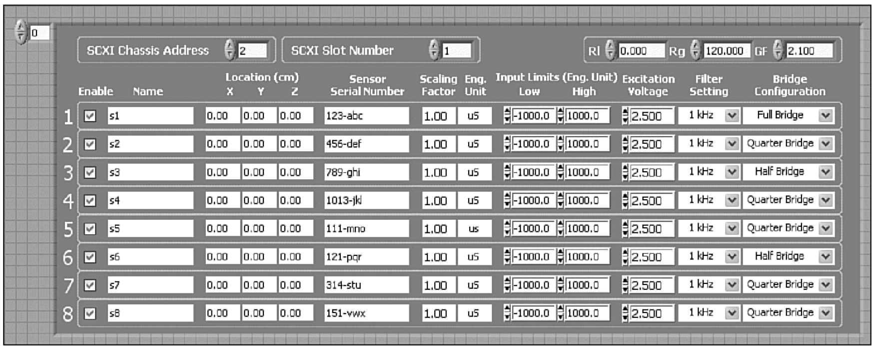


Рис. 6.16б. Этот массив кластеров содержит параметры конфигурации динамометра и является одной из семи структур на вкладке **Configure**, предназначенных для настройки соответствующего типа SCXI-модулей. На лицевой панели элемент управления индексом массива не виден, задний фон – прозрачный

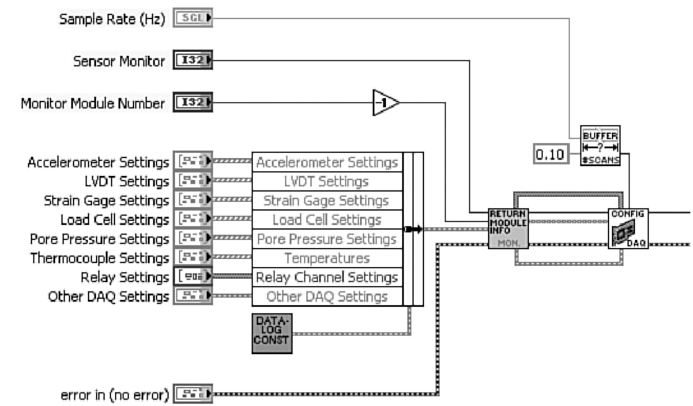


Рис. 6.16в. Процедура настройки соединяет структуры данных в сложную структуру данных, которая затем передается в ВПП. Каждая структура данных – это массив кластеров, содержащих параметры конфигурации разных типов модулей, по аналогии с рис. 6.16б

Ссылки

1. В теме «ВП Memory Usage» он-лайн справки по LabVIEW содержится много полезной информации, касающейся использования памяти.
2. NI Instrument Driver Guidelines, Control/Indicator Naming и Data Representation.
3. OpenG – это организованное сообщество, занимающееся разработкой и использованием инструментов LabVIEW с открытым кодом. Адрес в Интернет: www.openg.com.

Обработка ошибок 7

Типичное приложение LabVIEW управляет контрольно-измерительными приборами во время сбора, анализа, отображения и сохранения данных – 4 основных операций работы с данными. Однако все может пойти неправильно, вне зависимости от методологии графического программирования, уровня сложности предложения и мастерства разработчика. Приложения LabVIEW выполняются точно так, как запрограммированы, и не обладают иммунитетом к ошибкам. Кроме того, мы живем в динамичном мире, и приложения могут начать давать сбой спустя большее время после тестирования. Например, может «зависнуть» или отключиться прибор, или изменения в конкретном вычислительном устройстве могут вызвать конфликт ресурсов, или изменится путь к сетевой директории. Даже если в приложении на стадии разработки не было ни одной ошибки, это не поможет справиться с неожиданностями без эффективной системы обработки ошибок. Более того, то, как быстро разработчик и пользователь смогут определить и исправить возникшую ошибку, зависит от того, как хорошо приложение поддерживает обработку ошибок.

> ***Теорема 7.1*** *Обработка ошибок является сутью диагностики неисправностей, борьбы с неожиданностями и примером хорошего стиля!*

Обработка ошибок является основной частью всех приложений LabVIEW и служит многим целям от разработки до установки. Во-первых, обработка ошибок крайне важна при отладке приложения. Когда приложение сообщает об ошибке, мы можем быстро определить и исправить многие проблемы, опираясь на информацию об ошибке. Обработка ошибок помогает определить и описать ошибки задолго до того, как мы обнаружим некорректное поведение приложения. Отладка приложения без обработки ошибок аналогична графическому программированию с завязанными глазами. Невероятно тяжело обнаружить источник большинства проблем без хорошей обработки ошибок.

После отладки обработка ошибок позволяет нам протестировать пределы совместимости приложения. Во время тестирования можно проверить работу с максимальным числом каналов, определить наибольшую частоту сбора данных, установить самые требовательные процедуры анализа, определить скорость обновления графического интерфейса и частоту обращений к жесткому диску или протестировать самые невероятные комбинации входных данных. Обработка ошибок может рассказать нам, когда что-то пойдет не так, и указать операцию, на которой возникла проблема. Таким образом, можно произвести тонкую настройку приложения, например установить диапазоны значений для численных элементов управления, чтобы предотвратить ввод пользователем некорректных данных, которые могут стать причиной сбоя приложения.

Когда приложение содержит полную обработку ошибок, тщательно протестировано и не вызывает новых ошибок, мы можем быть уверенными в том, что это надежное приложение. После установки приложения процедура обработки приложения должна сохраниться, с тем чтобы доложить о новых ошибках, которые могут возникнуть из-за непредвиденных обстоятельств. Множество факторов нельзя контролировать в LabVIEW. Изменения в программном и аппаратном обеспечении, конфигурации сети – типичные источники новых ошибок в уже отлаженном приложении. Наличие тщательной обработки ошибок помогает пользователям и разработчикам быстро определять и устранять возникшие проблемы. Таким образом, тщательная обработка ошибок служит цели превентивной технической поддержки в наших приложениях.

Обработка ошибок важна не только для определения проблем, но и является признаком хорошего стиля программирования в LabVIEW. Как уже обсуждалось в главе 4 «Блок-диаграмма», поток данных – это фундаментальный принцип LabVIEW, а распространение кластера ошибок – основной момент в установлении потока данных. Кластеры ошибок в LabVIEW наиболее узнаваемые структуры данных на лицевой и соединительной панелях и блок-диаграмме ВП. Используйте их правильно!

7.1. Основы обработки ошибок

Ошибка – это ситуация, когда функция или ВП не могут завершить запрограммированное задание. Большинство узлов в LabVIEW распространяют данные об ошибке через кластеры **error in** и **error out**. Кластеры ошибок состоят из логической переменной, отражающей статус целого числа, которым записывается код ошибки, и строки, соответствующей источнику. Кластеры ошибок расположены на вкладке **Array**, **Matrix & Cluster**. Статус показывает, произошла ли ошибка. Код ошибки определяет ее тип и используется ВП, формирующими отчет об ошибке, чтобы найти соответствующее описание. Источник определяет функцию или ВП, где произошла ошибка. По умолчанию состояние кластера ошибок следующее: статус = FALSE, код = 0, источник = <пустая строка> – отражает отсутствие ошибок. Если статус = FALSE, а код не равен нулю и строка источника не пустая, значит, имеет место предупреждение. Предупреждение похоже на ошибку, но считается менее критичным. Например, узел может успешно завершить задание, но входные данные или результат необычны.

Примеры кластера ошибок **error out** без ошибки, с предупреждением и с ошибкой представлены на рис. 7.1. Значение по умолчанию – без ошибки – изображено слева. Предупреждение, показанное в центре, обычно получается после успешного обращения к функции VISA Read. Оно показывает, что число переданных байт равно затребованному, но можно получить и больше данных. Кластер ошибок справа содержит ошибку с кодом 43, что означает отмену операции пользователем. Это очень распространенная ошибка, которая означает, что было закрыто диалоговое окно. Большинство ВП имеют по одному элементу управления **error in** и индикатору **error out**, расположенных в нижнем левом и правом углах соединительной панели соответственно.

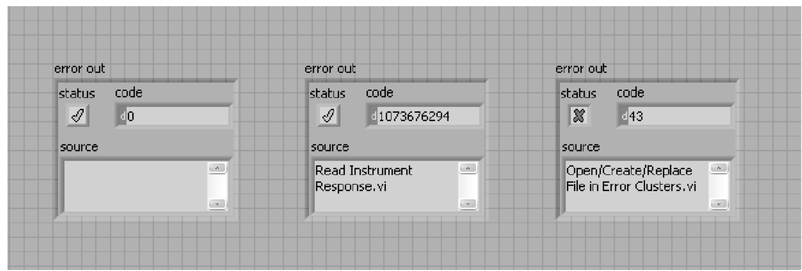


*Рис. 7.1. Кластеры **error out** показывают отсутствие ошибки или предупреждения слева, предупреждение в центре и ошибку справа*

Правило 7.1 Все ВП должны захватывать ошибки и сообщать об ошибках, возвращаемых терминалами ошибок

Обработка ошибок включает в себя создание ловушек для ошибок и отчетов об ошибках, возвращаемых функциями и ВП, которые имеют терминалы ошибок. Организация ловушки – это захват ошибки, возвращаемой терминалом ошибки каждого узла. Создание отчета включает в себя отображение или запись информации об ошибке с использованием диалогового окна или файла. Тщательная организация ловушек и отчетов – лучший способ обработки ошибок.

В следующем разделе представлены основы обработки ошибок, включая правила, технику и иллюстрации по организации ловушек и созданию отчетов об ошибках. Также обсуждаются коды ошибок и диапазоны.

7.1.1. Отслеживание ошибок

Правило 7.2 Отслеживайте ошибки, передавая кластер ошибок между терминалами ошибок

Ошибки можно поймать в ловушку, распространяя кластер ошибок через все узлы, которые имеют терминалы ошибок во всех ВП в приложении. Начните с размещения кластеров **error in** и **error out** на лицевой панели каждого ВП или просто используйте шаблон, на котором они уже есть. На блок-диаграмме располагайте узлы в группы по данным или по последовательности исполнения слева направо в том порядке, в каком они будут выполняться. Используйте инструменты выравнивания по вертикали, расположенные в выпадающем меню **Align Objects**, чтобы выровнять иконки и упростить соединение. Используйте инструменты распределения по горизонтали из выпадающего меню **Distribute Objects**, чтобы разделить узлы для разумного доступа к терминалам. Каждая группа последовательных узлов составляет цепочку ошибок.

Проведите проводник данных от элемента управления **error in** к терминалу ошибок **error in** первого узла каждой цепочки ошибок. Соедините терминал **error out** первого узла с терминалом **error in** следующего узла в цепочке и повторите процедуру для всех узлов, кроме последнего, для каждой цепочки ошибок. Для ВП с одной цепочкой ошибок соедините терминал **error out** последнего узла с индикатором **error out** или соответствующей ВПП. Если ВП обладает несколькими цепочками ошибок, объедините терминалы **error out** последних узлов каждой цепочки, используя ВП Merge Errors. На рис. 7.2 показана простая цепочка ошибок, сделанная из узлов ввода/вывода файлов. На рис. 7.3 показаны две параллельные цепочки ошибок, состоящие из узлов ВП DAQmx и узлов ввода/вывода файлов.

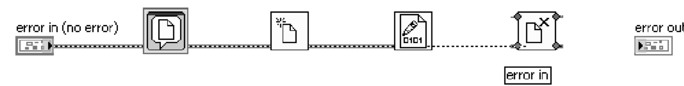


Рис. 7.2. Кластер ошибок распространяется через несколько узлов ввода/вывода файлов, создавая цепочку ошибок

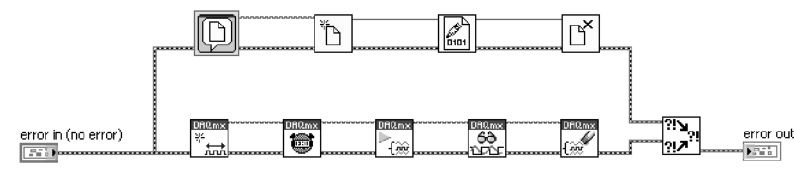


Рис. 7.3. ВП DAQmx и узлы ввода/вывода файлов формируют две параллельные цепочки ошибок, которые соединяются ВП Merge Errors

Большинство функций и ВП в LabVIEW разработаны так, чтобы проверять статус ошибки из терминала **error in**, и если ошибка есть, не выполняют собственный код и передают эту ошибку дальше через терминал **error out**. Поэтому, когда случается ошибка, последующие узлы цепочки ошибок передают информацию об этой ошибке по всей цепочке. Это желательно, потому как узлы цепочки, как правило, связаны и зависят от друга. На рис. 7.1, где показана цепочка узлов ввода/вывода файлов, операция записи зависит от правильности refnum, возвращаемого функцией Open/Create/Replace File, (Открыть/Создать/Заменить

файл), которая сама зависит от правильности пути возвращаемого экспресс-ВП **File Dialog** (Диалог выбора файла). Если все терминалы ошибок соединены правильно, то первая возникшая ошибка оказывается пойманной внутри цепочки узлов и проводников данных.

Как уже обсуждалось в главе 4, распространение кластера ошибок создает зависимость данных, определяющей порядок, в котором выполняются узлы на блок-диаграмме. Зависимость данных – это принцип, лежащий за понятием потока данных. Поэтому важно соединять кластеры ошибок и узлы с тем, чтобы задать желаемый порядок выполнения. Однако наиболее частая ошибка, с которой я встречался, – это незавершенная ловушка для ошибок. Похоже, многие разработчики используют кластер ошибок, чтобы создать зависимость данных, но не завершают создание ловушки для ошибок и отчетов.

Правило 7.3 Отслеживайте ошибки на всех итерациях цикла

Необходимо сделать несколько замечаний о циклах. Необходимо изучить поток данных и условия цикла, чтобы удостовериться, что ошибки отлавливаются на всех итерациях. На рис. 7.4а между терминалами ошибок узлов ввода/вывода файлов и элементами управления **error in** и отображения **error out** нет разрывов. На первый взгляд кажется, что ошибка будет поймана, но это не так. Вместо этого входной туннель цикла обнуляет все данные об ошибках на каждой итерации цикла. Также передающиеся на выходной туннель цикла данные переписываются на каждой последующей итерации. Только данные об ошибках на последней итерации будут переданы по выходному туннелю. Таким образом, цепочка ошибок не помнит никаких ошибок, которые могут произойти на всех итерациях, кроме последней. Есть две альтернативы: или прерывать цикл после появления первой ошибки, или передавать данные об ошибках между итерациями, используя сдвиговый регистр. Эти альтернативы показаны на рис. 7.4б и 7.4в. Первое решение

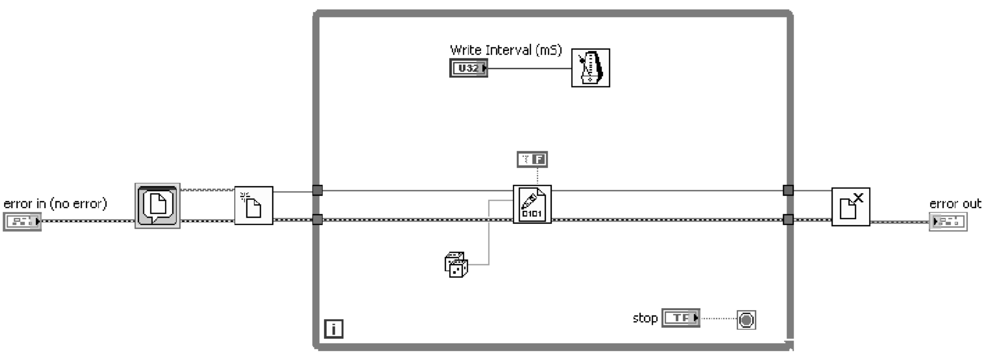


Рис. 7.4а. Данные об ошибке на входе в цикл обнуляются, а данные об ошибке на операции записи файла переписывают данные с выхода цикла на каждой итерации цикла

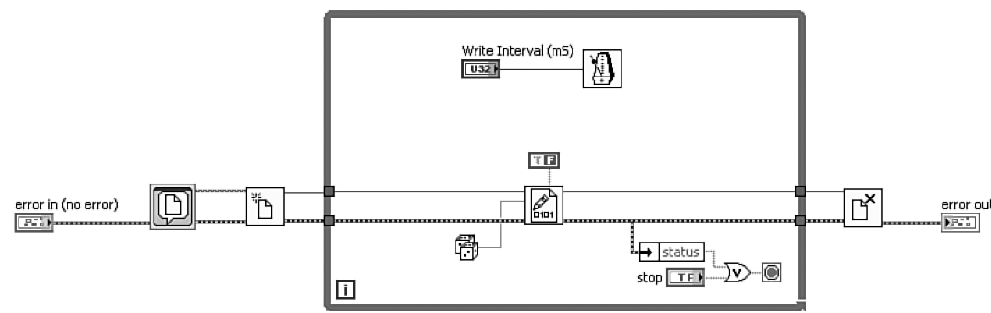


Рис 7.4б. На каждой итерации цикла проверяется статус ошибки, и если ошибка случается, то цикл прерывается. Любая ошибка будет поймана и передана через выходной туннель цикла на индикатор **error out**

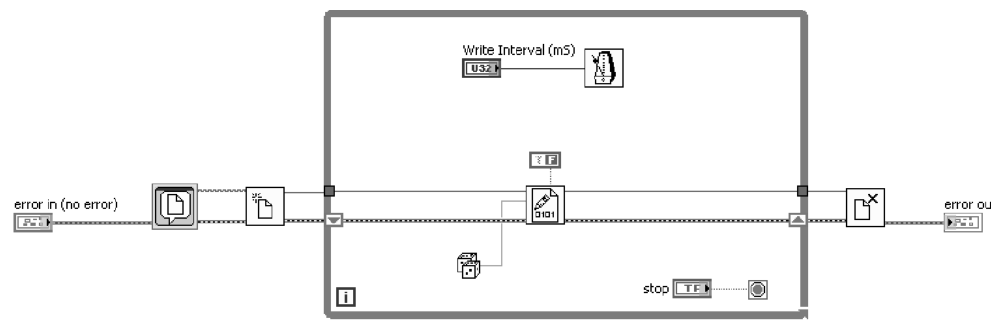


Рис. 7.4в. Для передачи данных об ошибках между итерациями цикла используется сдвиговый регистр. Если случается ошибка, то выполнение цикла продолжается, но функция записи файла не выполняется

останавливает цикл, если происходит ошибка, второе продолжает выполнение цикла, но проскакивает все последующие операции ввода/вывода файлов. В обоих случаях ошибка будет поймана и передана из цикла на индикатор **error out** по завершении цикла.

Правило 7.4 Отключите индексацию ошибок в продолжительных циклах

Обратите внимание, что цикл For выполняется заранее определенное число раз, не может быть прерван раньше времени, и по умолчанию включена индексация выходных терминалов цикла. Это означает, что кластер ошибок, проходящий через выходной терминал цикла, будет накапливаться в массив размером, равным числу итераций цикла. Если у вас есть весомая причина накапливать все ошибки и количество итераций цикла ограничено, индексируйте ошибки. Вне цикла For можно обрабатывать массивы ошибок при помощи ВП Merge Errors, как показано на рис. 7.5а. Merge Errors (ВП слияния ошибок) ищет и возвращает первый эле-

мент массива со значением status = TRUE. Никогда не следует вводить индексацию, если количество итераций цикла не определено.

По аналогии с циклом While, лучшим способом передачи данных об ошибках между последовательными итерациями цикла For является метод сдвиговых регистров, показанный на рис. 7.5б. Метод сдвиговых регистров увеличивает производительность приложения, поскольку функции и ВП, которые получают ошибку на вход терминала **error in**, не выполняют свой код. Также этот метод улучшает использование памяти, так как в памяти в каждый момент времени содержится только один кластер ошибок. Более того, сдвиговый регистр удлиняет цепь ошибок через весь цикл, тогда как индексация начинает цепь заново на каждой итерации. Поэтому метод сдвиговых регистров в общем случае является предпочтительным.

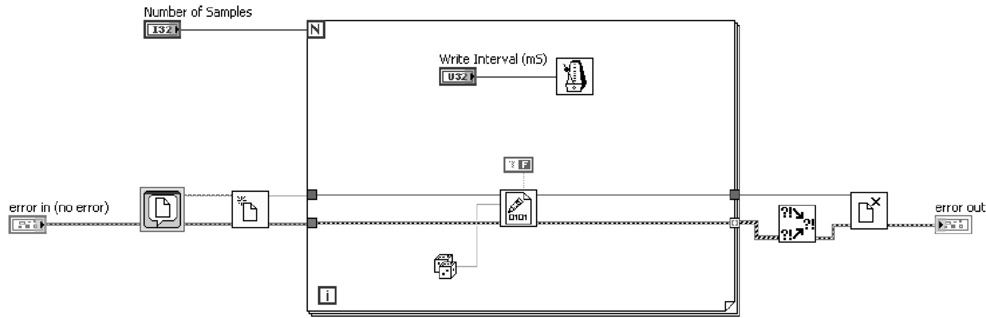


Рис. 7.5а. По умолчанию цикл For индексирует массив ошибок.
ВП Merge Errors возвращает первый элемент этого массива с status = TRUE

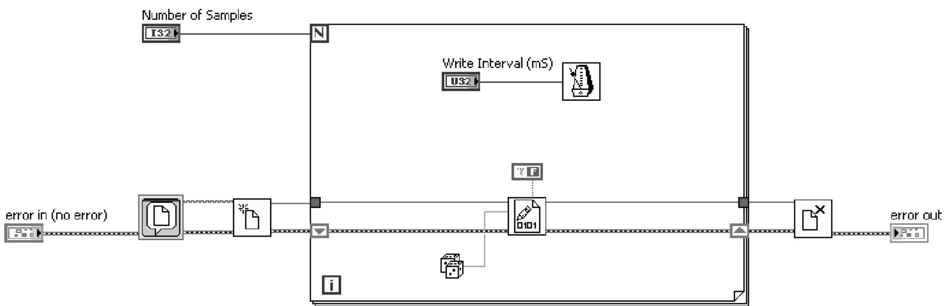


Рис. 7.5б. Сдвиговый регистр удлиняет цепь ошибок между итерациями цикла.
Такой подход более эффективный

Многие ВП средней или высокой сложности выигрывают от применения параллельных цепочек ошибок (см. рис. 7.3). Параллельные цепочки ошибок особенно полезны, когда параллельные цепочки узлов используют общие данные,

такие как имя ввода/вывода или refnum. Например, на рис. 7.3 задание передается через ВП DAQmx, и refnum файла передается через узлы ввода/вывода файлов. Расположение этих групп параллельно помогает облегчить распространение задания, refnum'а и кластера ошибок между узлами.

Однако в некоторых приложениях параллельные узлы функционально независимы и должны быть соединены в обычную цепочку ошибок. В этом случае узлы могут быть расположены параллельно, чтобы облегчить распространение refnum и имени ввода/вывода, но обе группы будут использовать одну цепочку ошибок.

Например, на рис. 7.6а ВП DAQmx получает данные в виде волнового сигнала, которые отображаются на графике, в то время как функции ввода/вывода файлов записывают случайные данные в файл. Эти параллельные узлы функционально независимы, поэтому используются раздельные цепочки ошибок. На рис. 7.6б ВП **DAQmx** получает данные, которые записываются в файл с использованием функций ввода/вывода. В этом случае параллельные узлы функционально зависимы, а именно – функция Write to Binary File зависит от данных DAQmx (ВП сбора данных). Если он возвращает ошибку, то данные, которые она возвращает, неправильные и не должны быть записаны в файл. Аналогично и сбор данных не должен происходить, если произошла ошибка на стадии выбора или создания файла. В этом примере параллельное расположение узлов упрощает распространение данных, а использование одного кластера ошибок создает зависимость от ошибок, что увеличивает эффективность.

Правило 7.5 Отслеживайте все ошибки от всех узлов, которые имеют терминалы ошибок

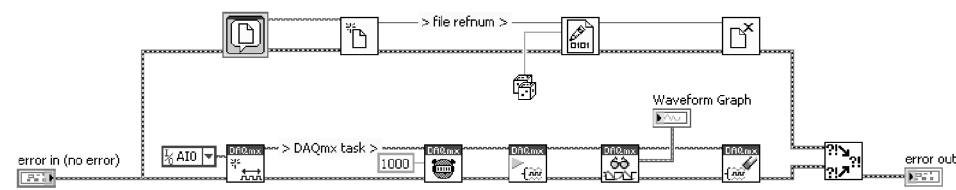


Рис. 7.6а. ВП DAQmx и функции ввода/вывода файлов независимы, используются параллельные цепочки ошибок

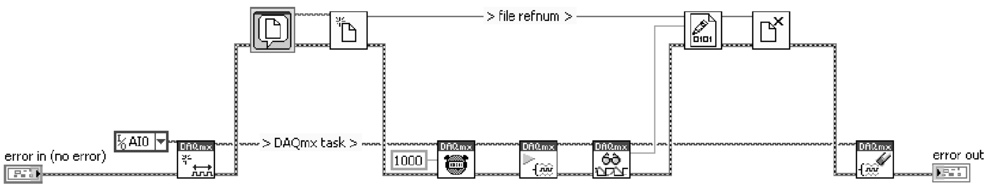


Рис. 7.6б. ВП DAQmx и функции ввода/вывода взаимосвязаны. Использование общей цепочки ошибок упрощает зависимость от ошибок и увеличивает эффективность

Для лучших результатов отслеживайте все ошибки от всех узлов, которые имеют терминалы ошибок. Это увеличит надежность приложения. Некоторые исключения из этого правила будут рассмотрены в разделе 7.3 «Приоритеты ошибок». Однако в общем случае я рекомендую отслеживать ошибки со всех терминалов ошибок.

7.1.2. Отчеты об ошибках

Правило 7.6 Выводите сообщение об ошибках в диалоговом окне и/или файле отчета

Когда ошибка зафиксирована, необходимо создать отчет об этой ошибке, используя диалоговое окно или запись в файл. Диалоговое окно – самый простой и наиболее популярный способ отчета об ошибке. Оно состоит из вызова ВП Simple Error Handler (Простой обработчик ошибок) или ВП General Error Handler (Более сложный обработчик ошибок). Эти ВП расположены на палитре **Dialog&User Interface**, изучают ошибку, которая передается на их терминал **error in**, ищут описание этой ошибки в базе LabVIEW по коду ошибки и создают сообщение с описанием ошибки. По умолчанию каждый ВП открывает диалоговое окно, которое отображает код ошибки, источник и описание, которое пользователь должен изучить. Пример приведен на рис. 7.7.

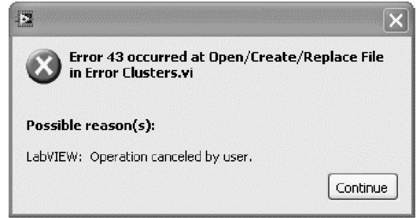


Рис. 7.7. ВП Simple Error Handler и ВП General Error Handler открывают диалоговое окно, отображающее код ошибки, источник и описание, которое пользователь должен изучить

Правило 7.7 Используйте ВП General Error Handler, а не ВП Simple Error Handler

ВП General Error Handler дает возможность обрабатывать определенные пользователем ошибки и исключения. ВП Simple Error Handler – ни что иное, как вызов ВП General Error Handler с меньшим числом доступных терминалов. Иконка, метки терминалов и описание каждого ВП приведены в окне контекстной справки на рис. 7.8а. Интересно, что Simple Error Handler гораздо чаще используется в поставляемых примерах LabVIEW и кажется более популярным. Однако

этот тривиальный ВПП нарушает Правило 4.9. Его блок-диаграмма показана на рис. 7.8б. Она устроена просто, но этого недостаточно, чтобы оправдать ее существование. Вместо этого она снижает гибкость приложения и добавляет ненужные операции. Каждый слой ВПП обладает номинальным временем выполнения, включая вызов ВП Simple Error Handler, который содержит ВП General Error Handler. Поэтому ВП General Error Handler более гибкий и эффективный способ отчета об ошибке в диалоговом окне.

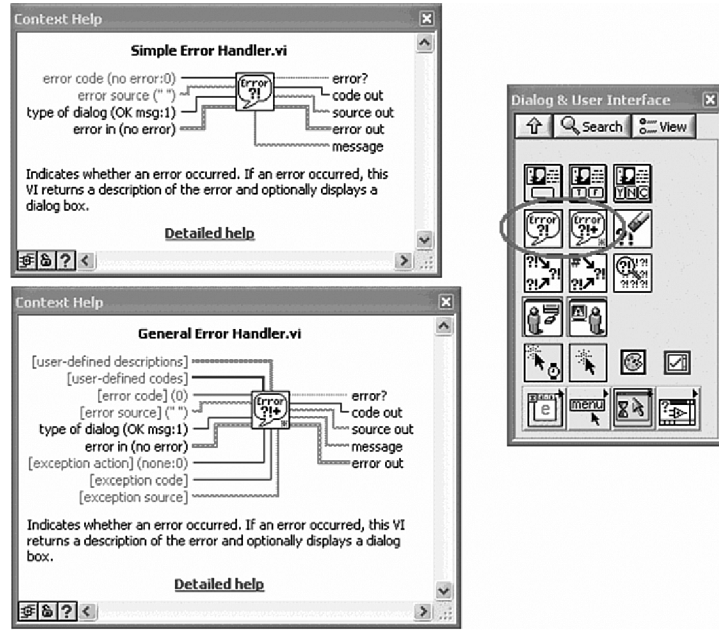


*Рис. 7.8а. ВП Simple или General Error Handler находятся в палитре **Dialog&User Interface**. Основное различие – в количестве отображаемых терминалов*

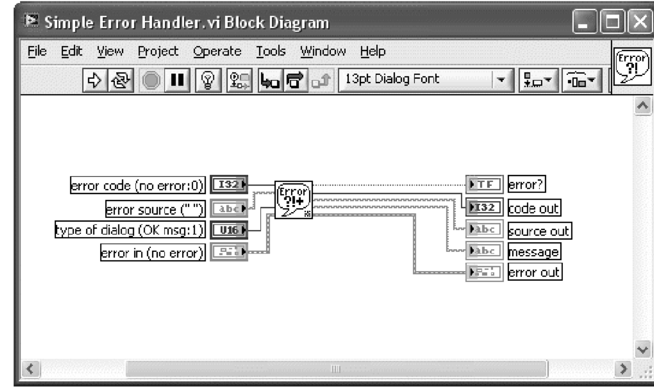


Рис. 7.8б. Блок-диаграмма ВП Simple Error Handler содержит вызов ВП General Error Handler. Этот тривиальный ВПП нарушает Правило 4.9

Правило 7.8 Используйте запись в лог-файл при внедрении приложения

После того как приложение внедрено с использованием диалогового метода сообщения об ошибках, чаще всего пользователям самим необходимо передавать информацию об ошибке разработчику. Без особых инструкций пользователь может не записать информацию об ошибке, или та информация, которую пользователь записал, может быть неполной или неправильно передана разработчику. Кроме того, такие диалоговые окна могут смущать и раздражать многих пользователей. Метод записи ошибок в лог-файл состоит из простой процедуры программной записи информации об ошибках в файл.

Существует множество способов реализовать такую процедуру. Основные моменты: удобочитаемость и производительность. Для максимальной читабельности записывайте полное описание ошибки, включая код, источник, дату и время, в текстовый файл. Как показано на рис. 7.9а, функция Get Date/Time String (Вернуть строку даты и времени), доступная с палитры **Timing**, используется для создания даты и времени как ASCII-строки. ВП General Error Handler возвращает описание ошибки. Данные с терминала **message output** (результат) комбинируются с кодом, источником, датой и временем ошибки и записываются в текстовый файл. Соответствующий лог-файл может быть прочитан в текстовом редакторе. Важно соединить входной терминал **type of dialog** (тип диалога) ВП General Error Handler и установить в соответствующем перечне **no dialog** (нет диалога), если вы хотите запретить появление диалогового окна.

Второй метод записи информации об ошибке в лог-файл использует формат datalog для увеличения производительности в ущерб читабельности. На рис. 7.9б показано, как кластер ошибок соединяется с временной меткой и соответствующий кластер записывается в файл. Функция Get Date/Time in Seconds, доступная с палитры **Timing**, возвращает текущее время в формате временной метки. Функции Open/Create/Replace Datalog, Write Datalog и Read Datalog доступны с палитры **File I/O ⇒ Datalog**. Они позволяют работать с данными в высокопроизводительном бинарном формате, однако информация в заголовке ограничена. Этот метод очень легко реализовать. Однако данные не читаются ничем, кроме функций LabVIEW. Процедура считывания datalog-файлов должна состоять из функции Open/Create/Replace Datalog (Открыть/Создать/Заменить файл), на вход

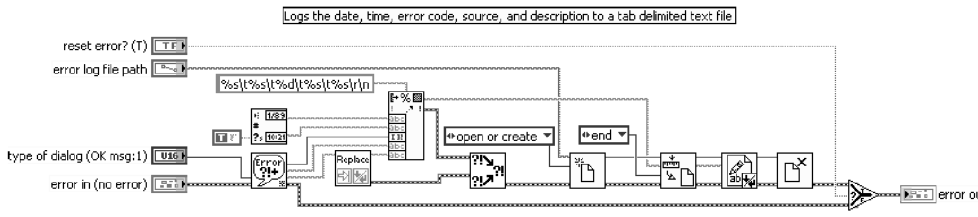


Рис. 7.9а. Код, источник, дата и время ошибки комбинируются в один кластер и записываются в текстовый файл

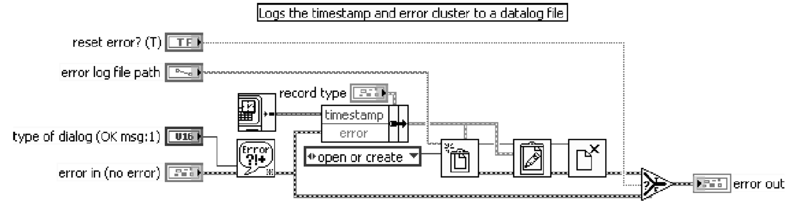


Рис. 7.9б. Производительность увеличена в ущерб читабельности путем записи данных об ошибке в формате datalog. Получившийся файл можно прочитать только средствами LabVIEW

record type (тип данных) которой необходимо подать кластер, и функции Read Datalog. Используйте этот метод, если существует возможность наличия множества ошибок и важна производительность или если хотите зашифровать содержание файла в бинарном формате. Однако текстовые файлы куда более универсальны, и более предпочтительно использовать именно их.

Важный момент, касающийся лог-файлов, – синтаксический разбор ошибок при записи в файл. При некоторых обстоятельствах, таких как создание отчета об ошибке внутри циклов, которые не заканчивают работу при обнаружении ошибки, добавление каждой ошибки в один файл может привести к созданию большого или неконтролируемого файла. Большие лог-файлы сложно загружать в текстовые редакторы, и они могут содержать избыточную информацию. Аналогично, создание нового файла под каждую новую ошибку предотвращает создание больших файлов, но создает огромное количество самих файлов. Как вариант рассмотрим умную процедуру записи ошибок в лог-файлы, которая ограничивает размер и количество ошибок, записанных в лог-файл. Такая процедура будет считывать содержание лог-файла или добавлять запись в файл, или переписывать самую старую запись, или вообще не будет производить запись. Количество записываемых ошибок, размер файла, наличие избыточных кодов ошибок может служить критерием добавления, замены или отмены записи в такой процедуре.

Очень важно оповестить пользователя или разработчика, что произошла ошибка. Для достижения наилучших результатов комбинируйте диалоговое окно или индикатор на лицевой панели с записью в файл. Диалоговое окно или индикатор оповестят пользователя о том, что произошла ошибка, а в лог-файле будет содержаться полная информация об этой ошибке. Индикатор ошибки может быть логическим и показывать, что была зарегистрирована ошибка, или численным элементом, отображающим количество ошибок, или еще проще – кластером ошибок, показывающим последнюю информацию об ошибке. Как вариант LabVIEW может посылать письмо по электронной почте с вложенным лог-файлом, используя ВП с палитры **SMTP Email**.

Правило 7.9 Запрещайте диалоговые окна ошибок для удаленных или не требующих внимания приложений

Метод диалогового сообщения об ошибках (с записью в файл или без) непрактичен для приложений, не требующих постоянного внимания или управляемых удаленно. Когда открывается диалоговое окно и нет никого, кто мог бы его изучить, то приложение может приостановить свою работу на неопределенный срок. Кроме того, следует помнить, что web-сервер LabVIEW не может публиковать диалоговые окна для удаленных клиентов, которые работают с приложением через Интернет-браузер. В этом случае диалоговое окно откроется только на сервере, но для всех удаленных клиентов оно не будет видно. Следует избегать диалогового отчета об ошибках в таких ситуациях.

Очень важно правильно просчитать процедуру формирования отчета об ошибках. Все цепочки ошибок должны прерываться вызовом ВП, составляющим отчет. Однако следует избегать вызова таких ВП внутри циклов или ВПП, который может быть вызван в цикле, если только не приняты соответствующие меры по устранению ошибок. Иначе ВП, формирующий отчет об ошибке, может вызвать постоянное появление диалогового окна или лог-файл станет слишком большим. Лучше всего ограничить «отчетные» ВП верхним уровнем приложения, это поможет упорядочить отчеты об ошибках и снизить вероятность отчета внутри цикла. Так же полезно слить все ошибки в один ВП формирования отчета. Хороший способ – использовать ВП Merge Errors для того, что бы объединить множество цепочек ошибок, чтобы затем одним ВП сформировать отчет.

На рис. 7.10 показана блок-диаграмма верхнего уровня ВП Torque Hysteresis. Обратите внимание, что ошибки отслеживаются по мере распространения кластера ошибок, отчет же об ошибках создается при помощи ВП General Error Handler. Если случается ошибка, то цикл прерывается, и создается отчет об ошибке. Расположение ВП General Error Handler вне цикла гарантирует, что диалоговое окно появится только один раз, даже если логика завершения цикла изменится. Этот пример использует наиболее распространенную методологию обработки ошибок, примененную к простой архитектуре приложения.

Правило 7.10 Избегайте ВПП с встроенными отчетами об ошибках

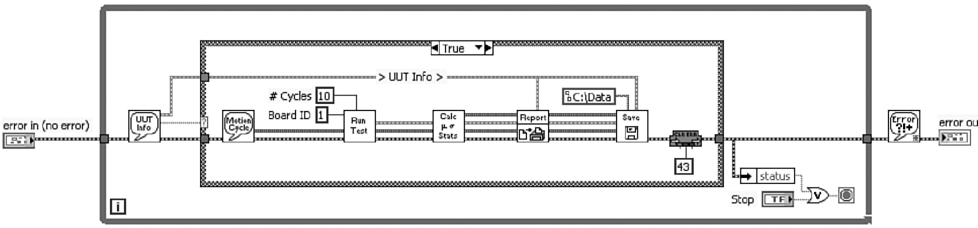


Рис. 7.10. ВП Torque Hysteresis – ошибки отслеживаются по мере распространения кластера ошибок, отчет же об ошибках создается при помощи ВП General Error Handler. Обратите внимание, что ВП General Error Handler вызывается только один раз

Палитры LabVIEW содержат несколько ВП со встроенными отчетами об ошибках. Эти ВП находятся в верхних строках своих палитр и не имеют терминалов ошибок. К ним относятся все ВП, формально известные как Easy I/O (ВП простого ввода/вывода) на палитрах **Data Acquisition**, и несколько высокоуровневых ВП ввода/вывода файлов. Все эти ВП негибкие, предоставляют только диалоговый способ отчета об ошибках. Кроме того, отсутствие у них терминалов ошибок делает проблематичным их использование в концепции «потока данных». Наконец, эти ВП мене производительны, чем низкоуровневые ВП. Таким образом, я рекомендую избегать этих ВП.

7.1.3. Коды ошибок

Кластер ошибок содержит элемент **code**, представленный как 32-битное целое с определенным знаком. Коды используются для идентификации ошибок, которые создают ВП и функции LabVIEW. LabVIEW поддерживает внутреннюю базу данных, в которой код каждой ошибки связан с ее описанием. Категории и диапазоны кодов ошибок для LabVIEW 8.2 показаны в табл. 7.1. Эта таблица связывает диапазоны заранее определенных кодов ошибок с типами операций, для которых они зарезервированы.

Таблица 7.1. Коды ошибок для LabVIEW

Диапазон	Область ошибки
от –2147467263 до –1967390460	Сеть
от –1950679040 до –1950678977	Разделяемые переменные
от –1967362045, –1967362022 до –1967361997 и от –1967345663 до –1967345609	Безопасность
от –1074003967 до –1074003950, –1074000001 и –1074000000	Драйвер прибора
от –1073807360 до –1073807192	VISA
от –90165 до –90149 и от –90111 до –90001	MathScript
от –41005 до –41000 и 41000	Генерация отчета
от –23096 до –23081	Преобразование формулы
от –23096 до –23000, –20141, –20140 и 20005	Математика
от –20337 до –20301	Обработка сигнала
от –20207 до –20201	По точкам
от –20119 до –20001	Обработка сигнала
от –4644 до –4600	Регулярное выражение
от –2983 до –2970, от –2964 до –2960, от –2955 до –2950 и от –2929 до –2901	Управление исходным кодом
–2586, –2585, –2583 до –2581, –2578, –2575, –2574, от –2572 до –2550, от –2526 до –2501 и 2552	Хранение
–1821, –1820, от –1817 до –1800	Осциллограмма
от –1719 до –1700	Apple
от –1300 до –1210	Драйвер прибора
от –823 до –800 и 824	Тактированный цикл
от –620 до –600	Доступ к реестру Windows

Таблица 7.1. Коды ошибок для LabVIEW (окончание)

Диапазон	Область ошибки
0	Обработка сигнала, КОП, драйвер прибора, обработка формулы, VISA
от 1 до 20	КОП
от 1 до 52	Общая ошибка
от 30 до 32, 40 до 41	КОП
от 53 до 66	Сеть
от 61 до 65	Последовательный порт
от 67 до 91	Общая ошибка
от 92 до 96	Связь средствами Windows
от 97 до 100	Общая ошибка
от 102 и 103	Драйвер прибора
от 108 до 121	Сеть
от 116 до 118 и 122	Общая ошибка
от 1000 до 1045	Общая ошибка
от 1046 до 1050 и 1053	Узлы сценариев
от 1051 до 1091	Общая ошибка
1101, 1114, 1115, 1132 до 1134, от 1139 до 1143 и от 1178 до 1185	Сеть
от 1094 до 1157	Общая ошибка
от 1158 до 1169, 1318, 1404 и 1437	Исполнительное меню
от 1172, 1173, 1189 и 1199	Связь средствами Windows
от 1174 до 1188, 1190 до 1194 и от 1196 до 1198	Общая ошибка
от 1301 до 1321, от 1357 до 1366, 1370, от 1376 до 1378, от 1380 до 1391 и от 1395 до 1399	Общая ошибка
1325, 1375, 1386 и 1387	Связь средствами Windows
1367, 1368 и 1379	Безопасность
от 1337 до 1356, 1369 и 1383	Сеть
1371, 1373, 1392 до 1394, от 1400 до 1403, 1448 и 1486	Объектно-ориентированное программирование в LabVIEW
от 1430 до 1455, от 1468 до 1470, от 1483 до 1485, 1487 до 1493 и от 1497 до 1500	Общая ошибка
от 1456 до 1467 и 1471 до 1482	Управление 3D-изображением
от 1800 до 1809 и 1814	Осциллограмма
от 4800 до 4806, 4810, 4811 и 4820 до 4823	Общая ошибка
от 16211 до 16554	SMTP
от 20001 до 20030, 20307, 20334 и от 20351 до 20353	Обработка сигнала
от 56000 до 56005	Общая ошибка
от 180121602 до 180121604	Разделяемая переменная
от 1073479937 до 1073479940	Драйвер прибора
от 1073676290 до 1073676457	VISA

Помимо этого, LabVIEW резервирует коды в диапазоне от –8999 до –8000 и от 5000 до 9999 для ошибок, определяемых пользователем. Если для вашего ВП необходимо отмечать неправильное поведение, причем описания нет в представленной таблице, вы может сами определить новый код ошибки в указанных диапазонах. Для создания отчетов об ошибках, определенных пользователем, можно использовать ВП General Error Handler. Просто определите коды ошибок и описания, как массивы целых чисел и строк, и соедините их с соответствующими терминалами. Этот метод очень легко использовать, если приложение содержит одну или ограниченное число процедур отчета об ошибках. Однако использовать данный метод в случае, если в приложении используется несколько экземпляров ВП General Error Handler, требующих множества кодов ошибок, определенных пользователем, может быть затруднительно.

Правило 7.11 Храните коды ошибок, определяемые пользователем, в XML-файле

Используйте XML-файлы, чтобы хранить коды ошибок и их описание, а именно, зарегистрируйте собственные коды ошибок, создав XML-файл в директории LabVIEW\user.lib\errors folder. Это можно сделать при помощи Error Code File Editor (**Tools ⇒ Advanced ⇒ Edit Error Codes**). Такой способ имеет несколько преимуществ над определением кодов ошибок на блок-диаграмме. Во-первых, вам не нужно соединять дополнительные входы ВП General Error Handler. Во-вторых, заранее определенные коды ошибок может использовать любой ВП, работающий в той копии LabVIEW, которая содержит XML-файл. Наконец, функция Explain Error распознает и объяснит ошибку, определенную таким образом. **Explain Error** – это встроенная утилита, которая отображает информацию о коде ошибки в диалоговом окне. Она активируется выбором пункта **Explain Error** в меню быстрого запуска любого кластера ошибок: **Help ⇒ Explain Error**. Когда внедряете приложение, убедитесь что инсталлятор или распространяемый источник включает в себя XML-файл ошибок.

Правило 7.12 *Используйте отрицательные коды для ошибок устройств ввода/вывода и положительные – для предупреждений*

При использовании устройств I/O (ввода/вывода), таких как оборудование сбора данных, *положительные* коды используются для представления предупреждений, которые могут не влиять напрямую на выполнение приложения, и *отрицательные* для представления значительных ошибок, которые требуют действий по их устранению или прерывания работы ВП.

7.2. Обработка ошибок в ВПП

Большинство функций и ВП в LabVIEW, обладающие терминалами ошибок, оценивают значение логического параметра **status** и не выполняют свой код, если это значение TRUE. Это улучшает способность приложений быстро и эффективно восстанавливаться после ошибок и создавать отчеты о них. Например, функции ввода/вывода, такие как VISA Read, могут подождать 10 секунд для ответа уст-

ройства, прежде чем возвращать ошибку «по истечении времени» (timeout), вызывающему приложению. Ошибка такого порядка, скорее всего, снизит производительность приложения. Если аналогичное устройство запрашивает выполнение с аналогичными результатами, то приложение, скорее всего. зависнет или будет нестабильно. Предотвратить такую ситуацию можно, отследив ошибку и передав ее через все последовательные узлы в цепочке ошибок. Тогда все последующие запросы устройства, которые получат ошибку, не будут выполнены, что ускорит передачу ошибки до процедуры отчета об ошибках, и приложение сможет изящно восстановить свою работу.

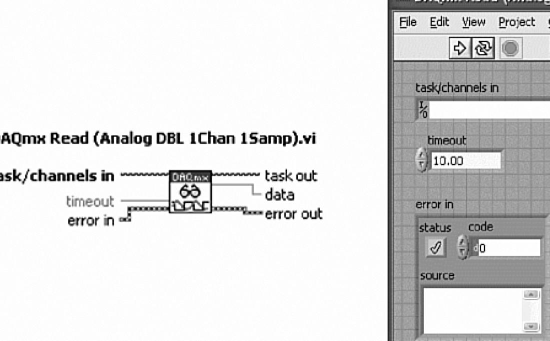
Правило 7.13 Пропускайте большинство блок-диаграмм ВПП при возникновении ошибок, используя Error Case Structure

Желательно чтобы ВПП имитировали обработку ошибок, свойственную функциям и ВП LabVIEW. А именно – большинство ВПП должны пропускать выполнение блок-диаграммы, когда получают на вход **error in** информацию об ошибке. Это осуществляется путем использования Structure Error Case (Структура варианта для ошибок) следующим образом. Во-первых, поместите стандартные кластеры **error in** и **error out** на панели и подсоедините их к терминалам в нижнем левом и нижнем правом углах соответственно. Эти стандартные соединения удовлетворяют Правилу 5.26. Затем окружите законченную диаграмму ВПП структурой Case и соедините кластер **error in** с терминалом селектора. По умолчанию код ВПП будет содержаться в случае **No Error**. Поместите терминал селектора внизу структуры Case, теперь следует выровнять терминалы **error in**, селектора и **error out** по вертикали. Проведите кластер ошибок от терминала селектора через все узлы, которые имеют терминал ошибок, внутри структуры Case, затем через туннель на правой границе структуры – к терминалу индикатора **error out**. Внутри случая **Error** структуры Case передайте кластер ошибок от терминала селектора напрямую к выходному туннелю на правой границе. В остальном случае **Error** пуст.

В случае **No Error** вы также можете изменять информацию об ошибках путем введения кодов описания ошибок, определяемых пользователем. Источник ошибок может быть установлен считыванием свойства **VI Name** или считыванием и индексацией цепочки вызовов, возвращаемой функцией Call Chain. Эта функция возвращает всю иерархию вызовов, что помогает указать конкретный ВПП, который вызывает ошибку. Введение структуры Case гарантирует, что весь код ВПП будет пропущен, если возникнет ошибка. Таким образом, вы можете увеличить производительность и отзывчивость ваших ВПП и приложений, которые их вызывают.

Рисунок 7.11 иллюстрирует обработку ошибок в ВПП, встроенном в DAQmx Read. Панель на рис. 7.11а содержит стандартные кластеры, присвоенные нижним левым и правым соединительным терминалам. На рис. 7.11б показана блок-диаграмма, включающая случаи **No Error** и **Error** структуры Error Case Structure. Как видно, случай **No Error** в основном состоит из узла вызова динамической биб-

лиотеки, который производит операцию считывания и вызова утилиты ВП DAQmx Fill In Error Info. Случай **Error** – пустой, за исключением передачи сквозь него кластера ошибок и константы. На рис. 7.11в показаны составляющие ВП DAQmx Fill In Error Info. Эта утилита проверяет, не произошла ли ошибка, и создает информацию об источнике ошибок, используя комбинацию данных, возвращаемых из Call Chain и любых сообщений, переданных на **extended error info**. Поскольку ошибка произошла при вызове ВПП, цепочка вызовов индексируется одним элементом.



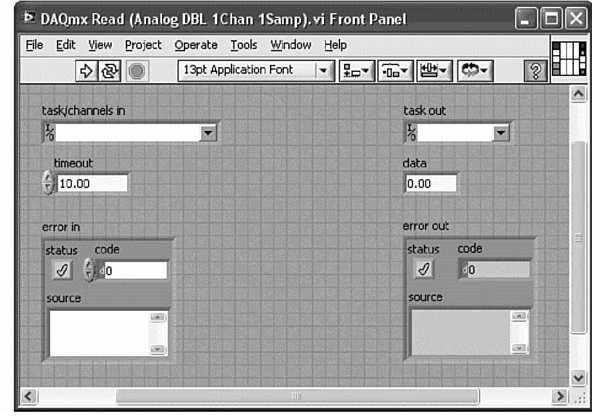


Рис. 7.11а. Лицевая панель ВП DAQmx Read состоит из стандартных кластеров ошибок, связанных с нижними левым и правым соединительными терминалами

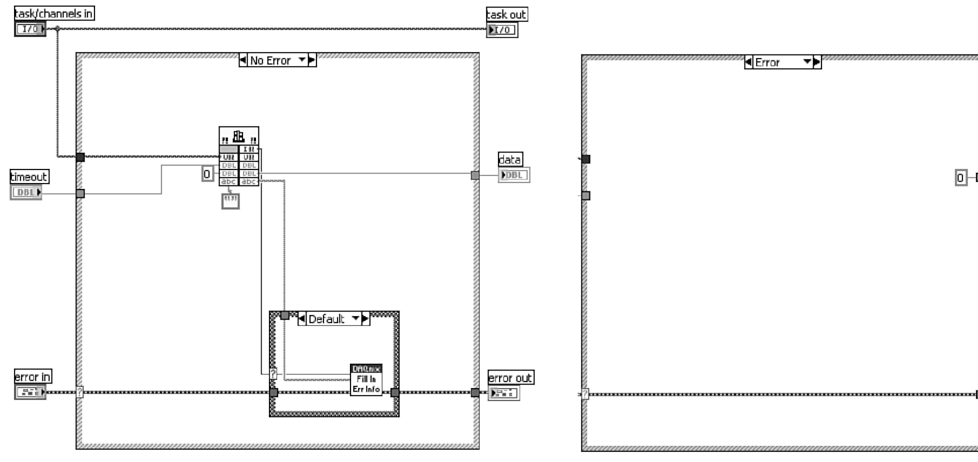


Рис. 7.11б. Блок-диаграмма ВП DAQmx Read использует структуру Error Case Structure для того, чтобы пропустить код, если произошла ошибка

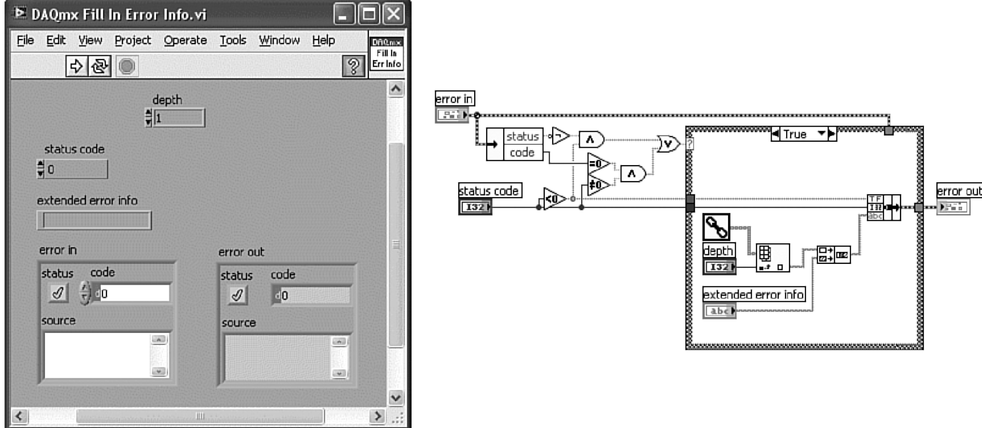


Рис. 7.11в. ВП DAQmx Fill In Error Info создает информацию об источнике ошибок, используя комбинацию данных, возвращаемых Call Chain, и любых сообщений, переданных на **extended error info**

Правило 7.14 Используйте значения по умолчанию для несоединенных туннелей в случае Error

В качестве замечания: удобство обслуживания ВП DAQmx Read может быть улучшено путем использования значений по умолчанию для несоединенных туннелей в случае **Error**, а именно – численная константа, соединенная с выходным туннелем, ведущим к терминалу **data** индикатора, должна быть удалена и заменена на значение по умолчанию. Это можно сделать через меню быстрого доступа туннеля, выбрав **Use Default If Unwired** (Использовать значение по умолчанию, если нет соединения). Такой способ дает несколько преимуществ по сравнению с использованием констант. Во-первых, индикаторы могут быть изменены или удалены без необходимости вручную исправлять проводники данных и константы. Во-вторых, легче убедиться, что значения по умолчанию используются простой проверкой. И последнее, блок-диаграмма выглядит чище без констант. Подсоединяйте константы к выходным туннелям, только если это значение не совпадает с значением по умолчанию для данного типа данных.

До версии LabVIEW 8.2 узел обращения к динамической библиотеке Call Library Node не имел терминалов ошибок. Соответственно, использование структуры Case было единственным методом предотвратить выполнение вызова функции динамической ссылки на библиотеку (*.dll). Структуры Error Case также используются для увеличения производительности внутри ВПП, содержащих функции, которые передают кластер ошибок. Если ваш ВПП является, например, драйвером устройства, передающим кластер ошибок через функции VISA, то он гарантирует, что ошибка на раннем звене цепочки ошибок предотвратит выполнение функций VISA на более поздних этапах. Однако большинство драйверов

устройств осуществляют серьезные операции в командной строке и поддерживают синтаксический анализ (parsing), помимо взаимодействия с устройством.

На рис. 7.12а ВПП-драйвер для цифрового мультиметра принимает команды, используя функции манипуляции строками, и посылает команды на устройство, используя VISA Write. Этот ВПП содержит незавершенную процедуру отслеживания ошибок. Кластер ошибок передается только на функцию VISA Write. Терминалы ошибок трех функций Format Into String (Форматирование в строку) не подключены. Соответственно, эти функции выполняются вне зависимости от статуса кластера **error in**. На рис. 7.12б в структуре Error Case Structure содержится весь код ВПП и кластер ошибок передается на все узлы, обладающие терминалами ошибок. Кроме того, функции, у которых нет терминала ошибок, такие как In

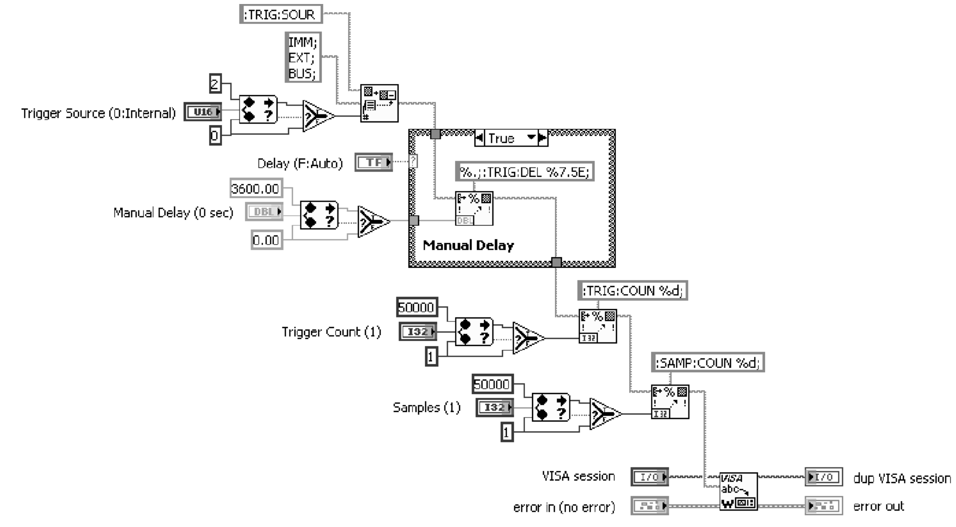


Рис. 7.12а. Драйвер устройства содержит незавершенную процедуру отслеживания ошибок

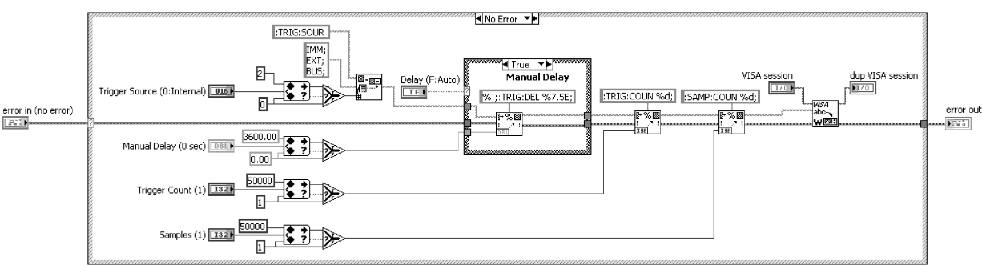


Рис. 7.12б. Кластер ошибок разделен между всеми функциями, имеющими терминалы ошибок, и структура Error Case увеличивает производительность в случае появления ошибки на предыдущих стадиях

Range and Coerce (В диапазоне? и округлить), Select (Выбор) и Pick Line (Взять строку), теперь также находятся в структуре варианта и не будут выполняться при наличии ошибок. Из-за отсутствия терминалов ошибок структура Case – единственный способ, предотвращающий выполнение этих функций. Таким образом, структура Error Case увеличивает производительность в случае появления ошибки. Кроме того, обратите внимание, что элементы на блок-диаграмме были выровнены по горизонтали по оси, определяемой терминалами ошибок.

Функции и ВПП в LabVIEW, не пропускающие выполнение кода при наличии ошибок, включают в себя большинство узлов, которые прерывают сессию с источником. К ним относятся Close Reference, VISA Close, Close File, и несколько других узлов в именах которых содержатся слова: Destroy, Stop, Close, and Clear. ВПП, вызывающие эти узлы, не должны пропускать выполнение кода при ошибке. Вместо этого следует допустить выполнение этих элементов, тщательно изучив описание терминала **error in** или блок-диаграмму этих узлов. Если в описании сказано, что узел работает нормально при наличии ошибки до него, то такой узел следует поместить вне структуры Error Case. Например, все Plug and Play-драйверы устройств в LabVIEW включают ВП Close, который вызывает функцию VISA Close. Эти ВПП не содержат структуры Error Case.

Правило 7.15 Используйте ВПП с шаблоном обработки ошибок

Если вы любите короткие пути, то начните создание вашего ВПП с шаблона subVI with Error Handling (ВПП с обработкой ошибок), доступного в меню **New ⇒ VI ⇒ From Template ⇒ Frameworks ⇒ SubVI with Error Handling**. Этот шаблон загрузит пустой ВП с механизмом обработки ошибок, описанным в этом разделе. Это шаблон позволит сэкономить пару минут при создании каждого ВПП и гарантирует качество во всем вашем приложении.

7.3. Определение приоритетов ошибок

Согласно правилу 7.5, лучше всего отслеживать все ошибки со всех узлов, обладающих терминалами ошибок. Это поможет увеличить надежность приложения. На практике, однако, я знаю очень мало дотошных разработчиков, которые соединяли бы все терминалы ошибок на каждом узле на каждой блок-диаграмме. На самом деле существует вероятность конфликта между Правилом 7.5 и некоторыми правилами, касающимися проводников данных, рассмотренных в разделе 4.2. На плотной блок-диаграмме, которая содержит параллельные потоки данных, соединение всех терминалов ошибок может привести к наложению проводников, невероятно большим блок-диаграммам и наличию потока данных «справа налево». Поиск решения, учитывающего все эти соображения, может потребовать времени. Какие же компромиссы между затраченным временем, надежностью и хорошим стилем допустимы? Для ответа на этот вопрос оцените ваш уровень комфорта и понимания операций, совершаемых каждым узлом, в рамках вашего приложения и склонности к риску.

Если вы, как и я, приучились использовать видимые (**Visible**) свойства на лицевой панели элемента управления еще с версии LabVIEW 3.0, которые раньше содержались в узле Attribute Node и который не имел терминалов ошибок, то вы можете решиться на пропуск отслеживания ошибок в этом узле. Однако с тех пор произошли серьезные изменения в поведении различных функций и свойств от версии к версии. Например, функции синхронизации и коммуникации, похоже, приобретают все больше улучшений с каждым новым релизом. Обработка ошибок помогает нам узнать и изучить разницу между версиями LabVIEW и сделать наши приложения более надежными.

Функции и ВП в LabVIEW можно классифицировать в терминах риска, чтобы лучше понять типы создаваемых ими ошибок и наметить стратегию обработки ошибок для данного приложения. Вероятность и последствия ошибки зависят от приложения и системы. Однако различные типы операций имеют различные уровни риска возникновения ошибок вне зависимости от приложения или платформы. Мне показалось полезным разделить узлы на палитрах по трем уровням риска: высокий, средний и низкий. Затем расставить приоритеты отслеживания ошибок, опираясь на допустимые риски.

Терминалы **error out** в LabVIEW – основной метод, с помощью которого ошибки в LabVIEW возвращаются из узлов, в которых возникают. Исключение составляют узлы, которые возвращают коды ошибок как скалярное целое число. К ним относятся ВП математических функций и многие узлы вызова библиотек, разработанные до версии LabVIEW 8.2. Наличие или отсутствие терминалов **error out** и **error code** на соединительной панели узла – один из индикаторов риска, связанного с этим узлом. Базовые математические функции, доступные на палитре **Numeric**, например, не имеют терминалов ошибок и не возвращают ошибки. Это связано с тем, что LabVIEW обрабатывает все возможные комбинации проблем, которые могут возникнуть, и неправильное поведение этих функций практически невозможно. Например, деление на ноль возвращает значение **Inf**, которое является допустимой константой, понимаемой всеми численными функциями, элементами управления и индикаторами в LabVIEW. Поэтому простые математические функции не требуют обработки ошибок. Так же, как и все функции на остальных палитрах, которые не имеют терминалов ошибок, не создают ошибок. К ним, в частности, относятся функции с палитры **Structures**, **Array**, **Cluster &Variant**, **Boolean**, **String**, **Comparison**, **Timing** и некоторые другие.

На другом конце спектра рисков находятся функции и ВП, которые осуществляют операции ввода/вывода (I/O). Эти узлы осуществляют вызов драйверов устройств, DLL-библиотек, обращаются к операционной системе или любому приложению или источнику, внешнему по отношению к среде LabVIEW, включая удаленные копии LabVIEW. Операции ввода/вывода включают в себя все узлы находящиеся на палитрах: **File I/O**, **Measurements I/O**, **Instrument I/O** и **Data Communication**. Эти узлы опасны, потому что они опираются на внешние драйверы или источники, которые могут находиться в состоянии (а могут и не находиться) ответить правильно на запрос приложения. Внешние источники осуществляют операции ввода/вывода независимо от LabVIEW, и существует много сценариев

нарушения операций. К тому же нарушение операции ввода/вывода, скорее всего, приведет к таким нарушениям в работе приложения, вызвавшего операцию, как зависание в ожидании ответа устройства. Большинство функций ввода/вывода и ВП в LabVIEW возвращают кластер ошибок. Отслеживание и обработка возвращаемой ими информации абсолютно необходима.

Узлы среднего уровня риска имеют терминалы ошибок, но не вызывают внешние по отношению к среде LabVIEW ресурсы. К ним относятся узлы свойств (Property Nodes), связанные с элементами на лицевой панели, все функции палитры синхронизации, ВП Server, когда контролирует локальную копию LabVIEW, большинство Express ВП (за исключением ВП на субпалитрах **Express Input** и **Output**), Scan from String, Format into String, ВП Mathematics. Большинство ошибок, возникающих в узлах среднего уровня риска, не вызовут больших задержек во времени или сбоя системы. Обработка ошибок – лучший способ определить то, как эти функции применялись, ее настоятельно рекомендуется проводить. Например, подача ошибки, возвращаемой с узла свойств, помогает определить что программное значение свойства имеет недопустимое значение, что в другом случае могло остаться незамеченным.

На рис. 7.13 представлены примеры кода функций из всех трех категорий риска. Рисунок 7.13а содержит математические операции с использованием численных функций, которые не имеют терминалов ошибок. Ни одна из функций на палитре **Numeric** не имеет терминала ошибок, и нет риска их возникновения при использовании этих функций. На рис. 7.13б находится узел свойств (Property Node) и функция Scan from String с подключенными терминалами ошибок. Эти функции обладают средним уровнем риска, рекомендована обработка ошибок. На рис. 7.13в находятся функции VISA Write и Read, которые являются примером функций ввода/вывода с высоким уровнем риска. Обработка ошибок – ключевой момент для операций ввода/вывода, включая взаимодействие с устройствами по средствам VISA.

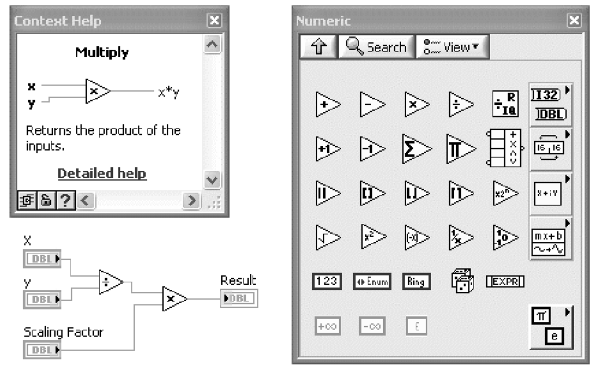
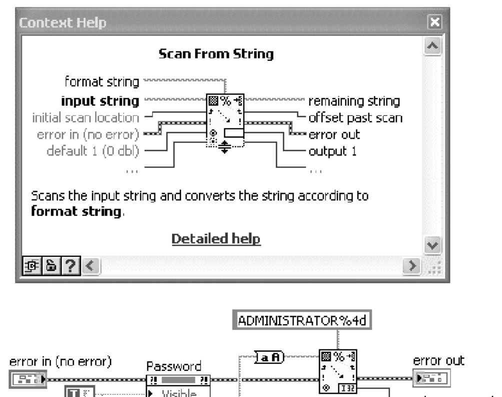


Рис. 7.13а. Функции с палитры ***Numeric****, которые не имеют терминалов ошибок, не могут их генерировать. Эти функции относятся к категории низкого риска*



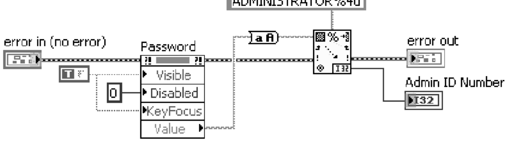


Рис. 7.13б. Узлы свойств, связанные с элементами управления на лицевой панели, и функция Scan from String являются примерами узлов со средним уровнем риска. Ошибки обычно не критичные, но отслеживать их настоятельно рекомендуется

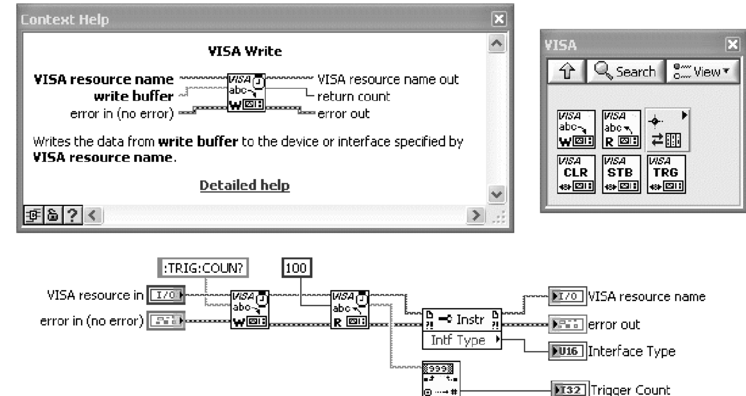


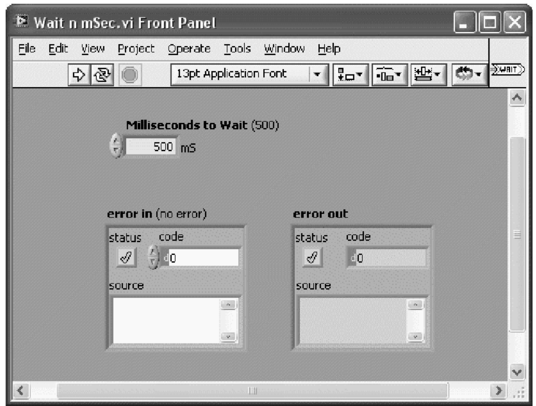
Рис. 7.13в. Функции, которые обращаются к драйверам внешних устройств, такие как VISA, обладают высоким риском возникновения ошибок, и соответствующие ошибки зачастую определяются вызывающим приложением. Отслеживание ошибок необходимо для функций ввода/вывода

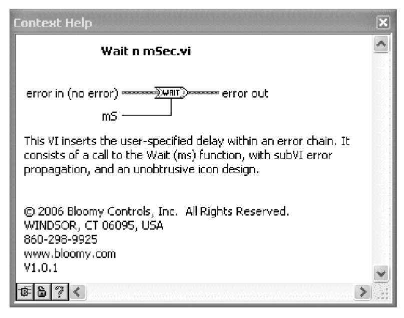
Правило 7.16 *Отслеживание ошибок* ***требуется*** *для узлов, осуществляющих операции ввода/вывода,* ***рекомендуется*** *для узлов, обладающих терминалами ошибок, и* ***является опциональным*** *для блок-диаграмм, не содержащих узлов с терминалами ошибок*

Отслеживайте все ошибки от всех узлов с терминалами ошибок для максимально возможной надежности. Однако, когда возникают конфликты стиля, следует расставить приоритеты ошибок согласно трем представленным классам узлов. Для успешной обработки любых ошибок, создаваемых узлами ввода/вывода, требуется минимум усилий. Как уже обсуждалось, функции ввода/вывода и ВПП имеют высокую вероятность возникновения ошибок и наиболее тяжелые последствия, если ошибки случаются. Кроме того, крайне рекомендуется отслеживать ошибки для функций среднего уровня риска, которые имеют терминалы ошибок, но не осуществляют операции ввода/вывода. Ошибки, возвращаемые этими функциями, помогают разработчику понять, как они работают и как улучшить надежность приложения. Наконец, когда вы создаете процедуры, содержащие узлы без терминалов ошибок, то передача ошибок через структуру Error Case или ВПП с терминалами ошибок является опциональной. В последнем случае передача кластера ошибок через ВПП полезна с точки зрения организации потока данных, поддерживается стандартными схемами терминалов соединительной панели и оптимизирует создание отчетов об ошибках и восстановление.

На рис. 7.14а содержатся лицевая панель, блок-диаграмма и окно контекстной справки для ВП Wait n mSec. Этот ВПП просто вставляет задержку в цепочку ошибок. Уникальная форма иконки и соединительная панель обсуждались в главе 5 «Иконки и Терминалы соединительной панели». Диаграмма содержит функцию Wait (ms) и соответствующий ВПП обработки ошибок. Обратите внимание, что функция Wait не имеет терминалов ошибок и не может их создавать. Это пример функции с низким уровнем риска. Однако этот ВПП поддерживает передачу ошибок и структуру Error Case для пропуска кода в случае ошибки согласно правилам раздела 7.2. Более того, терминалы ошибок этого ВПП позволяют вызывающим ВП использовать зависимость данных для определения момента задержки и позволяют проскочить задержку, если ошибка уже появилась.

На рис. 7.14б ВП Wait n mSec используется в примере взаимодействия устройств, а именно – ВП Wait n mSec расположен между функцией VISA Write и Property Node, который возвращает число байт в последовательном порту. Введение задержки дает время устройству ответить на запрос, прежде чем ответ будет считан. Обратите внимание, что наличие любых ошибок до ВП Wait n mSec повлечет отсутствие задержки. Поэтому обработка ошибок этого ВПП упрощает порядок выполнения посредством зависимости данных и оптимизирует производительность в случае ошибки.





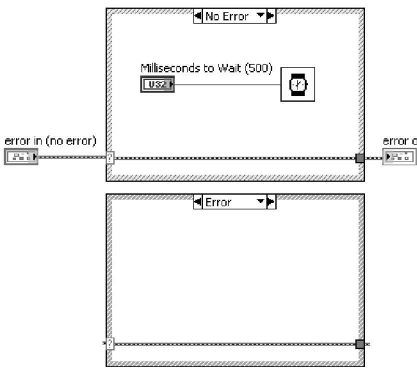


Рис. 7.14а. ВПWait n mSec состоит из функции Wait (ms) с низким уровнем риска и без терминалов ошибок и соответствующего ВПП обработки ошибок, включающего структуру Error Case

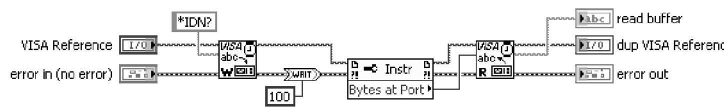


Рис. 7.14б. ВП Wait n mSec вставляет задержку в 100ms между функциями VISA Write и Property Node, что дает устройству время ответить на запрос

7.4. Советы по обработке ошибок

В этом разделе содержатся советы по обработке ошибок, включая соединение структур, слияние ошибок, очищение ошибок и автоматическую обработку ошибок.

7.4.1. Соединение структур

Правило 7.17 Туннели для кластера ошибок следует размещать внизу структуры

Провести кластер ошибок через такие структуры как циклы, События (Event), Случаи (Case), Последовательность (Sequence), можно через вертикально упорядоченные входные и выходные туннели на правой и левой границах структуры. Каждая структура должна иметь только одну пару входных и выходных терминалов ошибок, и они должны быть расположены ближе всех к нижней границе структуры. Все примеры в этой главе поддерживают это соглашение. Однако не следует соединять кластеры ошибок через структуру многих кадров, если только она не используется внутри другой структуры.

7.4.2. Слияние ошибок

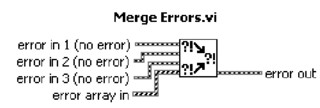
ВП Merge Errors была представлена в разделе 7.1.1 «Отслеживание ошибок» как способ соединения многих кластеров ошибок в один. Как показано в окне контекстной справки на рис. 7.15а, этот ВП имеет три терминала **error in**, один терминал **error in array** и один терминал **error out**. Этот ВП просматривает ошибки на входе сверху вниз и возвращает первую, которую найдет. Если ошибок нет, то он возвращает первое предупреждение, если нет и предупреждения, то говорит об отсутствии ошибок. Эта техника используется для комбинирования множества цепочек ошибок, как на рис. 7.3, или конвертирования массива индексированных циклом ошибок в скаляр, как на рис. 7.5. Объединенный выходной кластер ошибок упрощает распространение кластера ошибок, так как теперь требуется лишь один терминал **error out** или один ВП, создающий отчет об ошибке.

ВП Merge Errors нарушает Правило 5.25 «Связывайте кластеры ошибок с нижним левым и правым терминалами», так как кластер **error in**, который обычно связан с нижним левым терминалом, заменен массивом **error in array**. Это приводит к неувязкам между иконкой и терминалами, когда ВП Merge Errors используется со скалярным цепочками ошибок, как показано на рис. 7.3. Точно так же неувязки происходят с проиндексированными циклами массивов ошибок с параллельным номерами элементов, как на рис. 7.5а. Более того, терминал **error in array** имеет ограниченное значение. Массивы кластеров ошибок формируются индексированием ошибок на выходных туннелях циклов или созданием массива, объединяющего несколько кластеров ошибок. Как уже обсуждалось в разделе 7.1.1 «Отслеживание ошибок», индексирование ошибок нежелательно. Вместо этого

большинство циклов проверяют статус ошибок и прекращают работу при наличии таковых, или передача кластера ошибок осуществляется с использованием сдвигового регистра. Таким же образом операция слияния в ВП Merge Errors / устраняет необходимость формирования массива, если он существует.

На рис. 7.15б представлен ВП Merge Multiple Errors, альтернатива ВП Merge Errors, содержащий пять дополнительных терминалов **error in** и не содержащий терминала **error in array**. Он использует стандартную схему соединительной панели 4×2×2×4, где все средние и левые терминалы связаны с кластерами **error in**. Таким образом, этот ВП может объединить до восьми скалярных кластеров **error in** в один кластер **error out**. Этот ВП увеличивает число цепочек ошибок, которые можно объединить, обеспечивает хорошую обработку ошибок, удобство соединения и стиль.

На рис. 7.15а пять параллельных цепочек ошибок соединены с использованием ВП Merge Errors. Функция Build Array необходима для соединения двух кластеров ошибок в массив, и ВПП располагается примерно по середине между набором параллельных цепочек ошибок. На рис. 7.15б используется ВП Merge Multiple Errors для объединения всех цепочек сразу. Поскольку левый нижний терминал удовлетворяет Правилу 5.26, этот ВПП упорядочен по вертикали с терминалами ошибок и другими ВПП нижней цепочки ошибок. Таким же образом устранена функция Build Array, и блок-диаграмма теперь содержит меньше проводников данных. Наконец, средний соединительный терминал используется для объединения пяти цепочек ошибок в этом примере.



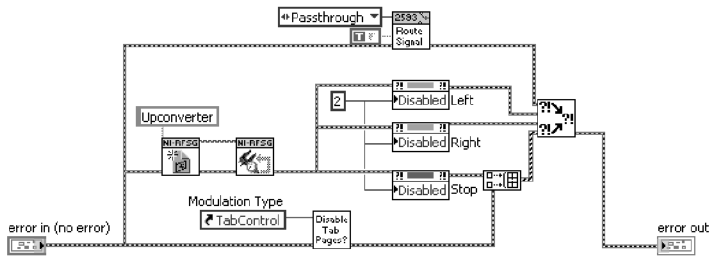
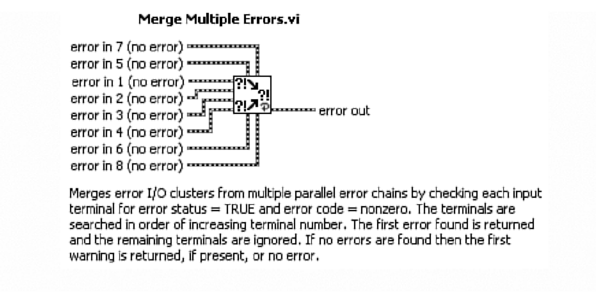


*Рис. 7.15а. Пять параллельных цепочек ошибок объединены с использованием ВП Merge Errors. Из-за нестандартного терминала **error in array** в нижнем левом углу, ВПП нельзя выровнять ни с какими другими ВПП. Также необходимо использовать функцию Build Array*



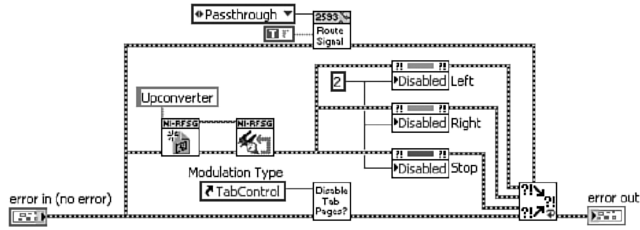


Рис. 7.15б. ВПMerge Multiple Errors
объединяет до восьми скалярных кластеров **error in**,
используя стандартную соединительную панель 4×2×2×4.
Этот ВПП упорядочен по вертикали вместе с терминалами ошибок других ВПП

7.4.3. Очищение ошибок

Обработка ошибок разделяет два класса нарушения работы: ошибки и предупреждения. Ошибки однозначно отличаются значением TRUE логической переменной статуса кластера ошибок. Предупреждения соответствуют любым не нулевым кодам ошибок вместе со статусом FALSE. Как уже говорилось, когда обнаруживается ошибка, то большинство узлов, расположенных дальше в потоке данных, будут пропускать выполнение кода. Исключениями являются большинство функций, закрывающих ссылки или сессии, такие как File Close, VISA Close, Close Reference и Release Queue. В некоторых ситуациях желательно игнорировать определенные ошибки, так чтобы последующие узлы выполнялись обычным образом.

Например, представьте себе общение приложения-сервера с клиентской частью, когда операции соединения и разъединения – обычное дело. Когда клиент отсоединяется, функции коммуникации возвращают ошибку. Приложению-серверу необходима эта информация, чтобы очистить соединение с клиентом и начать поиск новых соединений. Однако если ошибка разъединения распространяется через приложение-сервер, то функции коммуникации не будут выполнены, и попытки соединения с новым клиентом будут безуспешны. Вместо этого сервер должен обработать и очистить ошибки, связанные с отключением клиента.

На рис. 7.16 представлен ВПП, называющийся ВП Clear All Error or Specified (1). Его назначение – программно сбрасывать кластер ошибок на состояние без

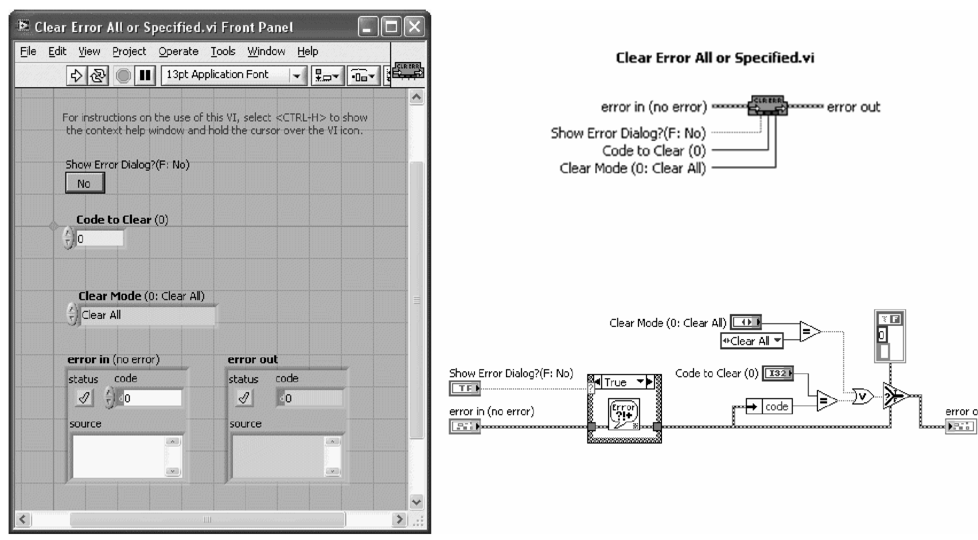


Рис. 7.16. ВП Clear All Error or Specified – это утилита,
которая очищает ошибки согласно значениям Clear Mode и Code to Clear

ошибок. В зависимости от значения **Clear Mod** этот ВП обнуляет код ошибок всегда или только тогда, когда код ошибки совпадает с заранее заданным **Code to Clear**. Также этот ВП может показывать диалоговое окно перед сбросом ошибки. Обратите внимание на малый размер иконки. Он позволяет другим проводникам данных на блок-диаграмме вызывающего приложения проходить над этим ВПП, иконка не блокирует им путь. Иконка и соединительная панель Clear All Error or Specified ВП были рассмотрены в главе 5.

Рисунок 7.17 содержит иллюстрацию применения ВП Clear All Error or Specified в ВП Torque Hysteresis, которая обсуждалась в главе 6 «Структуры данных». После каждого теста ВП Save Data спрашивает оператора об имени файла, в который она запишет полученные данные. Однако если пользователь отменит

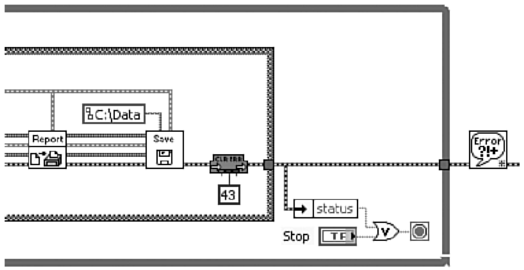


Рис. 7.17. ВП Clear All Error or Specified удаляет ошибку с кодом 43
в ВП Torque Hysteresis. Это обычно происходит, когда пользователь
нажимает кнопку **Cancel** в диалоговом окне

запись в файл в диалоговом окне, то ВП возвращает ошибку с кодом 43. Эта ошибка приводит к прекращению работы цикла до тех пор, пока информация об ошибке не будет удалена из кластера ошибок. ВП Clear All Error or Specified используется для удаления ошибки с кодом 43, и приложение продолжает работать.

7.4.4. Автоматическая обработка ошибок

Автоматическая обработка ошибок – это еще один прозрачный способ отслеживать ошибки на блок-диаграмме. Он состоит из свойства ВП под названием **Enable automatic error handling**, находящегося в категории **Execution** меню **VI Properties**. Когда в каком-либо узле с несоединенным терминалом **error out** случается ошибка, то LabVIEW приостанавливает выполнение приложения, открывается окно блок-диаграммы, узел, в котором возникла ошибка, подсвечивается, и возникает диалоговое окно ошибки. Если терминал **error out** узла соединен, то данные об ошибке распространяются, как запрограммировано, и никаких дополнительных действий не происходит. Автоматическая обработка ошибок по умолчанию отключена во всех версиях LabVIEW, кроме 7.0; в этой версии данная функция изначально и появилась.

Важно отметить, что автоматическая обработка ошибок не гарантирует того, что об ошибках будет доложено. Если терминал **error out** узла соединен, но передача кластера ошибок прерывается без процедуры создания отчета, то данные об ошибке будут потеряны. Так что автоматическая обработка ошибок помогает, только когда один или более терминалов **error out** узлов не соединены, например, в случае, когда расставлены приоритеты узлов, как обсуждалось в предыдущем разделе. Однако большинство проблем с обработкой ошибок, которые я видел на практике, являются результатом неполной процедуры обработки ошибок, когда множество терминалов ошибок соединены, но ошибки полностью не отслеживаются. Автоматическая обработка ошибок в такой ситуации не поможет.

Правило 7.18 Оставляйте автоматическую обработку ошибок выключенной

Если вы разрабатываете промышленное приложение, то последнее, что вам нужно, так это то, чтобы оно вдруг приостановило свою работу, открыло окно блок-диаграммы и высветило кусок исходного кода. Потому что это приглашение для любого, кто окажется возле компьютера, когда случится ошибка, поиграть с блок-диаграммой. Таким образом, автоматическая обработка ошибок должна быть отключена перед установкой приложения. Кроме того, автоматическая обработка ошибок обманывает программную. По теореме 7.1, программная обработка ошибок является ключевым моментом для преодоления непредвиденных ошибок до, во время и после установки приложения. Более того, программная обработка ошибок – это принципиальный метод установления потока данных и главная составляющая хорошего стиля. Поэтому используйте программную обработку ошибок, а автоматическую оставляйте выключенной.

7.5. Примеры

В этом разделе содержится еще больше примеров обработки ошибок в циклах, параллельных цепочках и ошибок, заранее определенных пользователем.

7.5.1. Постоянное получение данных и запись в файл

ВП Cont Acq&Graph Voltage-To File(Binary) поставляется вместе с LabVIEW в качестве примера, этот ВП постоянно получает данные и записывает их в файл, используя ВП DAQmx и функции ввода/вывода файлов. ВП DAQmx используют общее задание, а функции ввода/вывода используют общие номера ссылок и расположены параллельно для облегчения распространения данных между элементами. Поскольку узлы независимы, то общая цепочка ошибок делится между ВП для обеспечения зависимости данных и порядка выполнения. Цикл прерывается, если в каком-то узле происходит ошибка, и сообщение об ошибке отображается в диалоговом окне. Реализация показана на рис. 7.18а и может быть найдена через NI Example Finder: **Hardware Input and Output ⇒ DAQmx ⇒ Analog Measurements ⇒ Voltage ⇒ Cont Acq&Graph Voltage-To File(Binary) VI**.

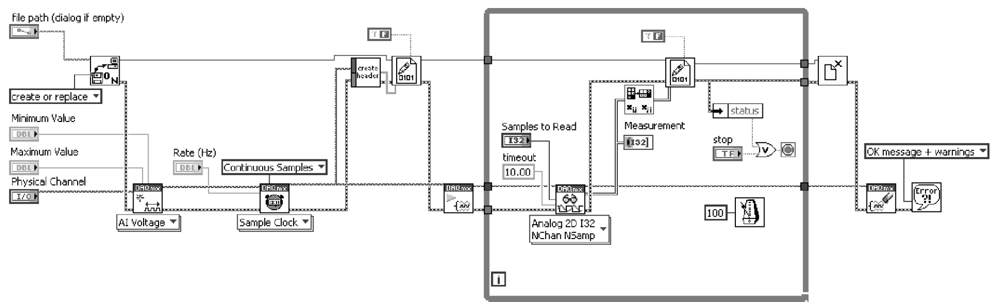


Рис. 7.18а. Поставляемое в качестве примера приложение постоянно собирает данные и записывает их в файл. Функции ввода/вывода и ВП DAQmx используют общий кластер ошибок. Если в каком-либо узле происходит ошибка, то приложение прекращает работу

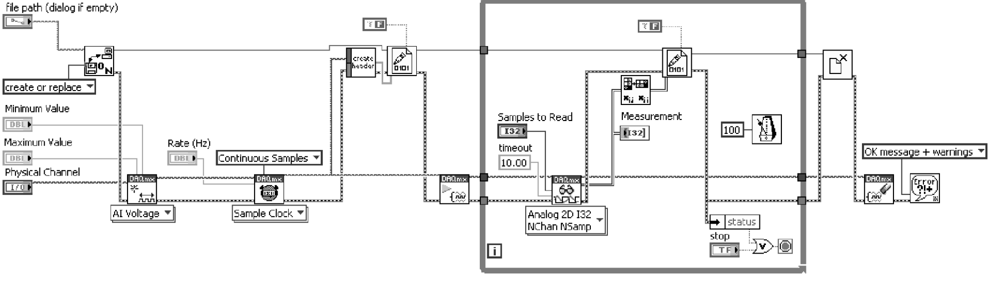


Рис. 7.18б. Некоторые незначительные улучшения поставляемого с LabVIEW примера

На рис. 7.18б содержится несколько небольших улучшений по сравнению с поставляемым примером. Во-первых, кластер ошибок входит в цикл While и выходит из него через упорядоченные по вертикали туннели, расположенные около нижнего края цикла. Кроме того, условие выхода из цикла теперь расположено в нижнем правом углу цикла, вместе с кодом, который разъединяет статус ошибки и оценивает состояние цикла. Наконец, вместо ВП Simple Error Handler используется ВП General Error Handler.

7.5.2. Suss Interface Toolkit

На рис. 7.19 содержится диаграмма драйвера устройства, который контролирует систему зондирования полупроводниковой платы. В диаграмму встроен ВПП обработки ошибок, содержащий терминалы кластеров ошибок и структуру Error Case. В случае **No Error** структуры Error Case выполняется узел вызова библиотеки и процедура ошибок.

Узел Call Library Function Node вызывает функцию, содержащуюся внутри библиотеки DLL. Функция возвращает параметр целочисленного типа с именем

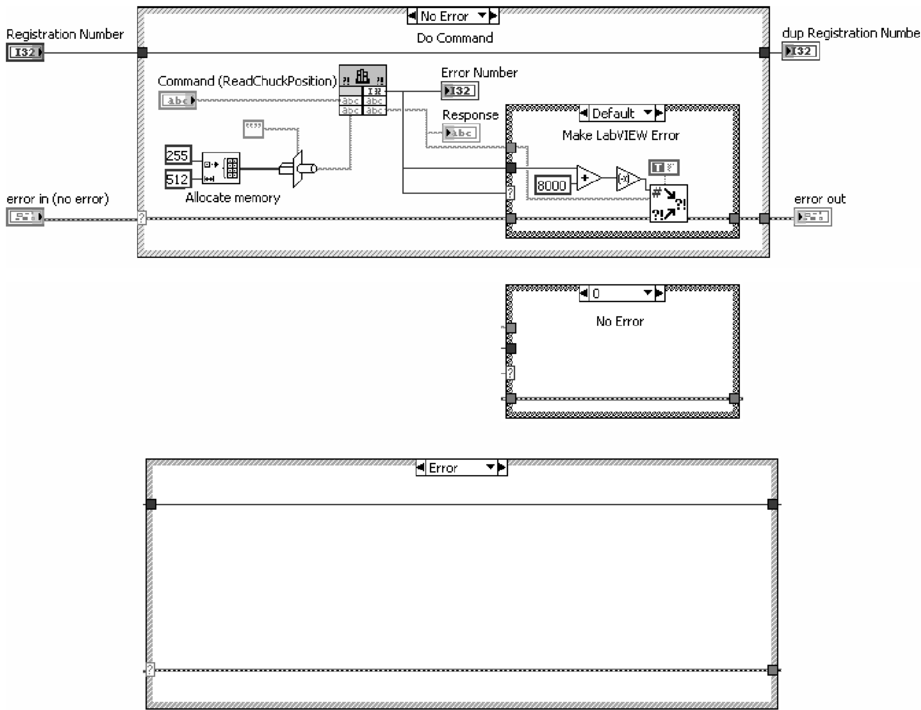


*Рис. 7.19. Драйвер устройства использует Call Library Function Node для вызова DLL. Если DLL возвращает ошибку, то **Error Number** сдвигается в диапазон ошибок, определенных пользователем*

Error Number. Этот параметр возвращает значение 0, если функция успешно выполняется, или целое число в диапазоне от 500 до 600, которое определяет ошибку, произошедшую в DLL. Процедура обработки ошибок оценивает это число – **Error Number** и создает кластер ошибок, используя кластер ошибок из ВП Error Code. Результирующий код ошибки в LabVIEW состоит из кода, полученного от исходного **Error Number**, получаемого из DLL, но сдвинутого в диапазон ошибок, заданных пользователем (от –8999 до –8000, от 5000 до 9999). Более точно – номер ошибки от 500 до 600 сдвигается в диапазон от –8500 до –8600. По Правилу 7.12 отрицательные коды предпочтительнее для ошибок устройств ввода/вывода. Также источник кластера ошибок заполнен интуитивно понятными сообщениями, состоящих из цепочки вызова и ответов из DLL.

7.5.3. Слияние параллельных ошибок

Рисунок 7.20 иллюстрирует процедуру инициализации, которая считывает параметры конфигурации из файла, используя ВП Config, устанавливает несколько свойств элементов управления через узел свойств (Property Node) и инициализирует значения всех элементов управления как значения по умолчанию, используя ВП Server. Эти узлы из трех параллельных цепочек ошибок перед выходом из структуры Case объединяются. ВП Merge Error имеет нестандартный нижний левый терминал, связанный с **error in array**. Это вызывает перегиб в проводнике данных кластера ошибок, как показано на рис. 7.20а. На рис. 7.20б Merge ВП Multiple Errors содержит только скалярные терминалы ошибок, включая нижний левый терминал, который, как обычно, связан с **error in**. Этот ВПП устраняет перегиб в проводнике данных.

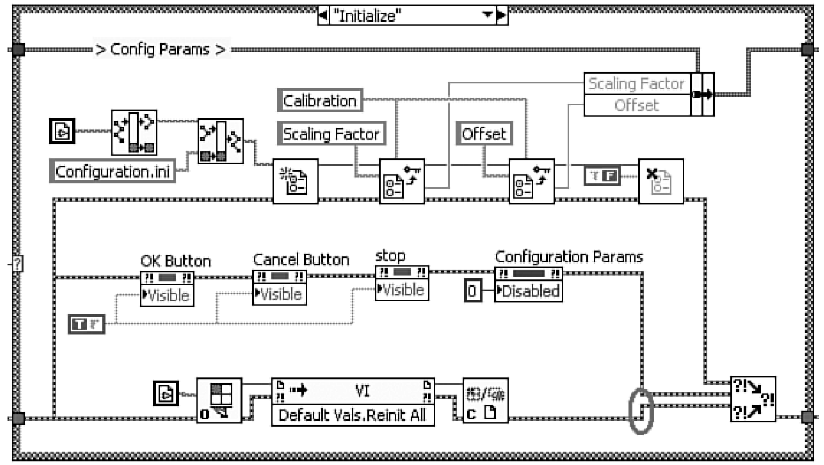


*Рис. 7.20а. ВП Merge Error имеет нестандартный нижний левый терминал, связанный с **error in array**. Это вызывает перегиб в проводнике данных кластера ошибок при объединении параллельных кластеров.*

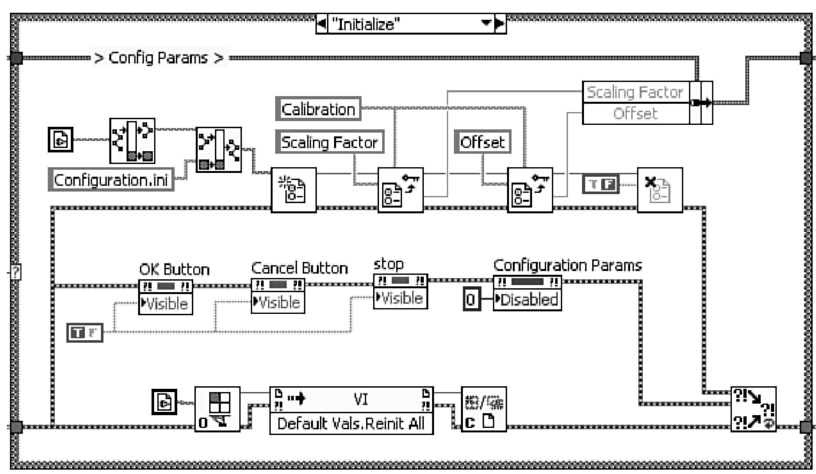


*Рис. 7.20б. ВП Merge Multiple Errors содержит восемь скалярных терминалов **error in**, что устраняет перегиб в проводнике данных*

7.5.4. ВП Screw Inspection

ВП Screw Inspection – это приложение, работающее с изображением, которое было ранее рассмотрено в главе 4. Часть блок-диаграммы на рис. 7.21а состоит из двух неполных параллельных цепочек ошибок внутри цикла For. Верхняя цепочка ошибок состоит из трех перекрывающихся ВП из палитры **Vision Utilities**. Непонятно почему, но самый правый ВП исключен из цепочки ошибок. Кроме того, кластер ошибок распространяется через ВП Find Screw Ends в нижней цепочке ошибок, но терминал **error in** первого ВП и **error out** последнего не соединены. Таким образом, в обеих цепочках ошибок ошибки не отслеживаются. Кроме того, нет зависимости данных между функцией Wait внутри структуры Case и остальными узлами, поэтому непонятно, когда появляется задержка.

На рис. 7.21б показано правильное отслеживание ошибок для параллельных цепочек ошибок и циклов For. Во-первых, кластер ошибок передается к терминалам **error in** всех узлов. Во-вторых, терминал **error out** последних узлов в каждой цепочке соединен с ВП Merge Multiple Errors. Передача данных об ошибках между итерациями цикла For осуществляется при помощи сдвиговых регистров. Наконец, функция Wait (ms) ВП заменена Wait n mSec. Терминалы ошибок этого ВП упрощают порядок выполнения через поток данных, а уникальный размер и форма иконки позволяют легко разместить ее на блок-диаграмме. Теперь задержка появляется строго в начале нижней цепочки ошибок, перед четырьмя ВП Find Screw Ends. Рисунок 7.21в похож на рис. 7.21б, за тем исключением, что цикл For заменен циклом While и некоторые узлы размещены иначе. Цикл While допускает прерывание на любой итерации в случае ошибки. Узлы теперь расположены так, чтобы кластер ошибок был самым нижним выходным туннелем. Для опреде-

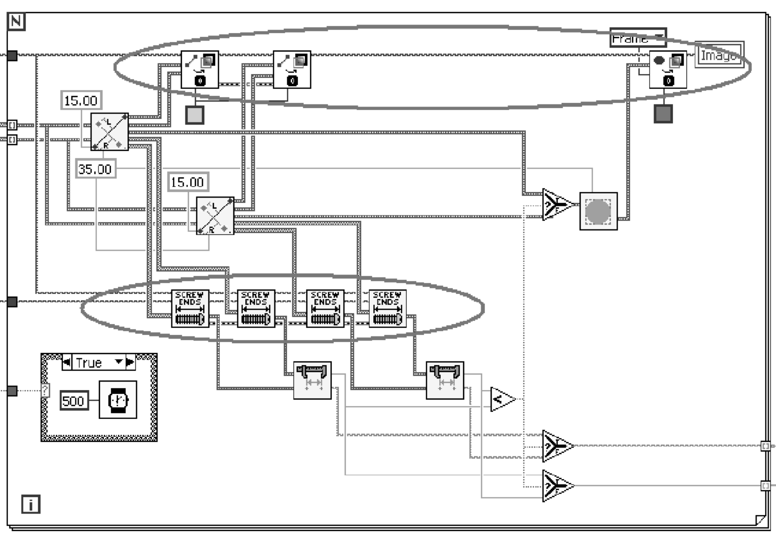


Рис. 7.21а. ВПScrew Inspection состоит из двух незаконченных параллельных цепочек ошибок внутри цикла For и функции Wait (ms) с неизвестным порядком выполнения

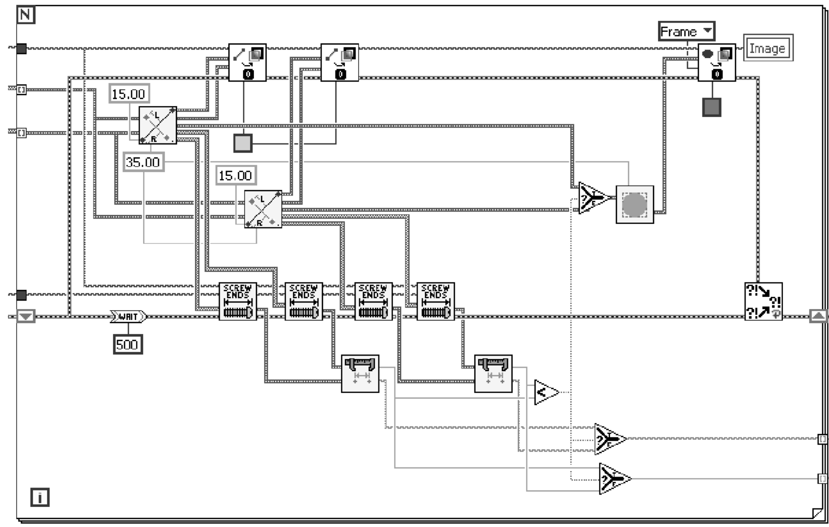


Рис. 7.21б. Отслеживание ошибок завершено путем подачи кластера ошибок на все терминалы, слияния параллельных цепочек ошибок и передачи данных между итерациями цикла с использованием сдвиговых регистров. ВП Wait n mSec использует поток данных, чтобы расположить задержку перед ВП Find Screw Ends

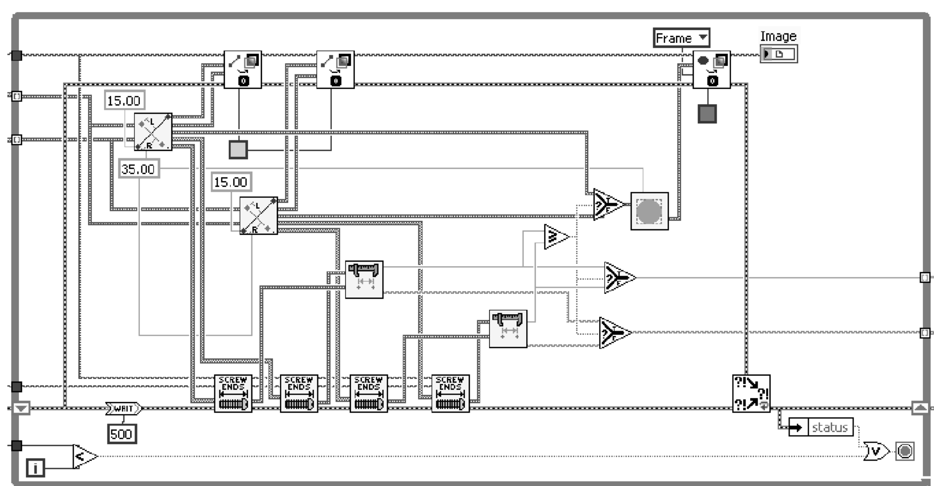


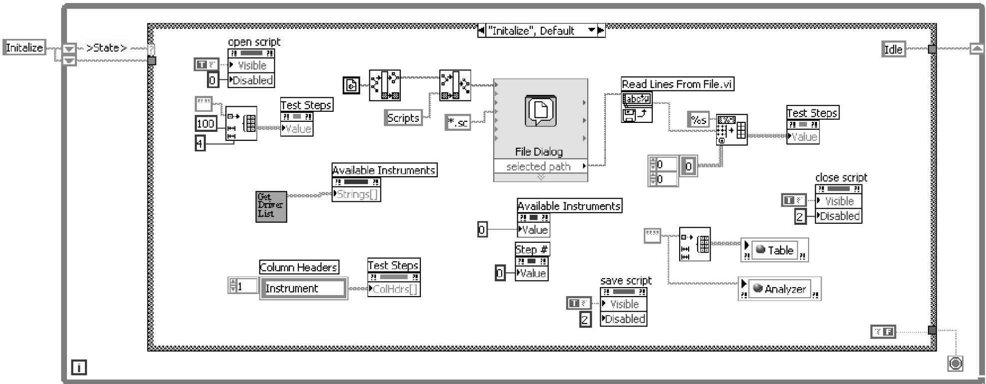
Рис. 7.21в. Используется цикл While, чтобы прервать цикл при первой ошибке. Узлы расположены так, чтобы устранить выходные туннели перед кластером ошибок

ления желаемого числа итераций требуется новый входной туннель. Это туннель расположен перед входным туннелем кластера ошибок для того, чтобы передать данные, которые определяют условие выхода из цикла, не пересекая другие проводники данных и объекты. Таким образом, Правилом 7.17 пожертвовали в пользу Правила 4.15.

7.5.5. ВП Test Executive

Приложение на рис. 7.22 является тестовым управляющим приложением, выполненным с использованием варианта шаблона конечного автомата (state machine), который более подробно рассматривается в главе 8 «Шаблоны». Показаны два случая основной структуры Case, **Initialize** и **Load Script**. В случае **Initialize**, терминалы ошибок не соединены на всех узлах, включая две переменные – девять переменных Property Nodes и ВП File Dialog Express. Также используются высокоуровневые ВП для считывания шагов тестирования из файла, имеющего встроенный обработчик ошибок, а именно – Read Lines из ВП File содержит вызов ВП General Error Handler с параметром **type of dialog**, равным **continue or stop message**, как показано на рис. 7.23. Если на этот ВПП приходит ошибка, то она создает диалоговое окно, что может быть нежелательным, если приложение должно работать автономно.

Случай **Load Script** содержит последовательность ВПП, которая взаимодействует с табличным приложением. Кластер ошибок распространяется среди всех ВПП, но ошибка не отслеживается, и о ней не докладывается. Терминал **error out** последнего ВПП в цепочке ошибок не соединен, что приводит к пропаже данных об ошибке. Точно так же и в оставшихся случаях процедура отслеживания оши-



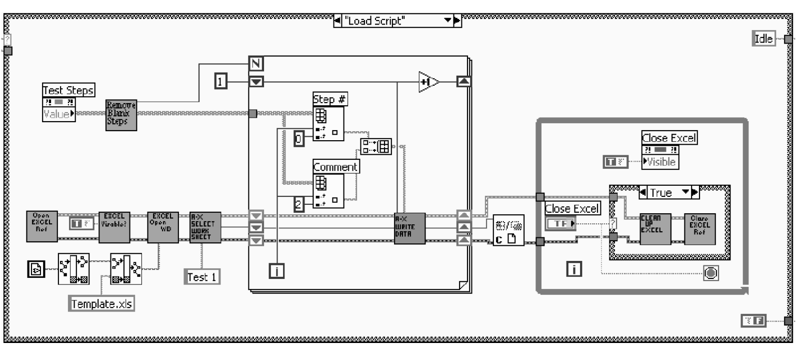


Рис. 7.22. Показаны случаи **Initialize** и **Load Script**. В случае **Initialize** терминалы ошибок многих узлов не соединены. В случае **Load Script** кластер ошибок распространяется между ВПП, но терминал **error out** последнего ВПП в цепочке ошибок не соединен, информация об ошибке пропадает

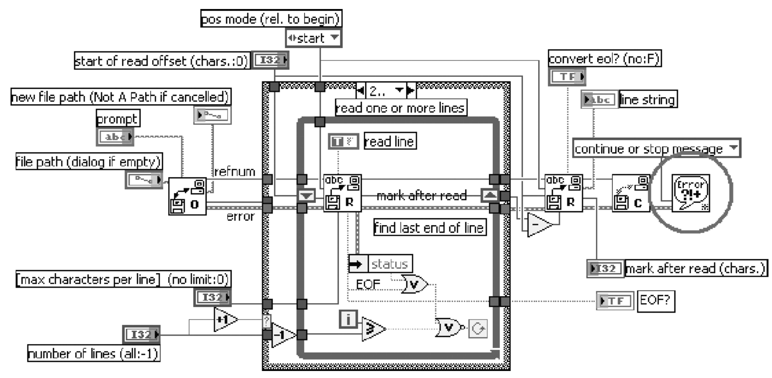


Рис. 7.23. Блок-диаграмма Read Lines из ВП File включает в себя обработку ошибок через ВП General Error Handling. Появление диалогового окна для автономного приложения нежелательно

бок не завершена. Это заметно с первого взгляда, так как кластер ошибок не входит и не выходит из структуры Case или цикла While. Незавершенная процедура отслеживания ошибок и отсутствие процедуры отчета об ошибках приводит к тому, что обработки ошибок фактически и нет.

На рис. 7.24 ВП Test Executive изменен так, чтобы обработка ошибок была правильной. Во-первых, кластер ошибок считывается с терминала **error in** и инициализует значение сдвигового регистра у нижней границы цикла While. Ошибка считывается из сдвигового регистра и передается через туннель в нижней части структуры Case. Каждый случай структуры Case, включая случаи **Initialize** и **Load**

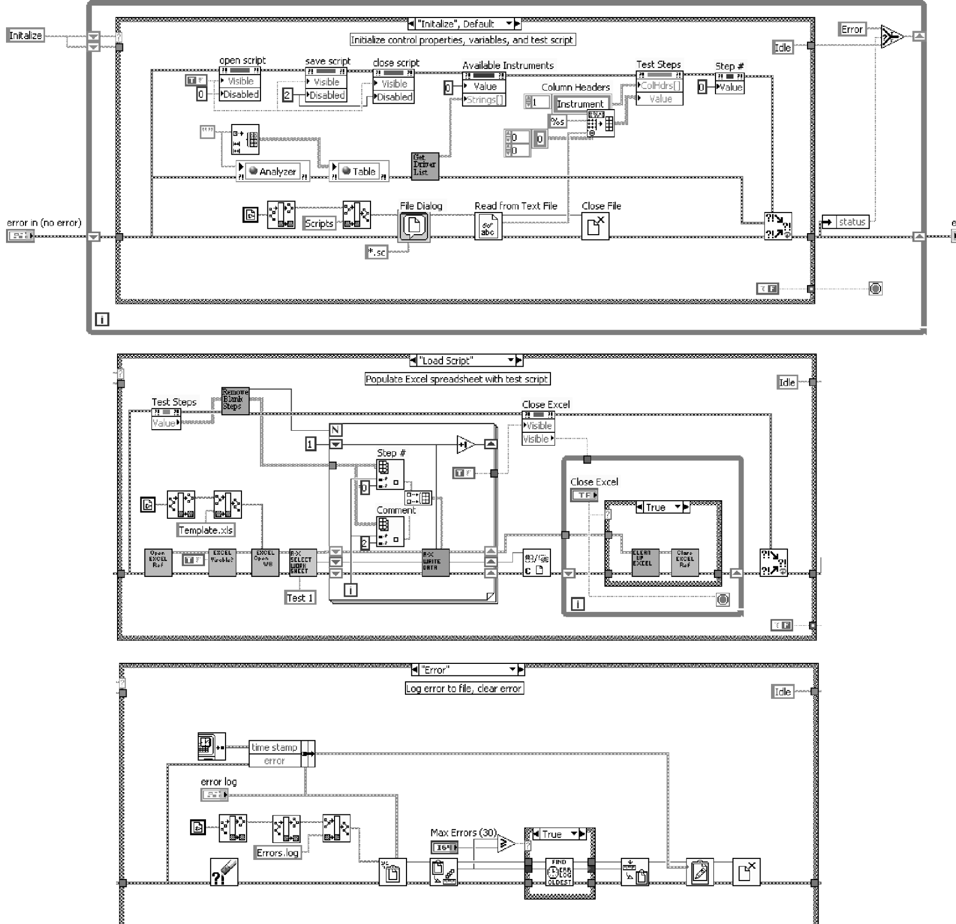


Рис. 7.24. ВПTest executive теперь использует правильную обработку ошибок. Ошибки отслеживаются через кластер ошибок, который распространяется через все узлы, имеющие терминалы ошибок. Информация об ошибках записывается в файл в случае ***Error***

Script, поддерживает распространение кластера ошибок между всеми узлами, у которых есть терминалы ошибок, и возвращает кластер ошибок через туннель в нижнем правом углу структуры Case. Таким образом, отслеживаются любые ошибки от любых узлов внутри цепочки ошибок. Вне структуры Case **status** кластера ошибок разъединяется, и если ошибка есть, то значение **Error** передается на сдвиговый регистр на правой границе и затем выполняется случай **Error**. В этом случае информация об ошибке и время, когда она произошла, записываются в файл, и кластер ошибок очищается.

Обратите внимание, что теперь случай **Initialize** содержит функции ввода/вывода файлов среднего уровня вместо ВП Read Lines from File. Это позволяет передавать кластер ошибок способом, совместимым со всем остальным приложением, и помогает избежать возникновения диалоговых окон во время автономной работы приложения. Наконец, обратите внимание, что элементы **error in** и **error out** в этом примере не требуются, так как теперь это ВП верхнего уровня. Однако хорошей практикой будет написать все ВП так, чтобы они могли быть использованы как ВПП. Для ВП с графическим интерфейсом кластеры ошибок могут быть размещены на лицевой панели вне зоны видимости.

Ссылки

1. ВП Merge Multiple Errors и Clear Error All или ВП Specified можно загрузить с www.bloomy.com/lvstyle.

Шаблоны

Шаблоны (design patterns) – это ВП со стандартной архитектурой, предназначенные для решения стандартных проблем. Они состоят из набора структур, функций, элементов управления и процедур обработки ошибок, которые формируют характерные конструкции для таких повседневных задач, как создание циклов, обработка событий, переход из одного состояния в другое, обмен данными и инкапсуляция. Использование шаблонов улучшает продуктивность и качество работы. Кроме того, шаблоны можно комбинировать с ВПП, дополнительными элементами управления, утилитами и библиотеками и другими объектными структурами приложения. *Объектные структуры приложения* (application frameworks) – это более тщательно разработанный набор инструментов, который продвигает начальную точку разработки приложения. И базовые конструкции, и высокоуровневые объектные структуры могут служить шаблонами, если они привычны, узнаваемы и если их можно использовать повторно.

Шаблоны являются элементами хорошего стиля программирования. Если все ВП в приложении напоминают хорошо известные шаблоны, то разработчикам легче узнать, понять, поддерживать в рабочем состоянии и повторно использовать код друг друга. Вместо того чтобы создавать ВП с нуля, начните с шаблона и измените его согласно вашим нуждам. Это поможет сэкономить время, и вы можете быть уверены, что последовательно используете стандартную архитектуру приложений. Более того, в организациях с большим числом разработчиков шаблоны и образцы задают стандарты, которые гарантируют качество программного обеспечения. Как уже обсуждалось в главе 1 «Зачем нужен стиль», хороший стиль означает хорошую читаемость, надежность, эффективность, простоту, производительность и устойчивость к ошибкам.

В этой главе обсуждается множество шаблонов и выделяются те, которым отдают предпочтение инженеры Bloomy Controls[1]. Потому как тема довольно длинная, в табл. 8.1 приведен краткий список. В иллюстративных целях шаблоны применяются в ВП Torque Hysteresis, который мы уже рассматривали в предыдущих главах. Прежде чем начать, напомним несколько правил из главы 4 «Блок-диаграмма», так как они являются основополагающими принципами хороших шаблонов.

Правило 4.6 Блок-диаграммы верхних уровней собирайте из блоков ВПП

Правило 4.27 Старайтесь использовать структуру Sequence только при необходимости

Правило 4.28 Избегайте вложений с более чем тремя уровнями

Таблица 8.1. Сводка шаблонов и объектных структур, представленных в этой главе

	ВПП	ВП интерфейса	ВП верхнего уровня	Сложность			Общего назначения	Реального времени	Событийное управление	Неэффективно/опрос	Раздел
				низкая	средняя	высокая					
Начальный ВПП	+			+			+	+			8.1.1
Глобальный функционал	+			+			+	+			8.1.2
Непрерывный цикл	+			+			+	+		+	8.1.3
Цикл обработки событий					+		+		+		8.1.4
Классический конечный автомат	+	+		+			+	+		+	8.2.1
Конечный автомат с очередью	+				+		+	+	+		8.2.2
Событийно управляемый конечный автомат			+		+		+		+		8.2.3
Автомат событий		+	+			+	+		+		8.2.4
Параллельные циклы		+	+			+	+	+	+		8.3.1
Динамическая структура			+			+	+		+	+	8.4.1
Структура со многими циклами		+	+			+	+		+		8.4.2
Модульная структура со многими циклами			+			+	+	+	+		8.4.3

Гораздо проще распознать какой-то шаблон, когда блок-диаграмма состоит из блоков. По возможности, нужно группировать узлы, которые работают вместе над одной задачей, в ВПП. На блок-диаграмме ВП верхнего уровня должно быть как можно меньше функций, не входящих в используемый шаблон. Структуры Sequence обычно не используются потому, что они нарушают концепцию потока данных и, как следствие, порядок выполнения функций. Забавно, но самый распространенный способ использования структуры Sequence помогает очертить и выделить шаблон. Структура кадров (Flat Sequence) может использоваться как внешний ограничитель ВП верхнего уровня или для инициализации переменных или ресурсов, но в других случаях ее следует избегать. Шаблоны в данной главе используют концепцию потока данных как альтернативу структуре Sequence. Отказ от использования этой структуры помогает предотвратить возникновение громоздких вложенных структур. Чем больше слоев структур внутри структур на

блок-диаграмме, тем сложнее понять направление потока данных, что означает плохую читабельность кода. Большинство шаблонов имеют три и меньше вложенных уровня.

8.1. Простые шаблоны

Простые шаблоны выглядят привычно, их легко понять и применять. К ним относятся: ВПП Immediate, Functional Global, Continuous Loop и Event-Handling Loop.

8.1.1. Шаблон ВПП Immediate

Шаблон начального ВПП (Immediate subVI) состоит из узлов, составляющих ВПП, и процедуры отслеживания ошибок. В нем нет циклов, диалоговых окон или панелей графического интерфейса. Вместо этого ВПП Immediate выполняет код от начала и до конца, в том порядке, в каком кластер ошибок проходит через узлы на блок-диаграмме. Встроенная процедура отслеживания ошибок удовлетворяет правилам главы 7 «Обработка ошибок». Она включает структуру Error Case для того, чтобы не выполнять код, когда обнаружена ошибка, стандартные кластеры **error in** и **error out** и стандартные назначения на соединительной панели; информация об ошибках передается через проводник данных кластера ошибок. Как это следует из названия, шаблон ВПП Immediate является самым простым и самым популярным шаблоном.

Вы можете использовать ВПП с шаблоном обработки ошибок, чтобы улучшить ВПП Immediate. Этот шаблон доступен при выборе опции **New** в меню **VI ⇒ From Template ⇒ Frameworks ⇒ SubVI with Error Handling**. Этот шаблон состоит из кластеров ошибок, стандартной соединительной панели 4×2×2×4, назначенных терминалов ошибок, структуры Case, селектор которой соединен с терминалом **error in**. Данный вариант показан на рис. 8.1. Таким образом, вы сможете сэкономить только минуту или две, но если вам нужно создать приложение, в котором 100 и более начальных ВПП, эти минуты сложатся в часы. Использование этого шаблона также помогает применять правила для лицевых и соединительных панелей и процедур обработки ошибок, которые мы обсуждали в главе 3 «Стиль лицевой панели», главе 5 «Иконка и контакты» и в главе 7 «Обработка ошибок». Вы можете встроить дополнительные функции в этот шаблон, например стандартную документацию, иконку, определение типов данных элементов и многое другое, чтобы сэкономить время.

На рис. 8.2 приведен пример шаблона ВПП Immediate применительно к ВП Torque Hysteresis: ВП Calculate Target Positions вычисляет несколько положений, опираясь на параметры, заданные пользователем. Раскладка, цвет и текст на лицевой панели удовлетворяют правилам для ВПП, которые мы обсуждали в главе 3. Код на блок-диаграмме находится внутри структуры Error Case. Обратите внимание, что ни одна функция в этом ВП не имеет терминалов ошибок. Однако обработка ошибок в ВПП повышает эффективность в случае появления ошибок и помогает упростить поток данных в вызывающем ВП.

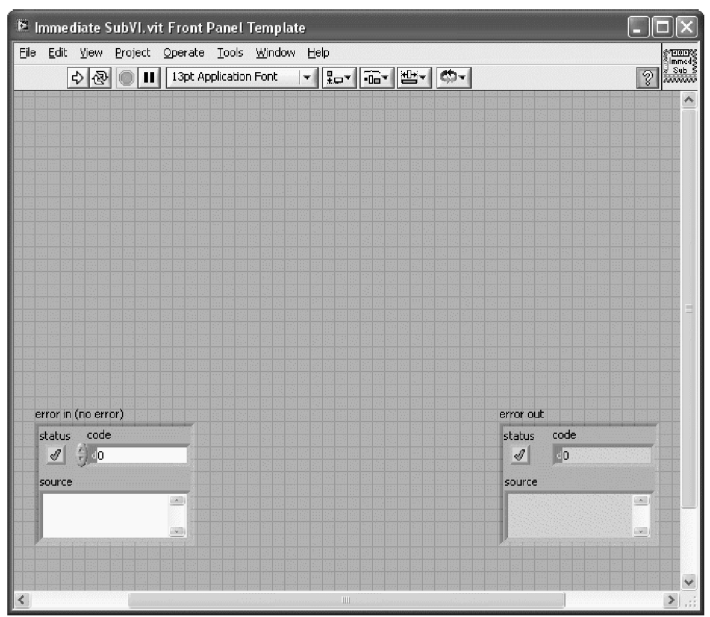


Рис. 8.1а. Лицевая панель шаблона ВПП Immediate содержит кластеры ошибок, стандартную соединительную панель 4×2×2×4 и назначения терминалов

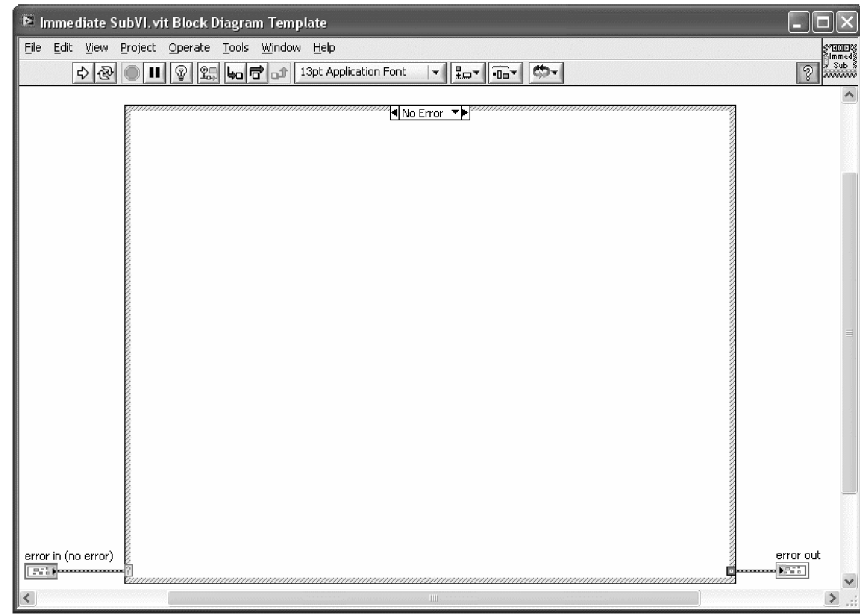


Рис. 8.1б. Блок-диаграмма содержит структуру Case, селектор которой соединен с терминалом ***error in***

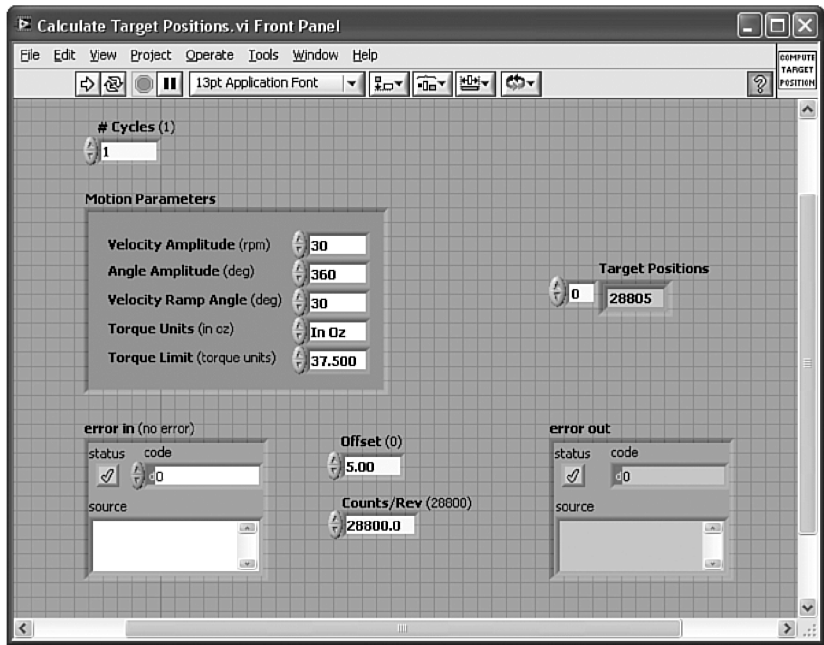


Рис. 8.2а. Раскладка, цвет и текст на лицевой панели ВПП Immediate удовлетворяют правилам для ВПП, которые мы обсуждали в главе 3

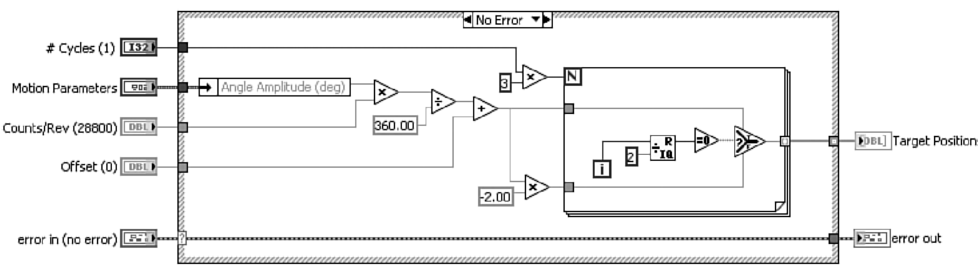


Рис. 8.2б. Обработка ошибок в ВПП повышает эффективность в случае появления ошибок и помогает упростить поток данных в вызывающем ВП

Если вам интересно, почему этот шаблон называется ВПП Immediate, хотя образец в LabVIEW называется ВПП с обработкой ошибок (SubVI with Error Handling), то позвольте мне объяснить. Термины *ВПП* и *error handling* используются неправильно. Сложность современных приложений LabVIEW такова, что ВПП могут иметь любой уровень сложности – от принципиальных составляющих до низкоуровневых драйверов. Шаблон ВПП Immediate предназначен для очень

простых ВПП. В главе 7 мы определили обработку ошибок как практику отслеживания ошибок и создания отчетов об ошибках. Поэтому ВПП Immediate выполняет свой код непосредственно, без взаимодействия с пользователем, без использования циклов или отчетов об ошибках, которые могут заставить ждать вызывающее приложение.

8.1.2. Шаблон Functional Global

Шаблон Functional Global (Глобальный функционал) состоит из ВПП, содержащего цикл While, структуры Case, перечня и элементов управления для записи и чтения данных. Цикл While содержит один или более не инициализированный сдвиговый регистр, к терминалу условия выхода из цикла подсоединена логическая переменная, которая останавливает цикл после первой же итерации. Основная цель цикла – сохранять глобальные данные в сдвиговых регистрах. В структуре Case предусмотрены отдельные случаи для чтения и записи данных сдвиговых регистров. Дополнительно вы можете добавить случаи для инициализации или изменения размеров таких структур данных, как массивы и строки, вычисления или манипуляции с данными. Перечень осуществляет программный выбор операций чтения, записи и других поддерживаемых операций с блок-диаграммы вызывающего ВП. Этот ВПП используется на всех участках приложения, где требуется доступ к данным, по аналогии с глобальными переменными.

Глобальные функционалы также называют **LabVIEW 2 style globals**, потому что они были только способом обмена данными и не поддерживали идею потока данных до введения в LabVIEW 3.0 локальных и глобальных переменных. Каждая копия локальной или глобальной переменной, открытая для чтения, создает копию данных во время считывания, что позволяет одновременно производить операции записи, которые не влияют на считанные данные. Это ведет к выделению дополнительных областей памяти и созданию условий соревнования. Условия соревнования возникают, когда запись данных в локальную или глобальную переменную изменяют данные после того, как предыдущее значение было считано, таким образом, считанные данные в буфере отличаются от новых данных в переменной. В функциональной переменной каждый сдвиговый регистр обеспечивает один буфер памяти, в который нельзя одновременно и записать данные, и считать. В дополнение у глобальных функционалов есть блок-диаграмма, на которой могут быть дополнительные функции, помимо чтения и записи.

На рис. 8.3 показаны лицевая панель и блок-диаграмма очень простого глобального функционала для хранения численных данных. Она поддерживает только две операции: чтение и запись, которым соответствуют случаи структуры Case. Обратите внимание, что функциональная переменная удовлетворяет тем же критериям, что и ВП Immediate. Обработка ошибок реализована так, как обсуждалось в предыдущем разделе.

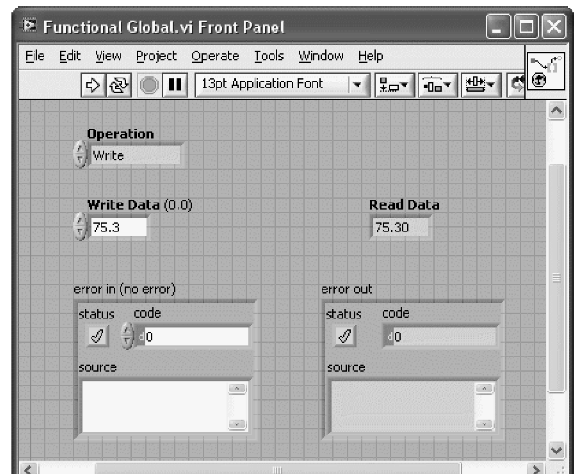


Рис. 8.3а. Лицевая панель функциональной переменной, которая хранит численные данные. Она содержит перечень для выбора желаемой операции, элементы управления для записи данных и индикаторы для отображения считанных данных, кластеры ошибок

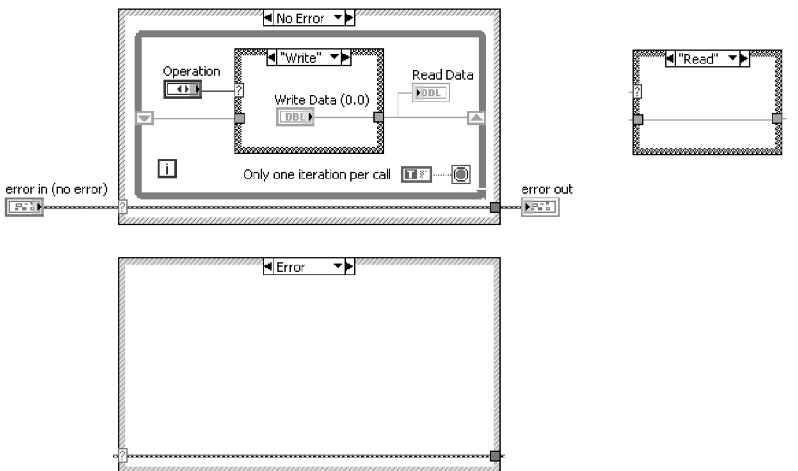


Рис. 8.3б. Блок-диаграмма функциональной переменной содержит цикл While и сдвиговый регистр для хранения данных, структуру Case для чтения и записи данных и передачу кластера ошибок

8.1.3. Шаблон Continuous Loop

Шаблон непрерывного цикла (Continuous Loop) состоит из единственного цикла While или цикла заданной длительности, сдвиговых регистров, процедуры вычисления времени завершения цикла и процедуры отслеживания ошибок. Этот шаблон применим как для ВП верхнего уровня, так и для ВПП.

Правило 8.1 Используйте несколько критериев выхода из цикла

Тщательно продумайте условия остановки/возобновления работы непрерывных циклов. Убедитесь, что прерывание цикла происходит корректно и, возможно, без использования инструмента Abort Execution. Хорошей практикой будет использование нескольких критериев, к ним может относиться событие пользовательского интерфейса (нажатие кнопки **Stop**), возникновение ошибки, возможно, задание максимального временного интервала или числа итераций. Никогда не используйте Abort Execution для остановки непрерывного цикла. Вместо этого используйте подходящие условия выхода из цикла и всегда скрывайте кнопку Abort Execution в ВП верхнего уровня в разрабатываемых вами приложениях. Это можно сделать, выбрав **File ⇒ VI Properties**, затем в выпадающем меню **Category** выбрать **Window Appearance ⇒ Top-level application**.

Правило 4.35 Используйте сдвиговые регистры вместо локальных и глобальных переменных

Правило 4.36 Группируйте большую часть сдвиговых регистров у верхней границы цикла

Правило 4.37 Маркируйте проводники данных выходящие из левого терминала сдвигового регистра

Используйте сдвиговые регистры вместе с циклами шаблона Continuous Loop для хранения и передачи данных между итерациями цикла. Сдвиговые регистры предпочтительнее локальных, глобальных и функциональных переменных, потому что поддерживают концепцию потока данных. Концепция потока данных позволяет более эффективно работать с памятью, избегать возникновения условий соревнования и делает код удобочитаемым. Группируйте сдвиговые регистры около верхней границы цикла и помечайте проводники, выходящие из левых терминалов сдвиговых регистров.

Правило 8.2 Используйте Timed Loop (цикл с ограничением по времени исполнения или числу итераций) для задания точного или сложного времени выполнения задачи, в других случаях используйте цикл While

Тактированные циклы (Timed Loop) используются в случае жестокого ограничения времени выполнения задачи, что невозможно реализовать с помощью цикла While. Обычно это необходимо для приложений, работающих в реальном времени. Например, Timed Loop может использовать аппаратное обеспечение или высокоскоростной процессор в режиме реального времени, чтобы получить разрешение во времени порядка 1 микросекунды. Вы можете настроить период, сдвиг и другие параметры, используя узел ввода и ВП с палитры **Timed Structures**. Дополнительно вы можете установить приоритет выполнения цикла Timed

Loop вне зависимости от приоритета вызывающего ВП. Однако стандартный цикл While проще, требует меньше памяти и в конечном итоге обработки и подходит для большинства настольных приложений. В общем случае следует пользоваться циклом While, а цикл Timed Loop использовать только для сложных и более требовательных к временной точности приложений.

Правило 8.3 Вставляйте задержки в непрекращающиеся циклы While

Используйте функции Wait (ms), Wait Until Next ms Multiple function или ВП Time Delay Express, чтобы встроить в цикл While задержку по времени. Каждый из этих узлов предоставляет вам канал управления центральным процессором и заставляет цикл ждать заданное число миллисекунд на каждой итерации. Это позволяет выполнять параллельные задачи и улучшает общую производительность приложения. Помните, что LabVIEW – многозадачная система, даже если узлы на блок-диаграмме выполняются последовательно. В LabVIEW есть множество процессов, выполняемых параллельно, включая такие элементы пользовательского интерфейса и блок-диаграммы, как параллельные циклы и узлы. Дополнительно операционная система может быть занята другими задачами на заднем фоне или работать с другими приложениями параллельно с LabVIEW. Добавление задержек в цикл While дает возможность этим параллельным задачам выполняться.

Функция Wait (ms) и ВП Time Delay Express добавляют задержку, синхронизированную с основным кодом цикла. Общее время выполнения цикла равняется наибольшему времени (задержки или выполнения кода цикла). Если указанное время задержки больше времени выполнения цикла, то цикл приостанавливает работу на время, равное разности времен выполнения цикла и задержки. Например, если выполнение кода занимает примерно 12 мс, а вы используете функцию задержки со значением 50 мс, то общее время выполнения цикла составит 50 мс, из них 38 секунд ожидания. Если код выполняется дольше 50 мс, то дополнительной задержки не будет. Однако функция синхронизированной задержки дает потоку контроль над процессором вне зависимости от времени задержки. Поэтому добавление даже очень маленького времени задержки в цикл дает свои преимущества.

Функция Wait Until Next ms Multiple ждет, пока значение системных часов не станет кратно указанному **millisecond multiple**. Вы можете использовать эту функцию для синхронизации времени выполнения различных циклов. Но убедитесь в том, что выполнение вашего кода не превышает того интервала времени, который вы пытаетесь установить, иначе время выполнения цикла удвоится. Если цикл опрашивает другие источники, такие как порты коммуникации, устройства, объекты интерфейса пользователя, или когда темп выполнения задается внешним источником, а не временем исполнения кода, тогда следует использовать функцию Wait (ms). Установите значение **milliseconds to wait** равным самому большому возможному значению, при котором цикл еще отвечает. Обратите внимание, что в конструкциях, основанных на структуре событий, к ним относится и Timed Loop, задержки не нужны. Эти структуры бездействуют, когда нет активных событий.

Шаблон Continuous Loop представлен на рис. 8.4. Лицевая панель состоит из кластера ошибок и кнопки **Stop**. Кнопка **Stop** имеет цветовую схему и шрифт, более пригодные для индустриального приложения, чем стандартные элементы управления с палитры **Boolean**. Скопируйте и вставьте кнопку **Stop**, чтобы создать меню логических переменных. Настройте текст и уникальные метки для каждого элемента меню. Правила для текста на лицевой панели, в том числе и текста в метках элементов управления, обсуждались в главе 3. Кластеры ошибок связаны с нижним левым и правым терминалами стандартной соединительной панели 4×2×2×4, как рекомендуется в главе 5. При желании вы можете убрать эти элементы из видимой зоны. Блок-диаграмма шаблона Continuous Loop содержит цикл While с обработкой ошибок, сдвиговыми регистрами, кнопкой **Stop** и условиями выхода и цикла. Обработка ошибок состоит из отслеживания ошибок при передаче кластера ошибок и создания отчета об ошибках средствами ВП General Error Handler вне цикла. Три сдвиговых регистра расположены в верхней части цикла. Замените логические данные теми, что должны сохраняться между итерациями. Функция Wait добавлена для эффективности. Статус ошибки и кнопка **Stop** используются как начальные критерии выхода из цикла. Функция Compound Arithmetic используется вместо бинарной функции ИЛИ для обеспечения гибкости в добавлении терминалов.

Измените шаблон Continuous Loop так, чтобы использовать его как ВПП или ВП верхнего уровня. Для создания ВПП удалите ВП обработки ошибок General

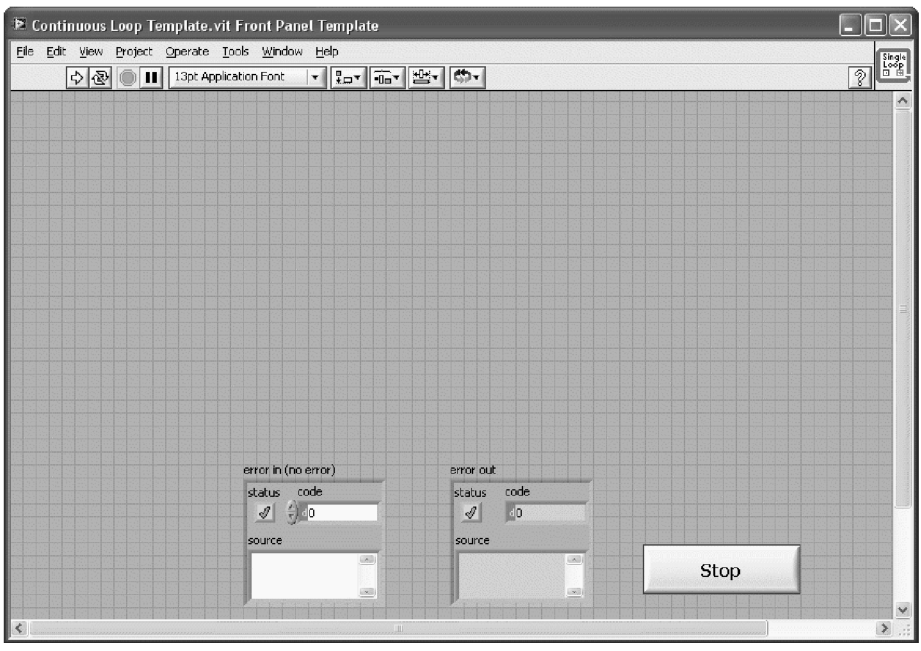


*Рис. 8.4а. Лицевая панель Continuous Loop состоит из кластера ошибок и кнопки **Stop***

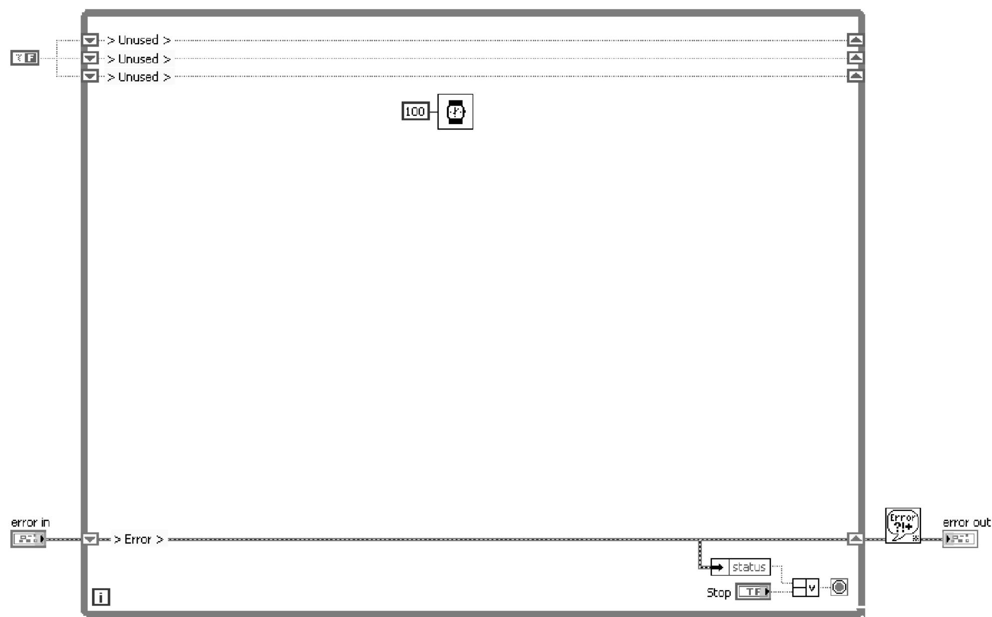


Рис. 8.4б. Блок-диаграмма шаблона Continuous Loop содержит цикл While с обработкой ошибок, сдвиговыми регистрами, кнопкой *Stop* и условиями выхода и цикла

Error Handler и логические элементы управления, окружите блок-диаграмму структурой Error Case. Для ВП верхнего уровня спрячьте кластеры ошибок и измените лицевую панель. На рис. 8.5 показан ВП Torque Hysteresis, реализованный через шаблон Continuous Loop. Лицевая панель содержит меню логических элементов управления и графический индикатор в одной вкладке, а результаты в другой. На блок-диаграмме функции разделены по трем структурам Case. В случае True первой структуры Case находится диалоговое окно, в котором оператор задает параметры тестируемого образца и параметры движения. Этот случай активируется, когда пользователь нажимает кнопку **New UUT**. Информацию об образце и параметрах движения возвращают ВПП и сохраняют сдвиговые регистры. Также установлен логический значок, который показывает, что данные об образце допустимы и логические данные распространяются через сдвиговые регистры. Средняя структура Case запускает тесты и анализирует данные, когда пользователь нажимает кнопку **Run Test** и введена допустимая информация об образце. Полученная зависимость момента вращения в зависимости от угла и вычисленные данные статистики хранятся в сдвиговых регистрах. Правая структура Case обеспечивает запись данных измерений в файл, когда пользователь нажимает кнопку **Log Data**. Цикл прерывается, если пользователь нажимает кнопку **Quit** или случается ошибка. Обратите внимание, что сдвиговые регистры обеспечивают возможность выполнения каждого задания по требованию, вместо того чтобы последовательно подавать самые последние полученные данные на терминалы

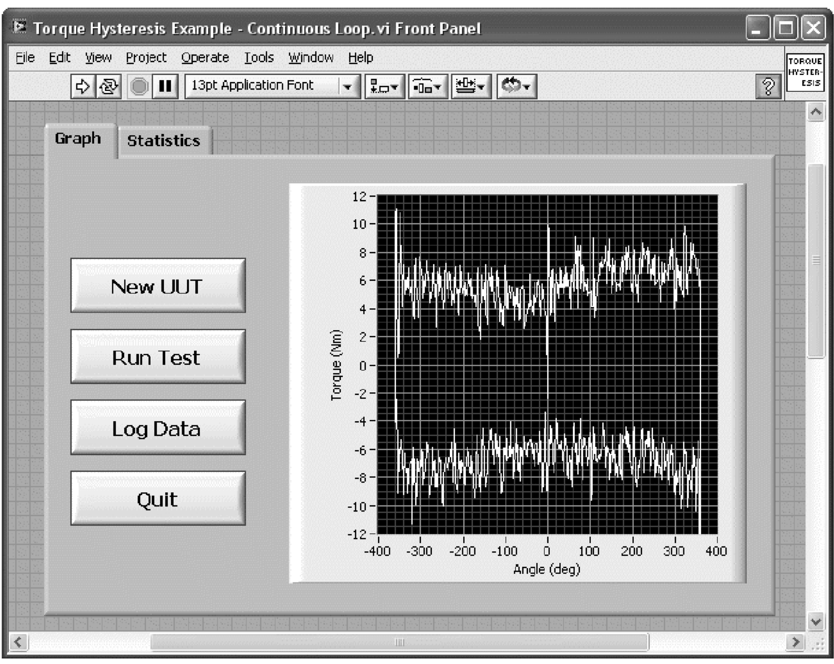


Рис. 8.5а. Лицевая панель ВП Torque Hysteresis на основе шаблона Continuous Loop состоит из меню логических переменных, графика и вкладок.

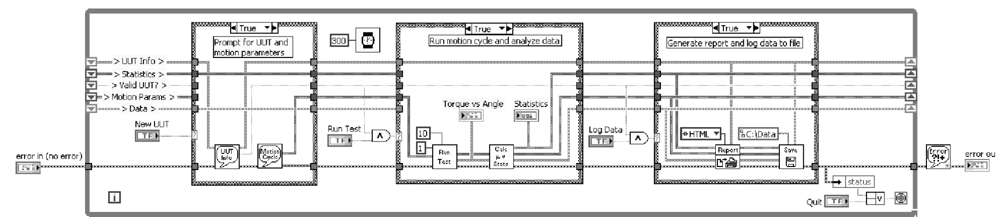


Рис. 8.5б. Блок-диаграмма разделена для выполнения трех основных задач, которые могут выполняться по требованию. Данные хранятся в сдвиговых регистрах

ВПП. В частности, это дает возможность оператору один раз настроить параметры образца и движения и выполнить несколько тестов перед тем, как записать результаты.

Правило 8.4 Избегайте опрашивать объекты графического интерфейса

Непрерывный опрос логических переменных в цикле, как тот, что показан в предыдущем примере, обычно используется разработчиками, не знакомыми со структурой событий. Сюда же относятся пользователи базового пакета LabVIEW

и версии LabVIEW ниже 6.1, в которой впервые появились структуры событий. В общем, не стоит опрашивать элементы графического интерфейса. Ниже приведен более эффективный способ.

8.1.4. Цикл с обработкой событий

Шаблон цикла с обработкой событий (Event Handling Loop) облегчает управление по событиям в LabVIEW. Он состоит из структуры событий, расположенной в цикле While. Структура событий содержит случаи для каждого события, которые реализуют некоторый код, когда происходит указанное событие. Такая структура является аналогом структуры Case с событийно-управляемым селектором. Структура событий улучшает эффективность обнаружения и обработки событий графического интерфейса пользователя и других определенных пользователем событий. В отсутствии зарегистрированных событий процессор освобождается для работы над другими задачами.

Шаблон цикла с обработкой ошибок очень полезен в приложениях, включающих любое количество событий интерфейса пользователя. Практически любой тип активности пользователя можно отследить и перевести на язык событий. Дополнительно структура событий поддерживает события, определенные пользователем, связанные, например, с последовательностью измерений. Более того, любое приложение, содержащее элементы графического интерфейса, только выигрывает от применения циклов, поддерживающих события. Единственная загвоздка в том, что структура событий недоступна в базовом пакете LabVIEW.

Правило 8.5 Используйте событие Value Change (Изменение Значения) для большинства элементов управления графического интерфейса

Событие **Value Change** регистрирует изменение пользователем данных элемента управления. И если значение меняется, вне зависимости логическая ли это кнопка, численный элемент управления, строка или еще какой-либо тип элементов управления, то регистрируется событие Value Change и выполняется соответствующий код. Для простых приложений можно использовать событие Value Change для обнаружения изменений значений. Это скажется и на производительности ВП, и на ее читабельности. Используйте такие события, как Key down (Клавиша нажата), Mouse Down (Щелчок мыши) и другие специфичные для каждого элемента события, для достижения специальных эффектов.

Правило 8.6 Терминалы элементов управления размещайте внутри их случая Value Change

Обратите внимание, что структура событий дает вам возможность считывать новые и старые значения элементов управления из узла Event Data Node. Логические элементы, настроенные на любое защелкивающее действие, однако, не сбросят свое значение на значение по умолчанию, пока данные с них не будут считаны через терминал. Размещение элементов управления внутри структуры событий, соответствующей случаю Value Change, гарантирует то, что данные с терминалов будут считаны, а защелкивающиеся логические переменные сбросятся каждый раз, когда значение изменится. Хорошей практикой будет размещение терминалов всех элементов управления внутри соответствующих случаев Value Change структуры событий, если только эти терминалы не нужны где-то еще. Таким образом, вы получите полезную для разработчиков точку отсчета для обнаружения элементов управления.

Правило 8.7 Измените размер узла Even Data Node, чтобы спрятать не используемые терминалы

В большинстве случаев я убеждался в том, что использую один, два (а то и ни одного) терминала Event Data Node. Сэкономьте место и устраните путаницу внутри структуры событий, уменьшив Event Data Node так, чтобы спрятать не используемые терминалы. В примерах, приведенных в этой главе, узел Event Data Node уменьшен до размера одного терминала и сдвинут в левую нижнюю зону структуры событий.

Правило 8.8 Избегайте продолжительных событий Timeout (По истечении времени)

По умолчанию структура событий имеет случай события Timeout (По истечении времени), которое становится доступным, как только терминал Timeout подсоединен. Событие «по истечении времени» наступает, когда в течение определенного времени не происходит никаких других событий, тогда выполняется соответствующий код. Событие Timeout зачастую используют ошибочно. Может показаться очень привлекательным разместить непрерывное задание внутри события Timeout, потому что это позволит сэкономить на создании дополнительного параллельного цикла и схемы сообщения, тем самым снизив сложность блок-диаграммы. Но появление какого-либо зарегистрированного события приостановит выполнение события Timeout до тех пор, пока зарегистрированное событие не будет обработано. Программа не может контролировать максимальное число регистрируемых событий и то, как часто они могут происходить. Поэтому на событие Timeout нельзя положиться при выполнении периодических во времени задач. Более того, длительные события Timeout приводят к тому, что структура событий ведет себя как обыкновенный цикл, опрашивающий элементы графического интерфейса через определенные промежутки времени. Код «по истечении времени» вызывает задержки в реакции на другие события и снижает общую эффективность структуры событий.

На рис. 8.6 приведен образец шаблона Event Handling Loop. Лицевая панель содержит несколько вкладок, одна страница содержит логические кнопки управления, настроенные под индустриальный графический интерфейс, затем идет страница с кластерами ошибок. Блок-диаграмма состоит из структуры событий внутри цикла While, сдвиговых регистров и процедуры выключения вне цикла. Структура событий содержит случаи для событий изменения значения каждой кнопки и случаи **Panel Close?** (Панель закрылась?) и **Application Exit?** (Приложение завершено?). Диалоговое окно с двумя кнопками предлагает пользователю подтвердить выход из приложения. Дополнительно структура событий настроена так, чтобы удовлетворять правилам хорошего стиля. Терминалы управляющих элементов расположены в соответствующих случаях, а событие Timeout удалено.

Обратите внимание, что к завершению приложения приводят три независимых события. На рис. 8.6в показаны случаи событий **Quit Value Change** (Кнопка Quit, значение изменилось), **Panel Close?** и **Application Instance Close?** (Копия

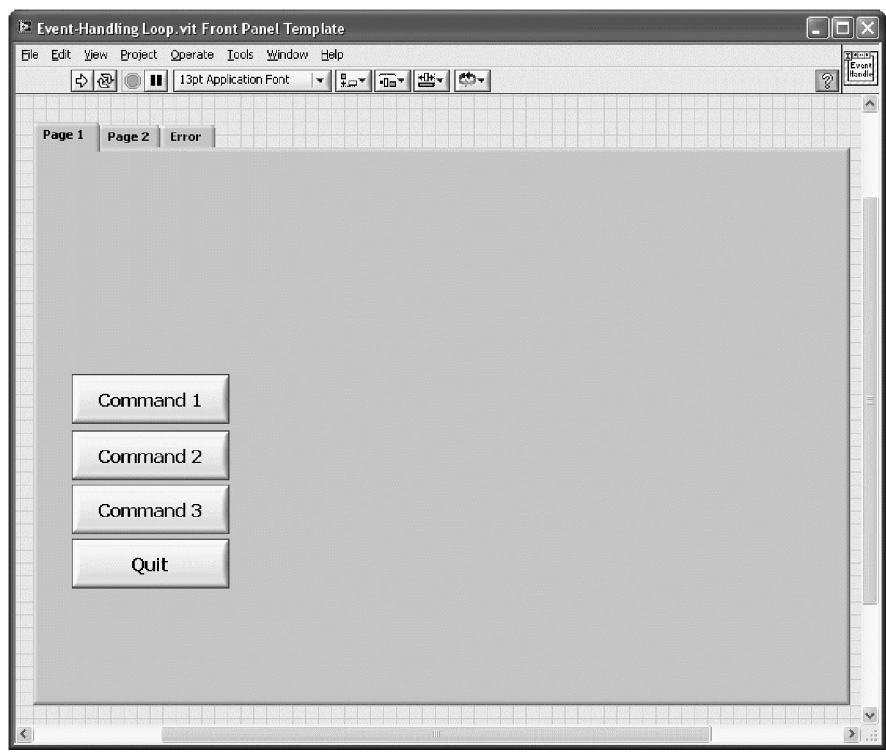


Рис. 8.6а. Лицевая панель шаблона Error Handling Loop содержит вкладки с несколькими командными кнопками, кластерами ошибок и запасную вкладку

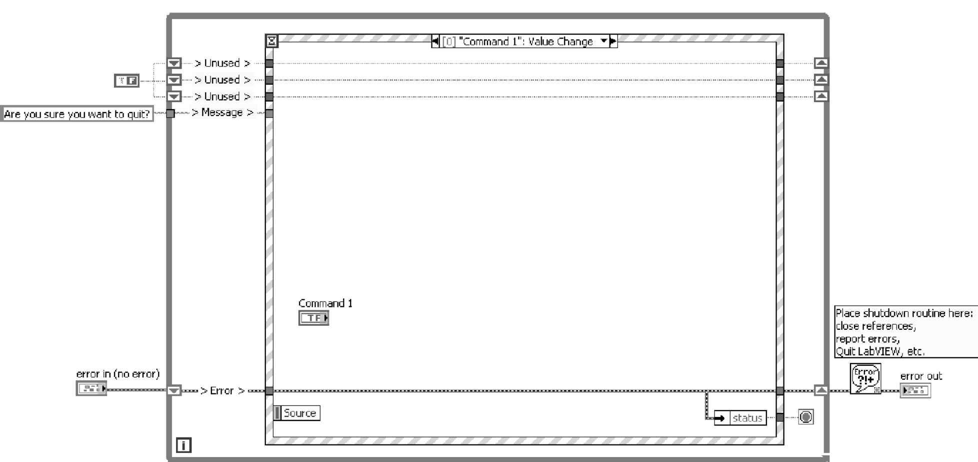


Рис. 8.6б. Блок-диаграмма состоит из структуры событий, расположенной внутри цикла While, сдвиговых регистров и процедуры завершения работы вне цикла. Терминалы управляющих элементов расположены внутри соответствующих случаев событий Value Change

приложения завершена?). Все три события ведут к появлению диалогового окна запроса подтверждения от пользователя, если запрос получен, то цикл прерывается. После того как остановлен цикл, запускается процедура завершения приложения. Тем самым снимается необходимость трижды повторять эту процедуру для разных событий. По желанию можно настроить процедуру завершения работы на то, чтобы закрывать все открытые ссылки, отчеты об ошибках, завершать работу в LabVIEW. Поскольку события **Panel Close?** и **Application Instance Close?** обычно ведут к немедленному завершению работы с приложением, то они должны быть отменены, чтобы запустилась процедура завершения работы, расположенная вне цикла While. Это достигается путем подачи значения TRUE на терминал **Discard?** (Отменить?).

На рис. 8.7 показан ВП Torque Hysteresis на основе шаблона цикла с событиями. Лицевая панель идентична той, что приведена на рис. 8.5а, и поэтому не показана. Каждая управляющая кнопка имеет соответствующий случай события Value Change, в котором выполняется тот же код, что и в случаях структуры Case на рис. 8.5б. Цикл с обработкой событий более компактный, гибкий и эффективный, чем Continuous Loop (Непрерывный цикл). Одна структура событий с множеством случаев компактнее множества структур Case. Кроме этого, к ней проще добавить случаи или разделить уже существующие без изменения размеров блок-диаграммы. Более того, такой способ эффективнее, так как блок-диаграмма не выполняется до тех пор, пока не произошло событие, освобождая процессор для работы над другими задачами.

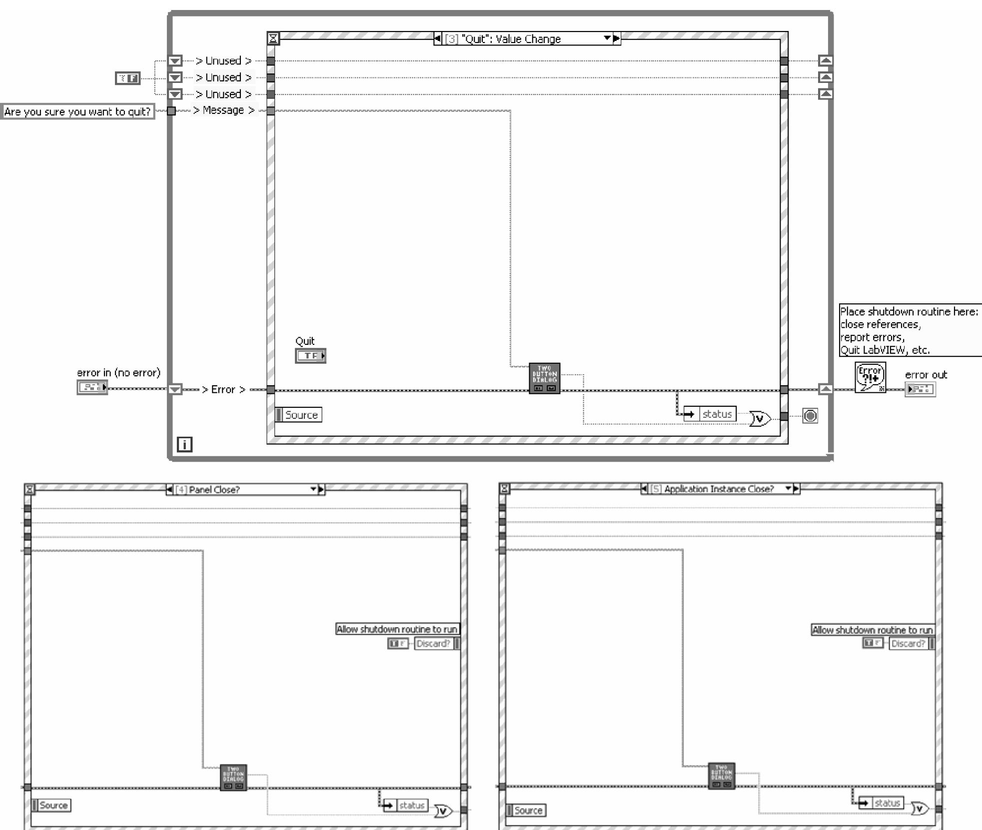


Рис. 8.6в. Три различных события, вызывающих завершение работы приложения: Quit Value Change, Panel Close? и Application Instance Close?. Последние два события отменены для того, чтобы сделать возможным выполнение внешней процедуры завершения работы

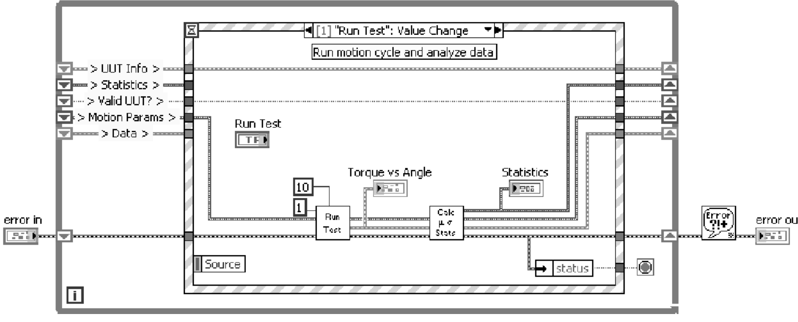


Рис. 8.7. ВПTorque Hysteresis, выполненный на шаблоне цикла поддерживающего события. Это более компактная, гибкая и эффективная альтернатива использованию непрерывного цикла (см. рис. 8.5)

8.2. Конечные автоматы

Конечные автоматы (State machine) – наиболее разрекламированные шаблоны в LabVIEW. Существует множество вариаций, большинство из которых состоит из структуры Case, расположенной внутри цикла While, сдвиговым регистром или конструктом сообщений (альтернативная схема обмена данными между итерациями, соединенным с терминалом селектора структуры Case. Каждый случай структуры Case содержит субдиаграмму, соответствующую состоянию приложения. Селектор может быть целочисленного, строкового или пронумерованного типа данных (перечень) и будет определять состояния системы. Сдвиговый регистр или конструкт сообщений передает следующее, выбранное в предыдущем случае, состояние на терминал селектора на следующей итерации. Этот принцип проиллюстрирован на рис. 8.8, где используется классическая форма конечного автомата. В обычном приложении выбор состояния определен событием пользовательского интерфейса, шагом в последовательности тестов или измерений, или по результатам предыдущего состояния.

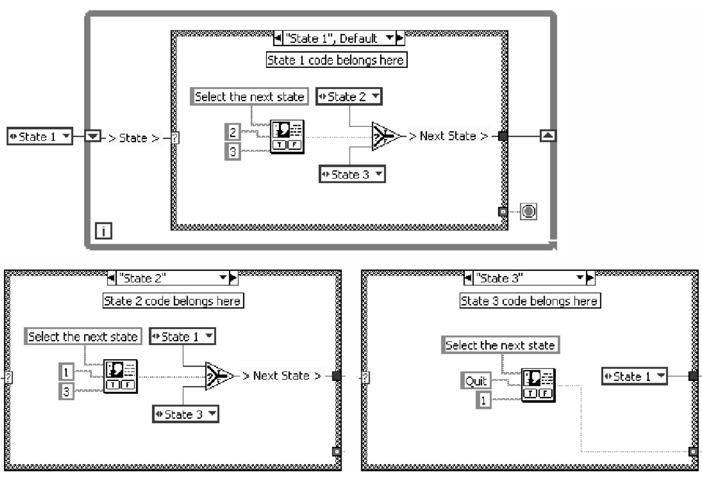


Рис. 8.8. Шаблоны конечного автомата состоят из структуры Case, расположенной внутри цикла While, сдвигового регистра или другой конструкции передачи сообщений между итерациями, соединенной с селектором структуры Case. Каждый случай этой структуры соответствует состоянию приложения

Правило 8.9 Используйте шаблон конечного автомата в большинстве ВП средней или высокой сложности

Конечный автомат – это невероятно гибкая структура с большими возможностями расширения. Сдвиговые регистры или другие конструкции сообщения позволяют перепрыгивать с одного любого состояния в любое другое. Каждое состоя-

ние может определять условия работы цикла While, так что его можно остановить в любом состоянии. Также вы легко можете добавить код в приложение, просто добавив случаи в структуру Case, без увеличения общего размера блок-диаграммы. Эти особенности делают схему конечного автомата бесценной. Использовать эту схему в ВП среднего и высшего уровня сложности – беспроигрышный вариант.

Правило 8.10 Основные состояния приложения берите из спецификации или проектной документации

Для того что бы использовать схему конечного автомата, приложение нужно разделить на серию состояний. Это можно сделать на стадии планирования, опираясь на требования, указанные в спецификации или проектной документации, или на стадии реализации, опираясь на опыт и интуицию. Как уже говорилось в главе 2 «Приготовка к хорошему стилю», чем тщательнее планирование, тем лучше конечный результат. Блок-диаграмма конечного автомата – это ключевой компонент UML и идеальный метод определения основных состояний системы. Дополнительно доступна утилита от NI под названием State Diagram Toolkit, которая предоставляет редактор схем, по которым соответствующий код LabVIEW создается автоматически. Однако несколько применяемых в этой главе схем конечного автомата нельзя создать, используя State Diagram Toolkit. Вместо этого следует начинать с шаблонов.

Правило 8.11 Разделяйте основные состояния на второстепенные

Иногда функциональные состояния системы нельзя точно перевести на язык блок-диаграмм LabVIEW, которые бы хорошо работали внутри схемы конечного автомата. По мере того как вы разрабатываете код, могут проявиться технические аспекты, которые приведут к другим очертаниям состояний. Например, данное функциональное состояние может иметь общую задачу с несколькими другими функциональными состояниями, что дает вам право разделить задачу на несколько различных случаев, так, чтобы к ним можно было обращаться из разных состояний. Если становится очевидным, что код в данном состоянии слишком большой для одного случая, не бойтесь создавать столько случаев, сколько необходимо. Помните о том, что ваш код должен состоять из модулей – ВПП, и определяйте состояния для возможности повторного использования. Для лучших результатов применяйте комбинацию планирования, опыта и интуиции.

Правило 8.12 Используйте перечень в качестве селектора случаев

Перечень, сохраненный как тайпдеф (type definition – определение типов), обеспечивает документацию и улучшает возможности поддержки. Когда перечень соединен с селектором структуры Case, то текстовые элементы перечня появляются в зоне селектора структуры Case. Используйте лаконичные и понятные имена для каждого состояния и вводите их как элементы перечня. Любые константы, созданные из тайпдефа, поддерживают синхронизацию имен элементов с тапдефом. Поэтому пронумерованные константы используются для определения следующего состояния в каждом случае. Когда элементы добавляются в перечень или удаляются из него, то соответствующие константы автоматически обновляются. Перечни, как тайпдеф, рекомендованы для всех вариаций шаблона конечного автомата, использующих структуру Case.

Правило 8.13 Минимизируйте код, внешний по отношению к структуре Case

Правило 8.14 Добавьте состояния Initilaize (Инициализация), Idle (Бездействие), Shutdown (Выключение) и Blank (Пустое)

Для большинства приложений нужен код общего обслуживания: процедуры инициализации и выключения и, возможно, процедуры, которая выполняется тогда, когда больше ничего не выполняется. Например, какие-то значения и свойства управляющих задаются до выполнения любых тестов. Может также понадобиться обработчик ошибок или может быть запущена процедура опроса ресурсов, когда не обрабатываются никакие другие состояния. Процедура выключения необходима для того, чтобы корректно завершить все соединения с устройствами, файлами данных и удаленными приложениями. Создайте соответствующие состояния **Initialize** (Инициализация), **Idle** (Бездействие) и **Shutdown** (Выключение). Об этих процедурах вы могли и не задумываться на стадии планирования, но, по моему опыту, они нужны всегда. Используйте шаблон конечного автомата по максимуму и встройте эти процедуры как состояния конечного автомата, а не внешним кодом. Наконец, создайте состояние **Blank** (Пустое), содержащее передачу данных сдвиговых регистров в целях дублирования. Новые состояния можно будет добавить, скопировав это пустое состояние, это сэкономит время.

Например, блок-диаграмма на рис. 8.9а содержит код инициализации и выключения в последовательных кадрах слева и справа от конечного автомата. Также конечный автомат опрашивает кнопку **Stop** слева от структуры Case и проверяет статус ошибки справа. Это неэффективное использование места. На рис. 8.9б содержится эквивалентный код, использующий конечный автомат с состояниями Инициализация, Бездействие, Выключение и Пустое. Два терминала сдвиговых регистров обеспечивают буфер, в котором хранится информация о двух следующих состояниях. Каждый раз, когда вызывается состояние **Idle**, оно добавляет другое состояние **Idle** в конец буфера, тем самым гарантируется, что состояние **Idle** чередуется с другими. Кроме того, состояние **Idle** проверяет статус кластера ошибок и останавливает цикл While, если происходит ошибка. Следовательно, блок-диаграмма на рис. 8.9б эффективно использует место, заменив внешний код состояниями конечного автомата.

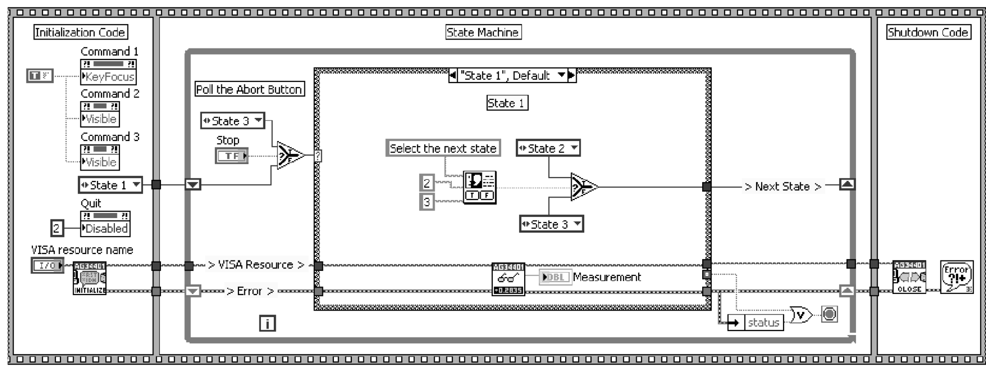


Рис. 8.9а. Шаблон конечного автомата в комбинации со структурой кадров, посредством которой осуществляется инициализация, опрос управляющих элементов и выключение, проверяет статус кластера ошибок. Блок-диаграмма использует место по горизонтали

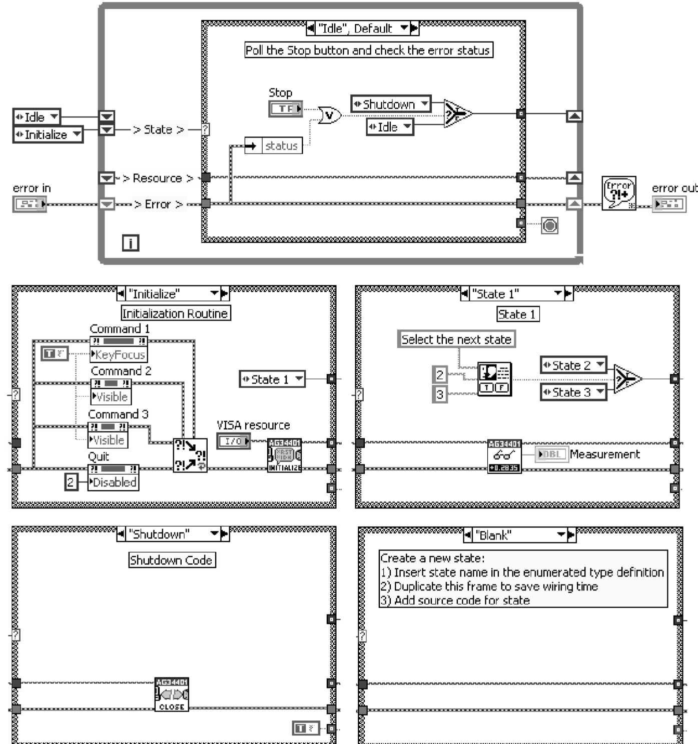


Рис. 8.9б. Улучшено использование свободного места путем добавления состояний, реализованное через случаи ***Initialize*** (Инициализация), ***Idle*** (Бездействие), ***Shutdown*** (Выключение) и ***Blank*** (Пустое) структуры Case. Двухэлементный сдвиговый регистр гарантирует, что случай ***Idle*** будет чередоваться с другими

Теперь давайте обсудим несколько популярных реализаций шаблона конечного автомата.

8.2.1. Классический конечный автомат

Блок-диаграммы на рис. 8.8 и 8.9 являются примерами *классического конечного автомата*, который использует сдвиговые регистры для передачи состояний. Активность графического интерфейса, состояние одной из логических кнопок, отслеживается опросом терминалов в состоянии **Idle**. Схема классического конечного автомата подходит для ВПП и процедур низкой и средней сложности. Например, эта схема – более гибкая и функциональная альтернатива структуре последовательностей (Sequence) для выполнения серий операций. Операции реализуются как состояния конечного автомата, а не как кадры структуры последовательностей. В отличие от этой структуры, в схеме классического конечного автомата можно на ходу менять порядок выполнения операций или вообще остановить последовательность до завершения. Кроме того, использование перечня как селектора гарантирует понятную маркировку случаев. Наконец, сдвиговые регистры в цикле While дают возможность обмена данными между состояниями. Пример классического конечного автомата реализован в главе 4 как Flexible Sequencer.

Схема классического конечного автомата является более гибкой и организованной альтернативой шаблону Continuous Loop. На рис. 8.10 представлено две реализации процедуры сбора данных приложения ВП Torque Hysteresis. Эта процедура производит конечное измерение момента вращения в зависимости от угла, используя ВП DAQmx, и записывает каждый образец в общую переменную. На рис. 8.10а представлена реализация этой процедуры через шаблон Continuous Loop, с ВП инициализации слева и ВП завершения работы справа. Блок-диаграмма на рис. 8.10б обладает той же функциональностью, но выполнена по шаблону классического конечного автомата. Эта реализация требует меньше места по горизонтали, содержит более полную документацию и обладает большими возможностями расширения по сравнению с шаблоном Continuous Loop, так как вы можете просто добавить новые состояния внутри случаев структуры Case, не добавляя новых элементов по горизонтали.

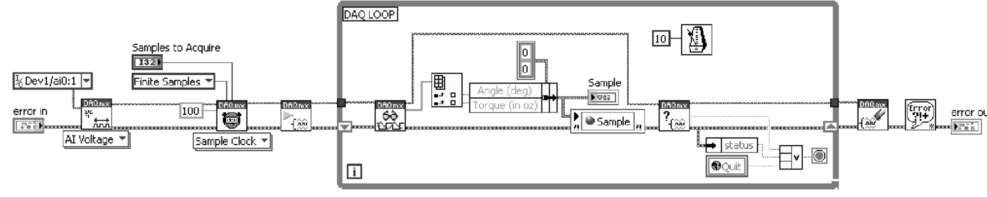


Рис. 8.10а. Процедура DAQ приложения Torque Hysteresis, выполненная по шаблону Continuous Loop. Код инициализации слева, непрерывное считывание внутри цикла, код завершения справа

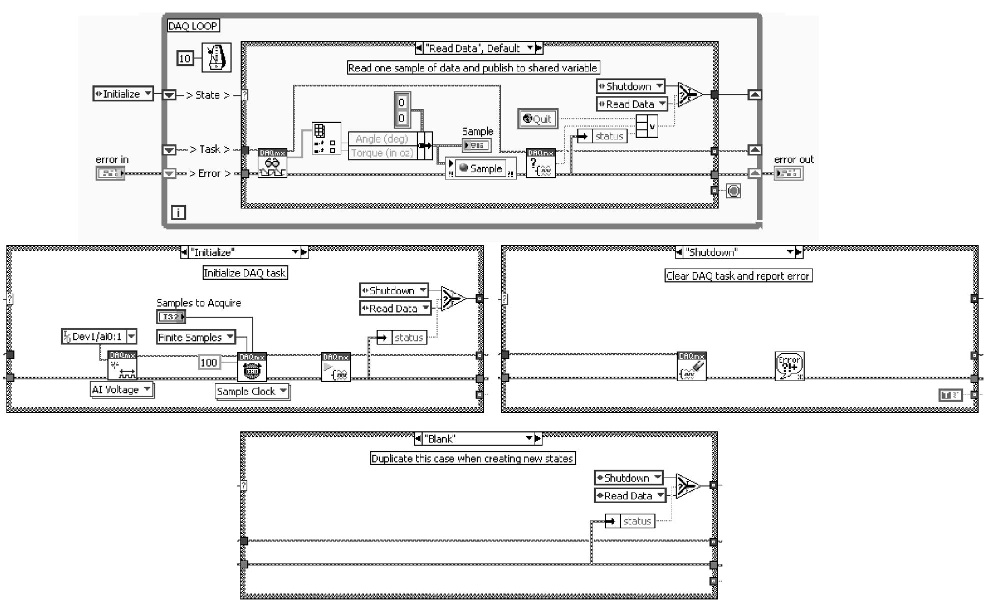


Рис. 8.10б. Реализация классического конечного автомата содержит состояния Инициализация, Бездействие, Выключение и Пустое. Эта реализация требует меньше места по горизонтали, содержит метки состояния и обладает большими возможностями расширения по сравнению с Continuous Loop

На рис. 8.11 представлен образец шаблона классического конечного автомата. Помимо инициализации значений и свойств элементов управления, используйте случай **Initialize** для инициализации устройств, считывания параметров конфигурации из файлов и всего остального, что нужно для установки приложения в желаемое начальное состояние. Используйте состояние **Shutdown**, чтобы установить оборудование в известное состояние, прервать связь с устройством, остановить сбор данных или файл-сессию. Случай **Idle** обычно используется для опроса устройств или контрольных управляющих элементов. В случае **Blank** все проводники просто распространяются от входного туннеля до выходного. Скопируйте состояние **Blank**, чтобы уменьшить себе работу при добавлении новых состояний.

Классический конечный автомат не подходит для сложных ВП, ВП верхнего уровня и ВП с графическим интерфейсом. Альтернативные реализации идеи конечного автомата используют очереди и структуры событий, что делает их более функциональными и эффективными в таких приложениях.

8.2.2. Конечный автомат с очередью

Шаблон классического конечного автомата с методом передачи состояний через сдвиговый регистр ограничен одним новым состоянием на конкретную итерацию цикла или состояние системы. В ВП среднего и высокого уровня сложности зача-

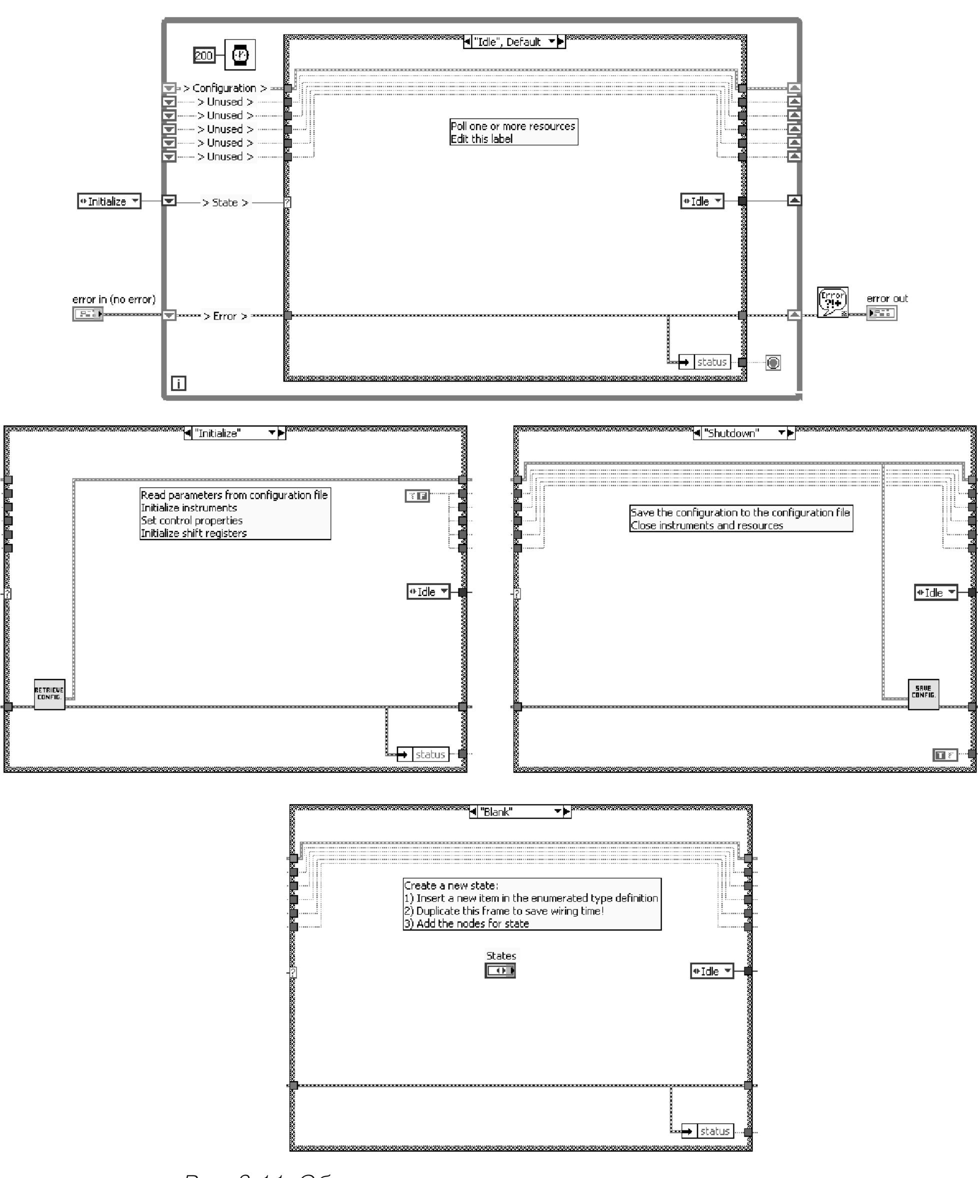


Рис. 8.11. Образец классического конечного автомата, содержащий состояния Инициализация, Бездействие, Выключение и Пустое

стую необходимо записать в буфер множество состояний, которые образуют последовательность и затем поочередно выполняются. Схема *конечного автомата с очередью* использует очереди для записи многих состояний в буфер. Каждое состояние системы может добавить любое число новых состояний в конец очереди, используя функцию Enqueue Element. Состояния по одному удаляются из очере-

ди и передаются на терминал селектора функцией Dequeue Element. Эта схема напоминает схему FIFO («первым вошел, первым вышел»). Дополнительно вы можете использовать функцию Enqueue Element At Opposite End, чтобы добавить состояние в начало очереди, тогда оно будет немедленно выполнено. Данная схема позволит приложению немедленно отвечать на действия или события с высоким приоритетом, например выполнять состояние **Shutdown** (Выключение), когда пользователь закрывает приложение.

Образец шаблона конечного автомата с очередью представлен на рис. 8.12.

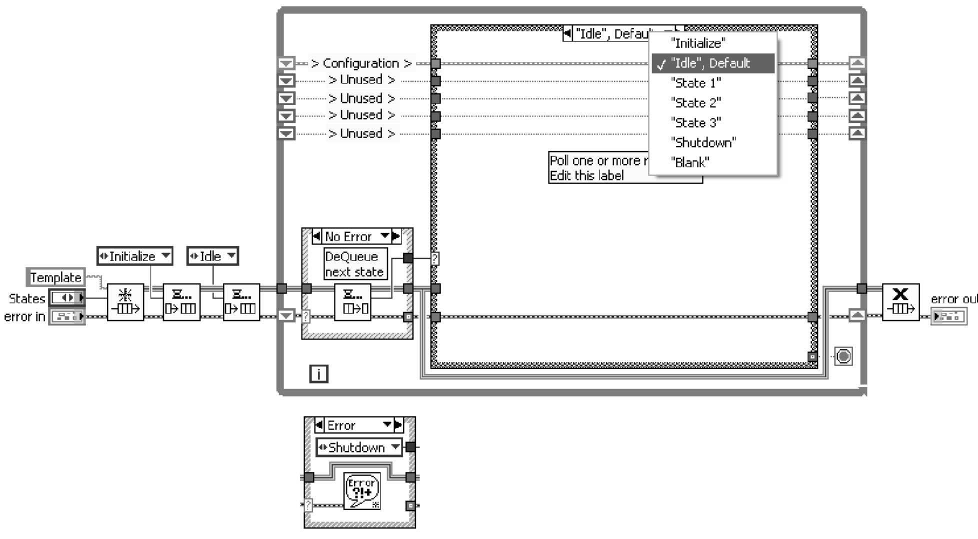


Рис. 8.12. Образец шаблона конечного автомата с очередью, который поддерживает накопление многих состояний в буфере

Очередь реализуется следующим образом: функции Obtain Queue и Enqueue Element инициализируют очередь слева от цикла While. Управляющий элемент пронумерованного типа данных (перечень, сохраненный как тайпдеф, далее перечень-тайпдеф) соединен с терминалом типа данных функции Obtain Queue. Этим определяется тип данных в очереди. Пронумерованные константы создаются из перечня-тайпдефа и соединены с терминалами элементов функции Enqueue Element. Инициализация (**Initialize**) и Бездействие (**Idle**) – первые два элемента в очереди. Это первые два состояния, выполняемые конечным автоматом. Функция Dequeue Element расположена внутри случая **No Error** структуры Error Case, вне основной структуры Case. Если в кластере ошибок нет ошибок, то следующее состояние удаляется из очереди и передается на терминал селектора основной структуры Case. Если возникает ошибка, то с помощью General Error Handler ВП создается отчет о ней и выполняется состояние **Shutdown** (Выключение). Внутри каждого цикла структуры Case в очередь добавляются дополнительные состояния через функцию Enqueue Element.

Другой вариант конечного автомата с очередью использует в качестве типа данных в очереди кластер – перечень-тайпдеф, объединенный с условным типом данных (variant). Перечень, как обычно, содержит желаемые состояния для селектора структуры Case. Условный тип данных используется для передачи данных из одного состояния в другое, только вместо сдвиговых регистров используются очереди. Например, на рис. 8.13 в состоянии **Acquire Waveform** происходит сбор данных волнового сигнала, которые конвертируются в условный тип данных, объединяются с константой типа перечень (выбранное состояние – **Update GUI**), полученный кластер добавляется в очередь. На следующей итерации цикла функция Dequeue Element удаляет кластер из очереди и разделяет элементы **State** и **Data**. Состояние **Update GUI** конвертирует условный тип данных обратно в волновой сигнал и обновляет соответствующий индикатор. Использование условного типа данных увеличивает гибкость данных, передаваемых между состояниями. Этот подход требует меньше проводников данных, чем использование сдвиговых регистров для обмена данными. Однако и данные хранятся в очереди лишь временно и к ним нельзя получить доступ в последующих состояниях – после того, как их удалили из очереди. Для сравнения, сдвиговые регистры сохраняют данные и обеспечивают доступ к ним из всех состояний и используют провод-

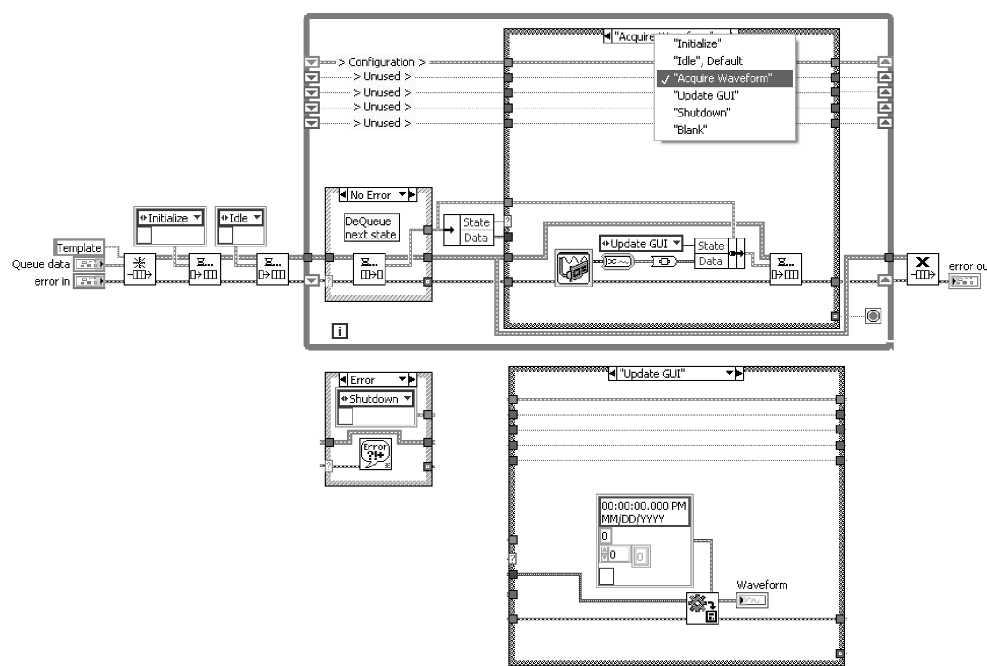


Рис. 8.13. Использование очередей для передачи данных между состояниями через кластер, содержащий условный тип данных и перечень. Это уменьшает количество проводников данных и сдвиговых регистров, но данные становятся недоступными из других состояний – после того как будут удалены из очереди

ники данных наряду с концепцией потока данных. Поэтому обычно сдвиговые регистры предпочтительнее очередей при осуществлении обмена данными между состояниями конечного автомата.

Правило 8.15 Избегайте пользоваться терминалом timeout с функций Enqueue Element и Dequeue Element

Управление через события в конечном автомате с чередями достигается игнорированием терминала **timeout** функциями Enqueue Element и Dequeue Element. Когда очередь пустая и нет активных состояний, цикл засыпает, выполняется код на заднем фоне. Как только в очереди появляется элемент, то немедленно просыпается функция Dequeue Element, удаляет элемент из очереди и запускается соответствующий код. Такая схема максимально эффективна и является основным преимуществом этого шаблона по сравнению классическим конечным автоматом, непрерывным циклом и др. Однако подсоединение терминала **timeout** уничтожает это преимущество. Поэтому во всех примерах в этой главе мы избегали терминала **timeout**.

Шаблон конечного автомата с очередями хорошо подходит для непрерывных процедур и ВПП среднего и более высокого уровня сложности. Аналогично классическому конечному автомату, шаблон конечного автомата с очередями не подходит для ВП верхнего уровня и ВП с графическим интерфейсом, если его не сочетать со структурой событий или отдельным циклом обработки ошибок. Эти шаблоны обсуждаются в разделах 8.2.3 «Событийно управляемый конечный автомат» и 8.3.1 «Параллельные циклы» соответственно.

8.2.3. Событийно управляемый конечный автомат

Как уже отмечалось в разделе 8.1.4 «Цикл с обработкой событий», структура событий является наилучшим методом отслеживания событий графического интерфейса, потому что управляется через события. Кроме того, конечный автомат с очередью является универсальным методом реализации схемы конечного автомата для процедур среднего и выше уровня сложности. *Шаблон событийно управляемого конечного автомата* объединяет конечный автомат с очередями и структуру событий в гибридный шаблон с одним единственным циклом, а именно, структура событий помещается в случай **Idle** (Бездействие) конечного автомата, что позволяет более эффективно обрабатывать события графического интерфейса. Таким образом, один цикл осуществляет обработку событий и состояний в буфере. Это мощная конструкция в компактной форме. Событийно управляемый конечный автомат подходит для ВП верхнего уровня и ВП с графическим интерфейсом среднего уровня сложности. Это мой любимый шаблон с одним циклом.

На рис. 8.14 ВП Torque Hysteresis выполнена по шаблону событийно управляемого конечного автомата. Лицевая панель на рис. 8.14а содержит меню логических элементов управления и индикатор графика. Блок-диаграмма на рис. 8.14б

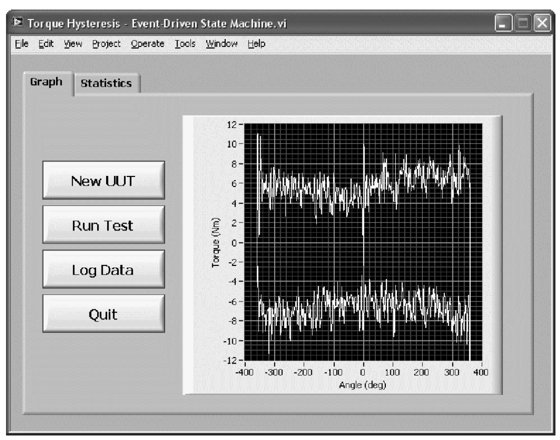
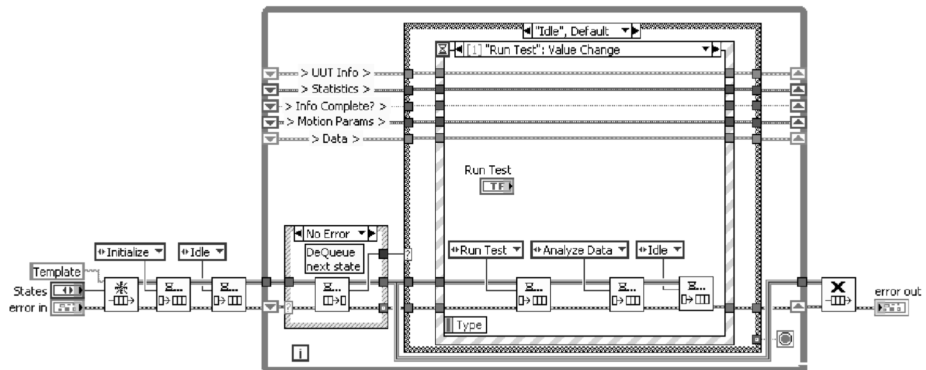


Рис. 8.14а. Лицевая панель ВП Torque Hysteresis содержит логические элементы управления и индикатор графика



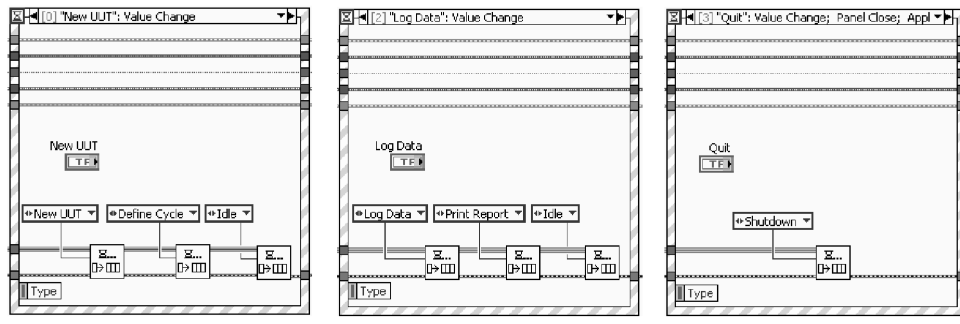


Рис. 8.14б. Блок-диаграмма ВП Torque Hysteresis, выполненная по шаблону событийно управляемого конечного автомата, состоит из конечного автомата с очередями и структуры событий в состоянии бездействия (**Idle**). Структура событий добавляет в буфер одно или несколько состояний для события Value Change для каждого логического элемента управления

состоит из событийно управляемого конечного автомата. Структура событий содержит случаи для событий Value Change всех логических элементов управления. Каждый случай события добавляет в очередь одно или несколько состояний. Эскиз состояния показан на рис. 8.14в. Каждая основная задача в приложении оформлена как состояние, кроме того, есть состояния Инициализация, Бездействие, Выключение и Пустое состояние. В целом такая реализация событийно управляемого конечного автомата является гибкой, функциональной, эффективной, организованной и компактной.

Событийно управляемый конечный автомат является наиболее мощным шаблоном из рассмотренных нами. Однако все состояния выполняются в цикле последовательно. Это означает, что любое состояние, на выполнение которого тре-

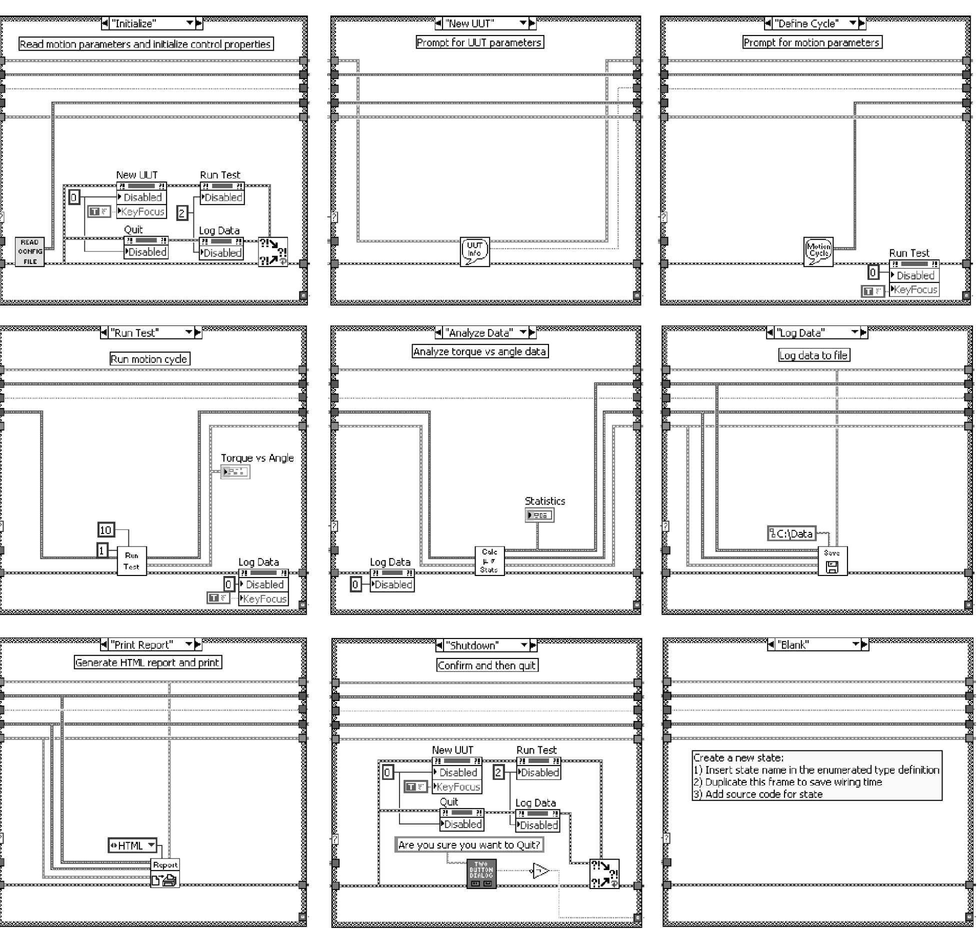


Рис. 8.14в. Отдельные состояния для каждой из основных задач приложения в дополнение к состояниям ***Initialize*** *(Инициализация),* ***Idle*** *(Бездействие),* ***Shutdown*** *(Выключение) и* ***Blank*** *(Пустое)*

буется много времени, будет тормозить приложение, так как состояние с длительными измерениями или тестами не будет давать возможность приложению выполнять другие задачи, в том числе состояние **Idle** (Бездействие), то есть приложение не будет реагировать на события графического интерфейса. Если в вашем приложении есть длительные или параллельные процедуры, тогда переходите к разделу 8.3 «Смешанные шаблоны». Событийно управляемый конечный автомат можно использовать для большинства ВП верхнего уровня, в приложениях, не содержащих длительных процедур или параллельных циклов.

8.2.4. Автомат событий

Использование структуры событий не следует ограничивать только событиями пользовательского интерфейса. Вместо этого вы можете сконфигурировать собственные события – пользовательские события (**user events**), которые создаются, регистрируются и запускаются программно функциями с палитры **Events** путем выбора пунктов меню **Programming** ⇒ **Dialog & User Interface** ⇒ **Events**. *Шаблон автомата событий* сочетает в себе функциональность структуры событий и структуры событий шаблона событийно управляемого конечного автомата. Так возникает новый шаблон, состоящий из структуры событий, расположенной внутри цикла While. Для каждого состояния задаются пользовательские события, случаи событий настраиваются для всех комбинаций пользовательских событий и событий графического интерфейса. Структура событий обрабатывает пользовательские события по аналогии с событиями графического интерфейса. Все зарегистрированные события (пользовательские и графического интерфейса) по мере появления пишутся в буфер по принципу «первым вошел, первым вышел» и обрабатываются в соответствующих случаях структуры событий. На рис. 8.15 представлен ВП Torque Hysteresis, выполненный на основе шаблона автомата событий.

Пользовательские события созданы и зарегистрированы для каждого состояния ВП Torque Hysteresis с помощью ВП Create & Register User Events (на рис. 8.15 – крайние слева). Блок-диаграмма и лицевая панель ВПП показаны на рис. 8.16. **User Event Data** – это кластер управляющих элементов условного типа данных, в котором каждому элементу соответствует состояние приложения. Элементы маркированы соответствующим образом. Этот кластер сохранен как тайпдеф и является аналогом перечня в обычном конечном автомате, с той разницей, что тип данных каждого элемента кластера определяет тип данных, используемых для передачи данных в событие. Данные событий полезны, если процедура запуска события требует передачи данных в событие. В частности, код, создающий событие, может находиться в местах, где нет возможности использовать проводники данных. Реализация на рис. 8.15 использует сдвиговые регистры для передачи данных между случаями событий, по аналогии с ранее рассмотренными шаблонами. Сдвиговые регистры дают возможность сделать данные доступными во всех случаях событий, а не только для процедуры создания и события, и соответствующего случая события. Однако сдвиговые регистры не могут быть использованы, если процедура запуска события находится вне цикла. Поэтому я ре-

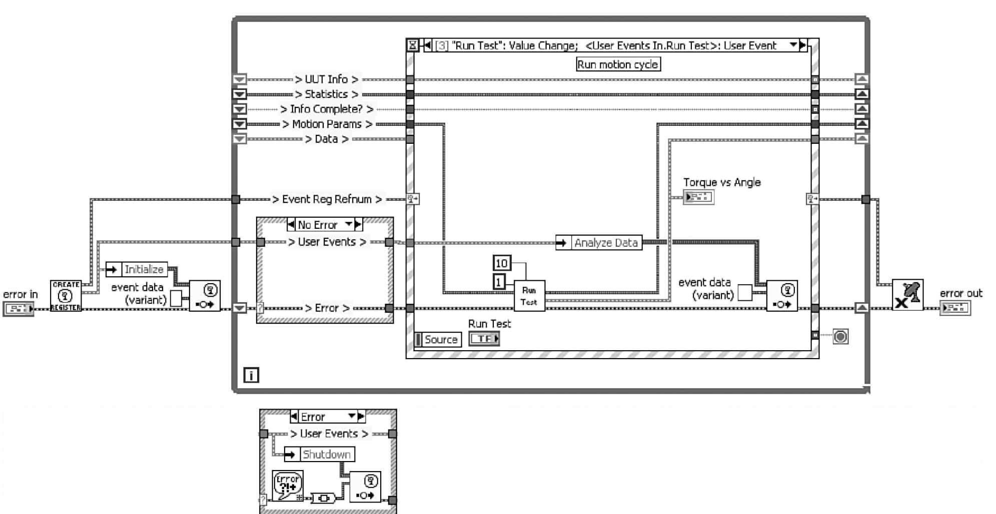
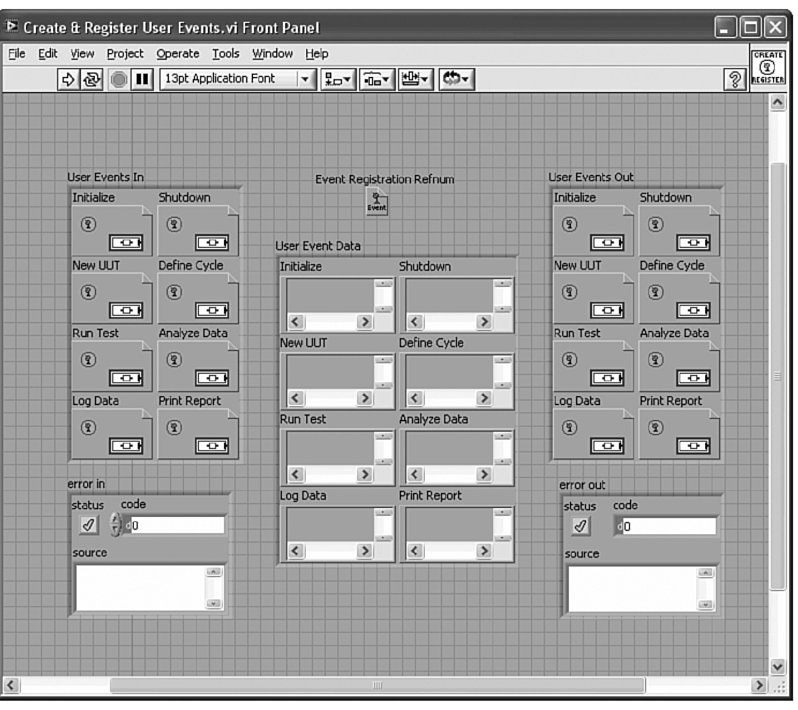


Рис. 8.15. ВП Torque Hysteresis, выполненный на основе шаблона автомата событий. Пользовательские события используются для объединения функциональности структур случаев (Case) и событий (Event)

комендую использовать сдвиговые регистры, когда данные события могут быть переданы по проводникам данных.

Как показано на блок-диаграмме (см. рис. 8.16), элементы кластера **User Event Data** разъединяются и передаются на входной терминал **user event data type** функции Create User Event. Эта функция определяет имя и тип данных для каждого пользовательского события, основанного на метках и типе данных, подаваемых на входной терминал. Create User Event возвращает ссылку – reference number (дословно – опорный номер), которая соединяется с кластером **User Events Out**. Эти операции повторяются в цикле For, пользовательское событие создается в каждом состоянии приложения. Эти события регистрируются путем передачи кластера **User Events Out** на терминал **event source** функции Register Events. Эта функция, в свою очередь, возвращает **event registration refnum** (номер зарегистрированного события), который возвращает ВПП и который соединен с терминалом **dynamic event** структуры событий, как показано на рис. 8.15. Наконец, пользовательские события запускаются в любом месте приложения через функцию Generate User Event. Ссылка на пользовательское событие, соответствующее желаемому состоянию, от кластера **User Events Out** передается на соответствующий вход функции Generate User Event вместе с данными события. В случае события **Run Test** запускается событие **Analyze Data**, на терминал **event data** подается пустая константа универсального типа данных.

Шаблон автомата с событиями похож на событийно управляемый конечный автомат, но лучше использует структуру событий. Основное преимущество состоит в том, что множество структур событий можно настроить на одно и то же



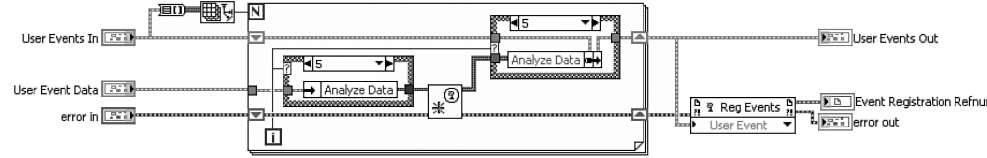


Рис. 8.16. ВП Create & Register User Events используется для создания и регистрации множества пользовательских событий. Работа ВП основана на метках и типах данных элементов кластера *User Event Data*

пользовательское событие, тогда как каждая очередь имеет уникальную ссылку для добавления и удаления элементов очереди. Эта особенность дает возможность иметь множество автоматов с событиями в одном приложении и все они могут отвечать на одну общую команду. Однако есть несколько ограничений в схеме автомата с событиями. Во-первых, техподдержка сложнее, чем в случае событийно управляемого конечного автомата. Для того чтобы добавить или удалить состояние приложения, нужно изменять как кластер данных пользовательских событий, так и кластер ссылок на пользовательские события. Необходимо также удалить или добавить соответствующую функцию Create User Event, что потребует регистрации нового события. Помимо этого, нужно будет внести изменения в конфигурацию структуры событий, которая в плане поддержки похожа на пронумерованную структуру случаев. Во-вторых, буфер автомата с событиями

менее функционален по сравнению с очередями в событийно управляемом конечном автомате. Используя очереди, вы можете предварительно просмотреть, убрать из очереди или вообще сбросить события в очереди без выполнения соответствующего элемента кода. Кроме того, вы можете добавить новые элементы как в конец очереди, так и в начало. Я использую эту особенность, чтобы добавить состояние **Shutdown** в начало очереди, как только требуется немедленно завершить работу с приложением. Автомат с событиями записывает события в буфер по принципу «первым вошел, первым вышел», и все события в буфере выполняются до тех пор, пока цикл While не остановится. Поэтому я рекомендую автомат с событиями только для сложных приложений с графическим интерфейсом, когда необходимо множество структур событий.

Наконец, общее ограничение для всех шаблонов конечных автоматов, которые мы успели обсудить, состоит в том, что каждый из них может обрабатывать только одно состояние приложения за раз. Это приводит к тому, что все состояния, содержащие блок-диаграммы, которые требуют значительного времени выполнения, не дают приложению быстро отвечать на другие события или состояния. К примеру, графический интерфейс может оказаться невосприимчивым к действиям пользователя во время длительного выполнения кода другого состояния. Эти ограничения можно преодолеть, используя составные шаблоны, которые мы обсудим далее.

8.3. Составные шаблоны

Все шаблоны, которые мы обсуждали ранее, имели ограничения при использовании их в качестве основания для больших приложений. Шаблоны ВПП Immediate и Functional globe предназначены только для ВПП. Шаблон непрерывного цикла трудно расширять. Цикл с обработкой ошибок и классический конечный автомат легко расширить добавлением поддерживаемых случаев, но они не могут выполнять функции остальных. Событийно управляемый конечный автомат и автомат с событиями сочетают поддержку событий графического интерфейса и передачу состояний, но их производительность ограничена последовательным выполнением одного цикла. Вместо этого представьте себе, что каждый шаблон с одним циклом является кирпичиком, из которых строится большое приложение. Большинство сложных приложений лучше всего реализовывать, используя комбинацию многих шаблонов с одним циклом, при этом вы выигрываете на каждом во время оптимизации и в конечной производительности.

Составные шаблоны создаются с использованием комбинации двух или более шаблонов с одним циклом и структуры сообщения для передачи данных между ними. Схема сообщения добавляет сложности. Поэтому понадобятся хорошие инструменты разработки, примеры и образцы. Разрабатывайте приложения, используя составные шаблоны или образцы, или объединяя шаблоны и образцы с единственным циклом через структуру обмена сообщениями. Удобно и логично иметь все шаблоны с одним циклом на палитре **Functions** для быстрого доступа, по аналогии с тем, как на палитре **Structures** находятся все имеющиеся структуры. Это можно осуществить, изменив внешний вид палитры **Structures**, добавив на нее иконки для каждого шаблона с одним циклом. Настройте иконки вложенного меню так, чтобы объединять содержание соответствующего шаблона с целевым ВП, а не просто размещать иконку как вызов ВПП. Тогда разработка составных шаблонов станет такой же легкой, как простое перетаскивание объектов на блок-диаграмму.

Настроенный вид палитры показан на рис. 8.17а и содержит иконки, представляющие четыре шаблона с одним циклом. Рисунок 8.17б содержит контур шаблона во время перетаскивания и объединения его с блок-диаграммой.

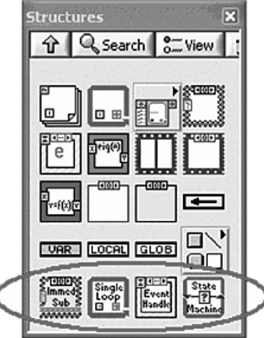


Рис. 8.17а. Настроенный вид палитры функций содержит иконки, представляющие четыре шаблона с одним циклом. Иконки настроены так, чтобы объединять соответствующий шаблон с выбранной блок-диаграммой

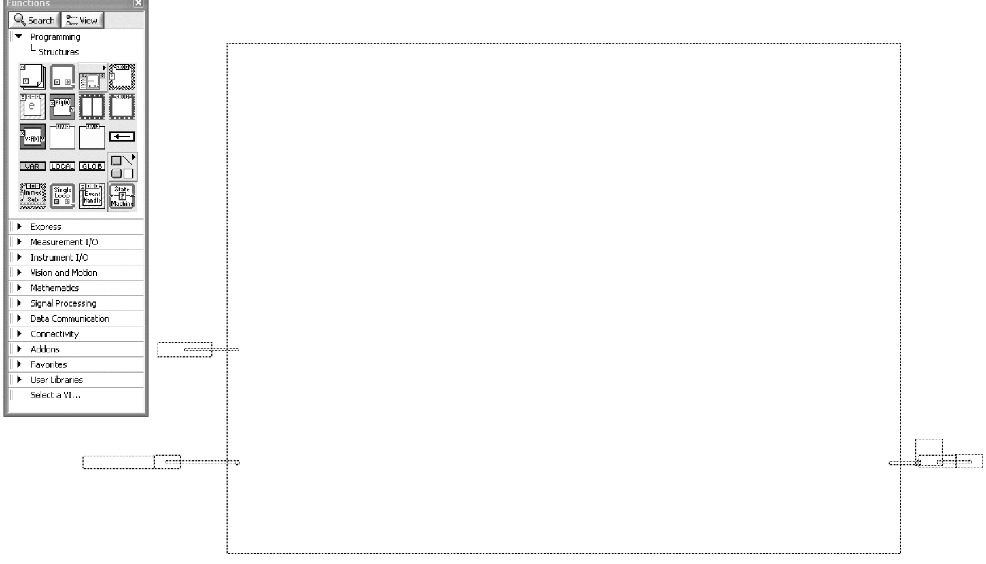


Рис. 8.17б. Щелкните, перетащите и отпустите – и выбранный вами с палитры функций шаблон будет встроен в блок-диаграмму

8.3.1. Параллельные циклы

Параллельные циклы необходимы в приложениях, где есть много непрерывных задач, которые должны выполняться параллельно. Например, у вас может быть приложение, собирающее данные и записывающее их в файл. Параллельно пользователю необходимо видеть и обрабатывать данные. Однако отображение и обработка данных не могут интерферировать со сбором и записью данных. Следовательно, вы не можете поместить обе задачи в разные случаи цикла с обработкой событий или конечного автомата. Использование любого шаблона с единственным циклом, любого ВПП, события или состояния временно приостановит цикл, не давая выполниться другим ВПП, событиям или состояниям до тех пор, пока не завершится выполнение кода. Вместо этого лучше всего разделить задачи, которые должны выполняться одновременно, и использовать два параллельных цикла. Используйте непрерывный цикл только для непрерывного сбора данных и записи в файл, так как он выполняется без прерываний со стороны пользовательского интерфейса. Поддержку событий пользовательского интерфейса и отображение данных следует организовать через цикл с обработкой событий. Создайте схему обмена данными между циклами.

Правило 8.16 Используйте очереди, общие переменные или RT FIFO для обмена данными между параллельными циклами

Обмен данными между циклами можно осуществить множеством различных способов, включая локальные и глобальные переменные, функциональные глобальные переменные, уведомления, очереди пользовательских событий, разделяемые переменные и буферы реального времени RT FIFO. Локальные и глобальные переменные являются самым простым, но самым не желательным, с точки зрения производительности и стиля, способом. Как уже обсуждалось в главе 4, локальные и глобальные переменные нарушают концепцию потока данных в LabVIEW. Функциональные глобальные переменные обладают большей гибкостью и лучше производительностью, но требуется некоторое время и усилия чтобы их создать. Уведомления эффективны, но могут хранить и передавать только одно сообщение за раз. Пользовательские события хранят данные события в буфере, но их сложнее настраивать и поддерживать, и буфер данных менее эффективен по сравнению с очередью. Я рекомендую использовать очереди или разделяемые переменные для настольных приложений, а RT FIFO – для приложений в реальном времени, требующих программной конфигурации.

Очереди и разделяемые переменные являются встроенными конструкциями LabVIEW, эволюционировавшими во времени. Очереди доступны с палитры **Synchronization**, а разделяемые переменные настраиваются в окне **Project Explorer**. Вы можете настроить очереди и разделяемые переменные так, чтобы хранить больше одного сообщения или элемента одновременно. Это важно, если циклы работают с разной частотой и вы не хотите потерять какие-либо данные, или если

приложение передает данные кусками, а не по одному элементу. Кроме того, используя очереди и разделяемые переменные, можно не только обмениваться данными между циклами, но и синхронизовать эти циклы. Размер очередей и RT FIFO задается программно. Размер разделяемых переменных задается во время редактирования, но они могут передавать данные по сети и включают временную отметку. Даже если эти особенности для вас сейчас не важны, они обеспечивают общую совместимость в случае, если в будущем они вам понадобятся.

Правило. 8.17 Расставьте приоритеты циклов, используя задержки или свойства потоков

Параллельные циклы очень важны для высокопроизводительных приложений, в том числе для приложений, которые должны работать в режиме реального времени. Приоритет каждого цикла можно задать, используя задержки, свойство Timed Loop Priority или свойства выполнения ВПП. Самый простой путь – встроить в каждый цикл While задержку, используя Wait (ms), Wait Until Next ms Multiple или ВП Time Delay Express. Цикл с наименьшей задержкой получит наивысший приоритет. Как вариант можно выделить каждый цикл в ВПП и настроить свойство Priority and Preferred Execution System в меню **File ⇒ VI Properties»Execution**. Эта техника включает настройку потоков, так что рекомендуется иметь базовые знания о работе со многими потоками. Свойство Priority цикла Timed Loop можно настроить через входной терминал **priority** (2). Множество приложений, работающих в режиме реального времени, обладают жесткими временными рамками, в этом случае выигрывают описанные особенности схемы Timed Loop. Для указания приоритетов в настольных приложениях используйте задержки для максимальной простоты. Для приложений, работающих в режиме реального времени, используйте свойство Priority цикла Timed Loop, для точности и лучшей производительности.

Правило 8.18 Ширину параллельных циклов установите одинаковой и выровняйте по вертикали

Правило 8.19 Минимизируйте свободное место между циклами

Правило 8.20 Маркируйте каждый цикл в левом верхнем углу

Вот несколько правил, которые обеспечивают опрятные, организованные блок-диаграмм с параллельными циклами: Ширину параллельных циклов установите одинаковой, выровняйте их по вертикали и минимизируйте свободное место между циклами. В главе 4 мы установили ограничение на размер блок-диаграмм – один экран в разрешении 1280×1024. Если блок-диаграмма содержит множество параллельных циклов, то правило одного экрана невыполнимо, если

циклы не превращены в ВПП. Если вы используете параллельные циклы на одной и той же блок-диаграмме, убедитесь, что вы минимизировали пустое место между циклами в попытках сделать видимую на экране часть блок-диаграммы максимальной. Наконец, разместите свободную метку в левом верхнем углу каждого цикла, которая бы указывала на основную задачу, выполняемую этим циклом.

Рассмотрим в качестве примера ВП Torque Hysteresis. Основная задача – это сбор, анализ и представление данных зависимости момента вращения от угла. В реализациях с использованием одного цикла, которые мы рассматривали в предыдущих главах, цикл движения во время сбора данных (состояние Run Test) контролировался через процедуру сбора данных внутри ВП Run Test. Когда вызывается этот ВПП, то открывается его лицевая панель, и график зависимости момента вращения от угла отображается на панели во время всего цикла движения. Когда тест заканчивается, данные записываются в файл и создается отчет. Однако хотелось бы получать и отображать данные зависимости момента вращения от угла непрерывно, во всех состояниях приложения. Цифровые индикаторы момента вращения и угла на лицевой панели ВП верхнего уровня (рис. 8.18а) позволяют пользователю в реальном времени отслеживать эти важные параметры без запуска теста. Это помогает оператору настроить измерительные приборы и тестируемый образец, вручную создать вращение и увидеть реакцию, подтвердить отсутствие силы в целях безопасности.

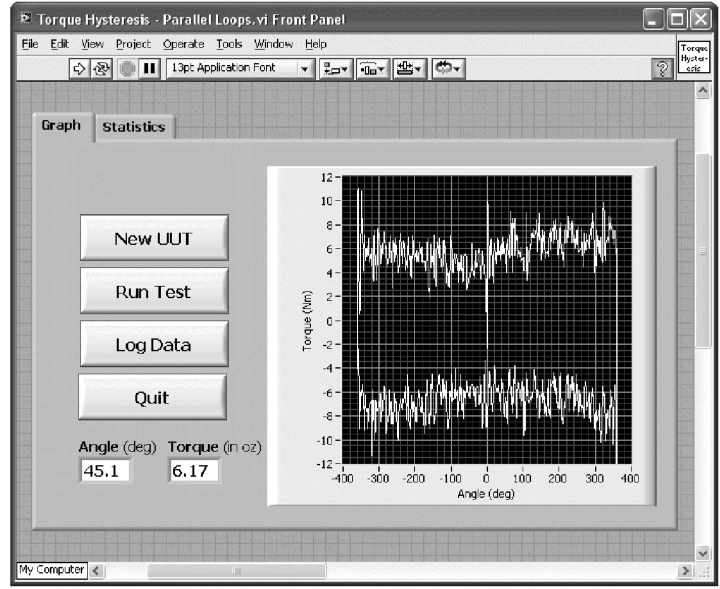


Рис. 8.18а. Лицевая панель ВП Torque Hysteresis содержит цифровые индикаторы, отображающие данные в реальном времени. Это облегчает непрерывное наблюдение за этими важными параметрами (угол и момент вращения)

Непрерывный сбор данных и отображение осуществляется добавлением второго непрерывного цикла на блок-диаграмму верхнего уровня, расположенного рядом с событийно управляемым конечным автоматом, как показано на рис. 8.18б. Процедура сбора данных взята из ВП Run Test и работает в параллельном цикле. Это обеспечивает непрерывность операций и предотвращает интерференцию событий пользовательского интерфейса и процедуры сбора и отображения данных. Данные передаются между циклами через общую переменную. Цикл DAQ включает задержку в 10 миллисекунд и имеет приоритет выше, чем цикл Main Loop, который содержит задержку в 100 миллисекунд.

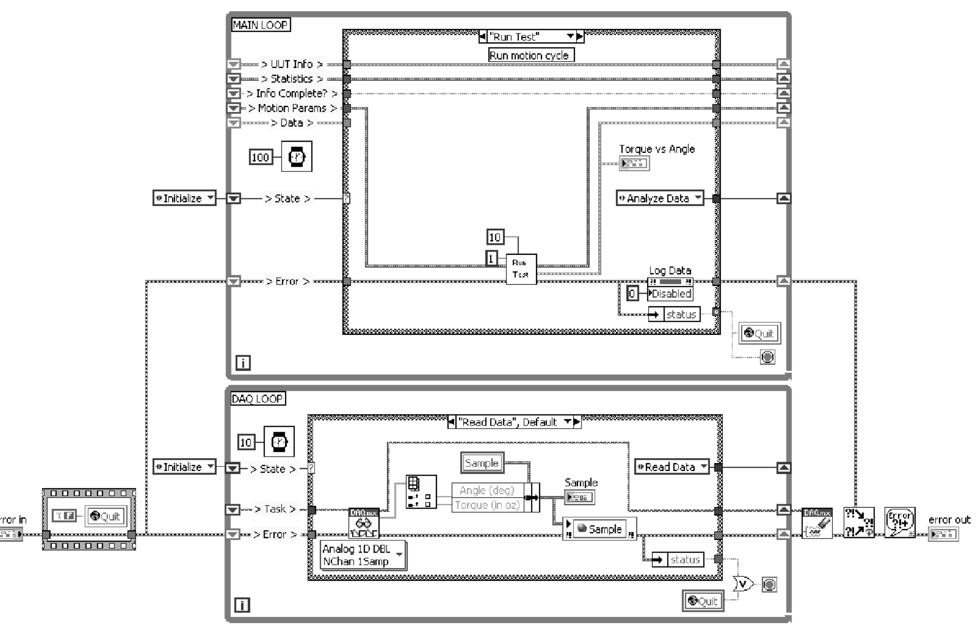


Рис. 8.18б. Сбор данных и передача состояния осуществляются в параллельных асинхронных циклах. Цикл DAQ осуществляет непрерывный сбор данных вне зависимости от цикла Main

Приложение, которое использует параллельные циклы, легко создать, объединив два или более шаблона с одним циклом на блок-диаграмме и добавив структуру обмена данными, использующую очереди, общие переменные или RT FIFO. Или, как вариант, вы можете создать образец со многими циклами, сохранив ВП, с множеством шаблонов с одним циклом и структур обмена сообщениями как тип VIT и поместить в папку LabVIEW\Templates. Образцы ВП могут быть загружены через диалоговое окно **New (File ⇒ New)** или из меню быстрого доступа узла в окне **Project Explorer**.

8.4. Объектные структуры сложных приложений

Многие сложные приложения используют несколько общих функциональных элементов или процедур или устроены похожим образом. Например, большинство приложений, которые разрабатывал я, содержат процедуры сбора данных, управления устройствами, анализа, обработки событий пользовательского интерфейса, передачи состояний и обработки ошибок. У меня есть свои любимые шаблоны и утилиты, которые я постоянно использую. Вместо того чтобы разрабатывать каждое новое приложение, начиная с шаблона и утилит, у меня есть объектная структура приложения, которая объединяет наиболее часто используемые процедуры в некоторый образец, который и является отправной точкой при разработке новых приложений. *Объектная структура приложения* – это тщательно продуманный образец приложения, заполненный множеством структур, включая шаблоны, ВПП, структуры данных, схемы обмена сообщениями и т.д. Объектная структура служит детальным планом приложения.

Примером создания сырой объектной структуры может служить копирование всего исходного кода из предыдущего проекта и агрессивное редактирование его, так чтобы код удовлетворял требованиям нового приложения. Я отношусь к этому подходу, как к взлому, и настоятельно не рекомендую его использовать, потому что время, сэкономленное на старте, гораздо меньше времени, которое вы потратите на постоянный поиск, проверку, редактирование исходного кода. Более того, неважно, как тщательно вы проверите код, он наверняка будет содержать какие-то остаточные элементы старого приложения, которые не применимы в новом приложении. Эти остатки обычно проявляются наихудшим из возможных образов. Например, при финальной проверке приложения может выясниться, что в отчетах или на экране содержит название, адрес и логотип другой компании. Будет, мягко говоря, очень неловко! Кроме того, взломанный код обычно более запутанный, чем код, изначально созданный для какой-то конкретной цели. Техническая поддержка исходного кода, в общем случае, куда сложнее и требует больше времени, чем разработка сценария. Для лучшего стиля я рекомендую использовать объектные структуры приложений, которые можно использовать многократно, такие структуры используют несколько функциональных элементов, общих для тех приложений, с которыми вы сталкивались. Сделайте объектную структуру пригодной для многократного использования, характерной для данного класса приложений и невероятно гибкой.

В этом разделе представлены три объектные структуры, полезные в сложных приложениях. Важно отметить, что возможно множество вариантов этих структур. Приведены относительно простые реализации объектных структур с примерами и иллюстрациями – для облегчения обсуждения.

8.4.1. Динамическая объектная структура

Динамическая объектная структура использует функции ВП-сервера для того, чтобы динамически загружать и запускать высокоуровневые ВП из файлов источника на диске. Высокоуровневые ВП, составляющие приложение, называются

компонентами. ВП, которые динамически загружаются объектной структурой, независимо от уровня сложности, называются модулями . До того как в версии LabVIEW 8.0 была представлена опция ВП Call Configuration, динамическая объектная структура широко использовалась для улучшения времени загрузки и более эффективного использования памяти в очень больших приложениях. Таким же образом мы приходим к динамическому загрузчику (**dynamic loader**). Однако диалоговое окно ВП Call Configuration облегчает динамическую загрузку с оговоркой, что все компоненты должны быть жестко связаны с приложением, подобно ВПП. Диалоговое окно ВП Call Configuration можно открыть, выбрав **Call Setup** из меню быстрого доступа вызова ВПП. В настоящий момент динамическая объектная структура требуется в случаях, когда компоненты не связаны с приложением и не определены до момента запуска. Объектная структура считывает доступные компоненты из указанной директории на диске и загружает их в память по необходимости. Эта совместимость позволяет разрабатывать и распространять компоненты после того, как было установлено приложение, что облегчает техническую поддержку и обновление приложения.

Образец динамической объектной структуры показан на рис. 8.19. Лицевая панель состоит из структуры вкладок со страницами **Main** и **Error Info**. Страница **Main** содержит список, субпанель и логическую кнопку **Quit**. Список отображает

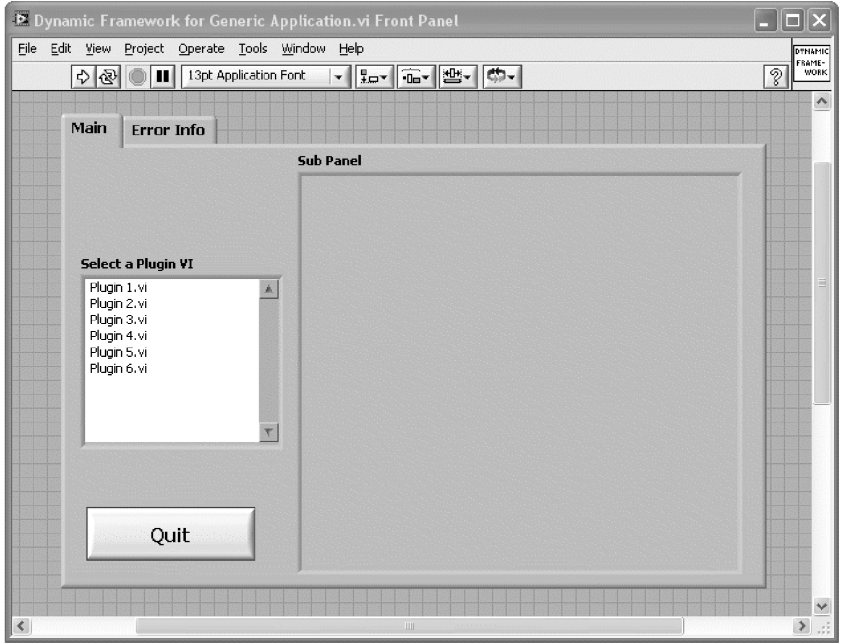
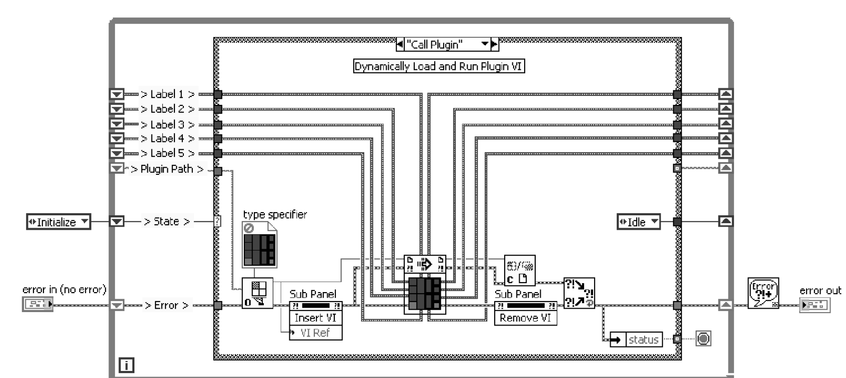


Рис. 8.19а. Лицевая панель образца динамической объектной структуры содержит список компонентов и субпанель, отображающую лицевую панель выбранного компонента



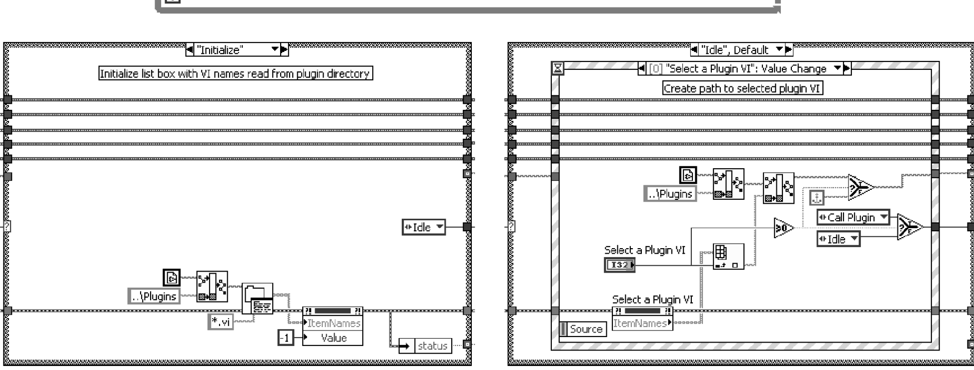


Рис. 8.19б. Блок-диаграмма содержит шаблон событийно управляемого конечного автомата без очереди, показано три основных состояния. Данные между компонентами передаются через сдвиговые регистры

компоненты, считанные с диска, а субпанель отображает лицевую панель выбранного компонента. Панель компонента рассматривается как дочерняя панель, когда загружена в субпанель. Эта дочерняя панель полностью интерактивна, пока компонент запущен.

Блок-диаграмма содержит шаблон событийно управляемого конечного автомата без очереди, показано три основных состояния: **Initialize** (Инициализация), **Idle** (Бездействие) и **Call Plagin** (Вызов модуля). В состоянии **Initialize** происходит считывание содержимого директории модулей с использованием функции List Folder и запись имен модулей в свойство Item Name управляющего элемента типа список. Состояние **Idle** просто ждет событий пользовательского интерфейса. Когда пользователь выбирает компонент из списка, происходит событие Value Change, создается указатель пути к выбранному компоненту, и следующим для запуска назначается состояние **Call Plugin**. В состоянии **Call Plugin** компонент загружается в память, для этого используется функция ВП Reference ВП-сервера Open, и в субпанели открывается лицевая панель выбранного компонента, используется метод ВП Insert. Данные передаются к терминалам компонента, и компонент выполняется через узел Call By Reference Node, по аналогии с вызовом ВПП. Когда компонент завершает работу, он выгружается из памяти через функцию Close Reference, и дочерняя панель закрывается, для этого используется метод ВП Remove.

Правило 8.21 Используйте Call By Reference Node вместо метода Run

Правило 8.22 Используйте стандартные назначения соединительных терминалов

Правило 8.23 Все компоненты храните в выделенной директории

Альтернативой узлу Call By Reference Node является узел Invoke Node (Узел запросов) с методами Set Control Value (установить значение), Run, Get control value (получить значение). Однако узел Call By Reference Node выполняется быстрее, требует меньше кода, чем множество узлов Invoke Node. Также Call By Reference Node имеет сходство с обычным вызовом ВПП, что упрощает поддержку и улучшает читабельность. Обратите внимание на то, что все компоненты должны использовать общую схему соединительной панели и одинаково назначенные терминалы, чтобы использовать Call By Reference Node. Также размещение компонентов в соответственно названной директории на диске поможет упростить поддержку и обновление. Все примеры в данной секции используют папку \Plugins.

Правило 8.24 Данные между компонентами передавайте через сдвиговые регистры

Разрабатывайте модули как независимые ВП, которые могут работать в любом порядке, без особой поддержки со стороны объектной структуры. Ограниченные данные можно передавать между компонентами, используя сдвиговые регистры. Ограничения обусловлены выбором схемы соединительной панели и присвоенными структурами данных. Требуется тщательная оценка структур данных для того, чтобы они были согласованы с остальными компонентами, включая возможные будущие дополнения. Это достигается через анализ требований к входным и выходным параметрам каждого компонента, группировкой близких данных в кластеры (как тайпдефы) и определением максимум четырех структур данных, которые сочетают известные требования к данным всех компонентов. Продублируйте эти структуры данных и как управляющие элементы, и как индикаторы на лицевой панели каждого компонента. Таким образом, каждый компонент будет иметь доступ для чтения и записи ко всем структурам данных, вне зависимости от того, используются они в этом компоненте или нет. На блок-диаграмме передайте структуры данных от элементов управления ко всем необходимым узлам, терминалам индикаторов. Все неиспользуемые структуры данных сразу от управляю-

щих элементов сразу подаются на терминалы индикаторов. Назначьте элементы управления и индикаторы на соединительной панели 4×2×2×4, в соответствии с правилами главы 5. Нижним левым и правым терминалам поставьте в соответствие кластеры ошибок **error in** и **error out**.

Правило 8.25 Один входной и выходной терминал назначьте как универсальный

Управляющие элементы и дублированные индикаторы четырех заданных структур, в сочетании со стандартным кластером ошибок, занимают 10 терминалов на соединительной панели. Таким образом, остается два свободных терминала, если использовать схему 4×2×2×4 соединительной панели. Однако при использовании модулей свободные терминалы бесполезны. Когда в объектной структуре настраивается определитель типа Open VI Reference, основанный на соединительной панели представленного модуля, назначения на соединительной панели всех компонентов должны в точности совпадать с определителем типа, то есть назначения терминалов не могут быть изменены после внедрения приложения. Так зачем ограничивать себя четырьмя структурами данных? Для того чтобы приспособиться к будущим неизвестным требованиям, назначьте одну дополнительную пару входных и дублированных выходных терминалов в качестве универсальных и, используя другой сдвиговый регистр, передайте условный тип данных между компонентами. Как уже обсуждалось в главе 6 «Структуры данных», универсальный тип данных – это обобщенный тип данных, который определяется динамически по время выполнения приложения. Реальные данные, содержащиеся в данных условного типа, определяются внутри компонентов. Такой подход увеличивает гибкость при дальнейшем расширении приложения.

Блок-диаграмма объектной структуры на рис. 8.19б содержит пять сдвиговых регистров для передачи данных между состояниями и один сдвиговый регистр – для передачи указателя пути к желаемому компоненту для динамической загрузки. Это позволяет всем компонентом создавать, передавать и изменять данные. Все структуры данных в шаблоне объектной структуры – универсального типа. Замените до четырех из них стандартными структурами данных, которые требуются для компонентов вашего приложения.

Правило 8.26 Отображайте лицевую панель модуля в субпанели

Субпанели облегчают тесную интеграцию панелей компонентных ВП в объектную структуру графического интерфейса пользователя. Однако в субпанели единовременно отображается только одна лицевая панель, несмотря на то, что можно открыть лицевые панели в отдельных окнах и расположить их согласно желанию пользователя. Поскольку динамическая объектная структура загружает, выполняет и выгружает только один компонент за раз, субпанели чуть предпочтительнее, чем отдельные окна.

Рассмотрим, как динамическая объектная структура применяется в ВП Torque Hysteresis. Все ранее рассмотренные нами реализации этого приложения состояли из модулей, каждое состояние приложения было выполнено как ВПП. Некоторые из них были диалогами, как ВП UUT Information Prompt и ВП Define Cycle. Диалоги несложные и не требуют много памяти. Другие ВПП, такие как ВП Run Test и ВП Compute Statistics, являются ВП верхнего уровня и содержат иерархию ВПП и большие структуры данных. Они являются высокоуровневыми компонентами приложения. Производительность приложения улучшается при динамической загрузке высокоуровневых компонент. Эту процедуру легко осуществить, используя диалоговое окно **VI Call Configuration** внутри любого шаблона, описанного ранее. На рис. 8.20 показан вызов конфигурации ВП Run Test. Поскольку ВП Run Test обычно выполняется много раз, отмечена опция **Loading and retaining on first call** (Загрузка и сохранение по первому вызову).

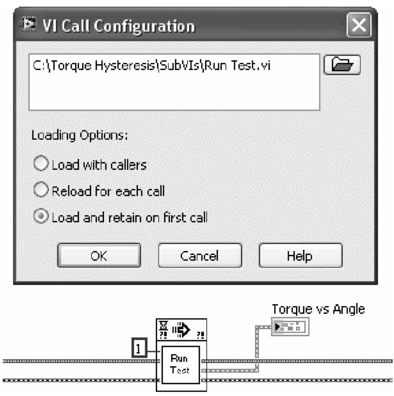


Рис. 8.20. Производительность ВП Torque Hysteresis улучшена динамической загрузкой основных компонентов, включая ВП Run Test. Это осуществляется чрез диалоговое окно VI Call Configuration

Вы можете улучшить долгосрочную поддержку и общую производительность ВП Torque Hysteresis, используя динамическую объектную структуру. На рис. 8.21 показана реализация ВП Torque Hysteresis, в которой все диалоги, включая высокоуровневые ВП, реализованы как компоненты. Тем самым улучшена гибкость в вопросах обновления. Кластер параметров движения объединен с различными параметрами для каждого компонента в новый кластер конфигурационных параметров. Эти параметры задаются в состоянии **Initialize** из файла конфигурации. Каждый компонент был изменен так, чтобы получать и возвращать все структуры данных, вне зависимости от того, нужны ли они в этом компоненте или нет. Кроме того, добавлена одна пара (вход/выход) терминалов универсального типа данных для будущего расширения. Эти улучшения увеличивают функциональность и гибкость каждого компонента и поддерживают стандартные назначения на соединительной панели для всех компонентов.

Рис. 8.21а. Динамическая объектная структура в ВП Torque Hysteresis. В субпанель загружен компонент ВП Run Test

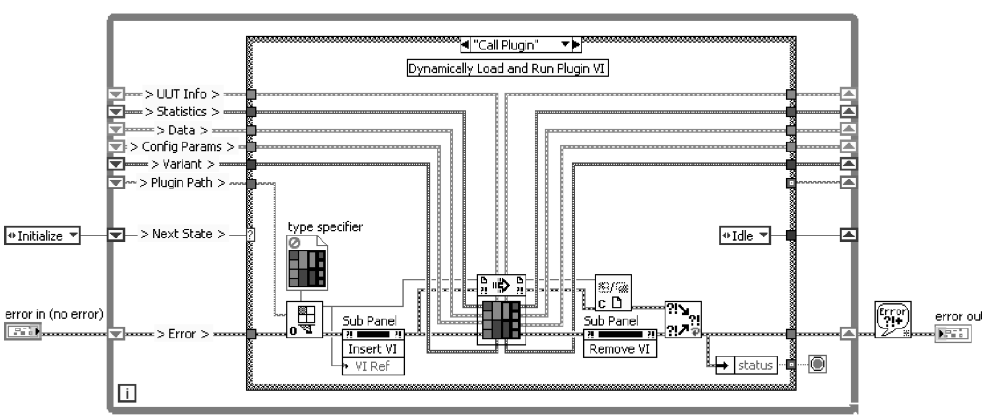


Рис. 8.21б. На блок-диаграмме основные структуры данных приложения распространяются между компонентами через сдвиговые регистры в состоянии *Call Plugin*

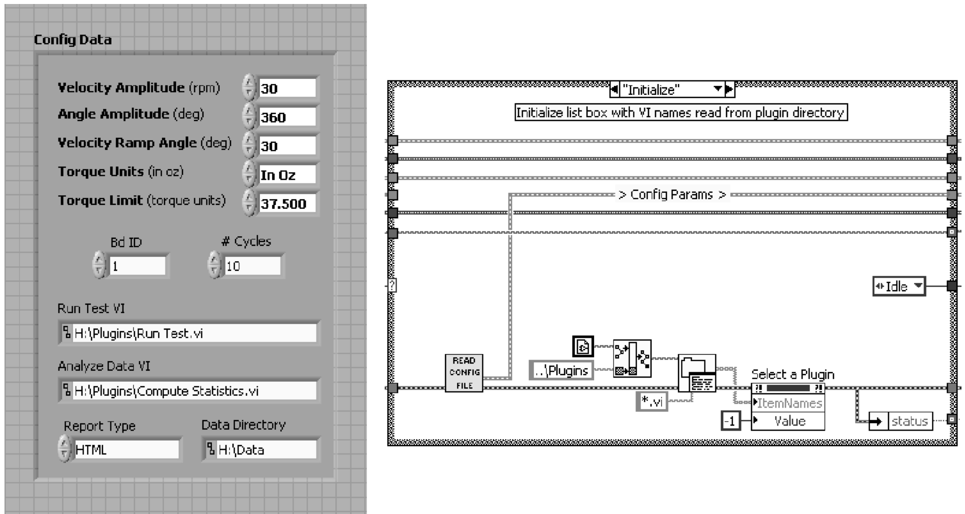


Рис. 8.21в. Кластер параметров движения объединен с различными параметрами для каждого компонента в новый кластер конфигурационных параметров. Эти параметры считываются в состоянии *Initialize* из файла конфигурации

8.4.2. Объектная структура приложения со многими циклами

Объектная структура приложения со многими циклами расширяет концепцию составных шаблонов и сочетает множество шаблонов с единственным циклом, процедуры обмена сообщениями, утилиты и управляющие элементы.

Правило 8.27 Создавайте параллельный цикл для каждой связанной параллельной задачи

Можно оптимизировать гибкость и производительность объектной структуры приложения, разделив обычные функциональные элементы на связанные задачи и применив для каждой параллельные циклы. Параллельные циклы дают возможность выполнять множество задач одновременно. LabVIEW для каждого параллельного цикла порождает отдельный поток, и каждый поток может обрабатываться отдельным процессором, если такие есть. Кроме того, параллельные циклы позволяют вам определить и четко настроить приоритет каждой задачи, используя задержки для циклов While и приоритеты для циклов Timed Loop, то есть объектная структура, основанная на нескольких параллельных циклах, может улучшить производительность системы.

Недостатком использования множества параллельных циклов является дополнительная сложность процедуры обмена данными между циклами. Поэтому

важно выделить те задачи, которые связаны друг с другом, но не слишком зависят друг от друга в вопросе использования данных. Реализация, представленная в этом разделе, использует очереди для обмена данными между циклами. Область применения этой объектной структуры – персональный компьютер, поэтому RT FIFO и общие переменные не нужны. Очереди более функциональны в данном случае, чем локальные и глобальные переменные. Реализация каждой задачи как параллельного цикла и использование очередей для обмена данными и синхронизации помогают увеличить число приложений, в которых может быть использована такая объектная структура.

Правило 8.28 Добавьте циклы обработки событий, основного конечного автомата, аппаратного ввода/вывода и обработки ошибок

Сложные приложения выигрывают от использования следующих параллельных циклов: обработки событий, основного конечного автомата, аппаратного ввода/вывода и обработки ошибок. Цикл обработки событий является наиболее эффективным методом отслеживания событий графического интерфейса, улучшает отзывчивость приложения на эти события и опыт пользователя. Всегда включайте цикл обработки событий для всех сложных приложений с графическим интерфейсом. Основной конечный автомат облегчает передачу, организацию, расширяемость и поддержку состояний. Все характерные для данного приложения последовательно выполняемые задачи реализуйте в рамках схемы основного конечного автомата. Аппаратный ввод/вывод зачастую является основной задачей приложения LabVIEW. Создавайте цикл аппаратного ввода/вывода для каждой задачи непрерывного ввода/вывода и расставляйте соответствующие приоритеты относительно других задач. Цикл обработки сообщений дает возможность создания отчетов обо всех значительных ошибках во всей системе, используя единственную процедуру.

Управление через события достигается использованием очередей или структуры событий для каждого непрерывного цикла. Используя структуры событий согласно Правилу 8.8, следует избегать событий Timeout (по истечении времени) или удалять случай этого события из структуры, а по Правилу 8.15 следует избегать терминала **timeout** также и для функций Enqueue и Dequeue Element. Используйте данную технику для всех циклов, чтобы гарантировать управление приложением через события. Казалось бы, расстановка приоритетов с использованием задержек или циклов Timed становится не такой важной, если каждый цикл управляется через события. Однако может быть так, что некоторые циклы должны выполняться в определенные временные интервалы. Также задержки предотвращают появление сжатых циклов. Сжатый цикл – это цикл, который монополизирует, согласно приоритету или типу выполняемой задачи, ресурсы процессора. Каждый вызов таких функций, как Wait (ms), Wait Until Next ms Multiple и ВП Time Delay Express, принудительно отдает потоку контроль над процессором, что позволяет выполняться другим задачам. Для лучших результатов следует всегда использовать задержки в непрерывных циклах While.

На рис. 8.22 показана объектная структура приложения со многими циклами, разработанная Грегом Барроузом (Greg Burroughs), главным инженером проектов в Bloomy Controls. Основная лицевая панель содержит большой невидимый элемент управления вкладками и инструкции по редактированию. Графический интерфейс пользователя состоит из множества экранов на вкладках, которые программно контролируются на блок-диаграмме. Блок-диаграмма состоит из цикла обработки событий, основного конечного автомата, цикла обработки ошибок и системы обмена данными, использующей очереди. Обратите внимание на то, что каждый цикл контролируется через очередь или структуру событий без использования терминалов и событий timeout, что облегчает управление через события. Кроме этого, данная объектная структура содержит множество конструкций для инициализации, настройки, безопасности, управления приложением, обработки ошибок, документации и т.д. Встроенных функций достаточно много, и их рассмотрение находится за пределами данной главы. Основная особенность этой объектной структуры – множество параллельных циклов. Для лучшей произво-

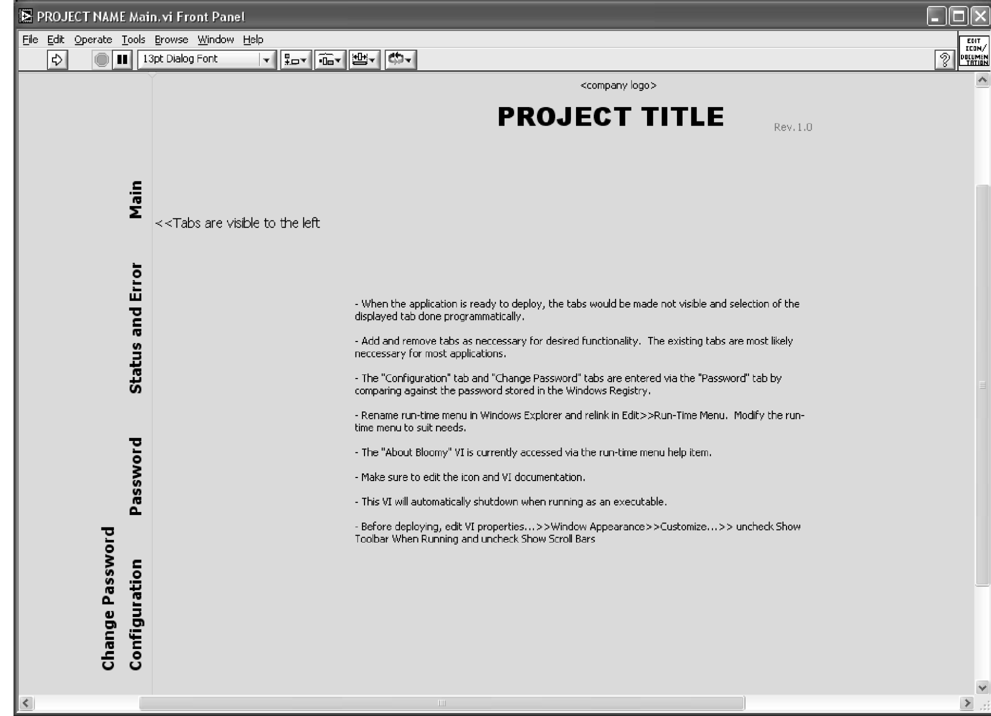


Рис. 8.22а. Лицевая панель ВП верхнего уровня объектной структуры приложения со многими циклами состоит из невидимого элемента управления вкладками и инструкций по редактированию и использованию

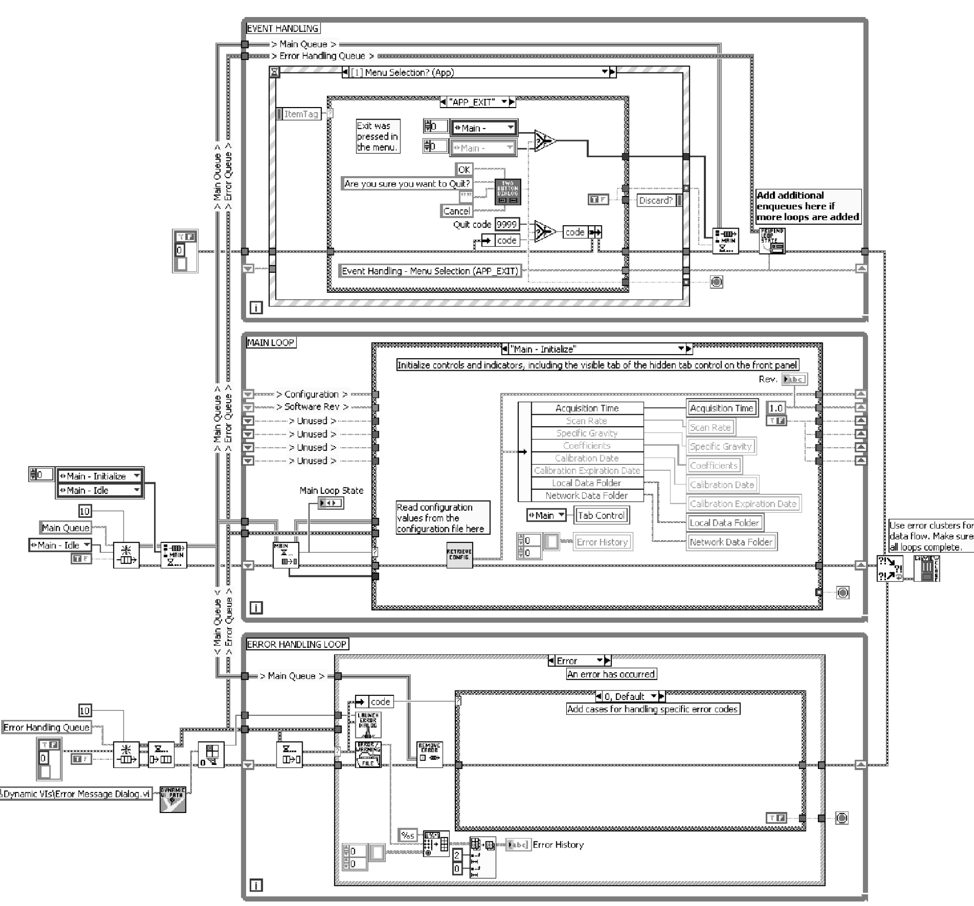


Рис. 8.22б. Блок-диаграмма объектной структуры верхнего уровня состоит из цикла обработки событий, основного конечного автомата, цикла обработки ошибок. Данная объектная структура обладает множеством особенностей, эволюционировавших во времени

дительности, расширяемости и простоты поддержки в данной структуре объединены лучшие шаблоны с одним циклом.

Эта объектная структура и все ВПП эволюционировали во времени и теперь представляют надежную стартовую позицию для разработки сложного приложения. Кроме этого, данная объектная структура и все ВПП являются общепризнанными стандартами в нашей организации, что обеспечивает общность, производительность, простоту поддержки и расширения и безусловно хороший стиль программирования. Объектная структура интегрирована в проект LabVIEW для создания организованной структуры файлов по аналогии с тем, как это сделано в главе 2.

В разделе 8.3 «Составные шаблоны» мы улучшали производительность ВП Torque Hysteresis, используя параллельные циклы. На рис. 8.23 мы снова улучшили это приложение, использовав объектную структуру приложения со многими циклами (Multiple-Loop Application Framework). Раздельные цикл обработки сообщений (помечен как Event Handling) и основной конечный автомат (помечен как Main Loop) улучшают реакцию приложения на события графического интерфейса и производительность передачи состояний по сравнению с шаблоном событийно управляемого конечного автомата. Сбор данных происходит в специальном цикле, помеченном как DAQ Loop, для непрерывности наблюдения данных

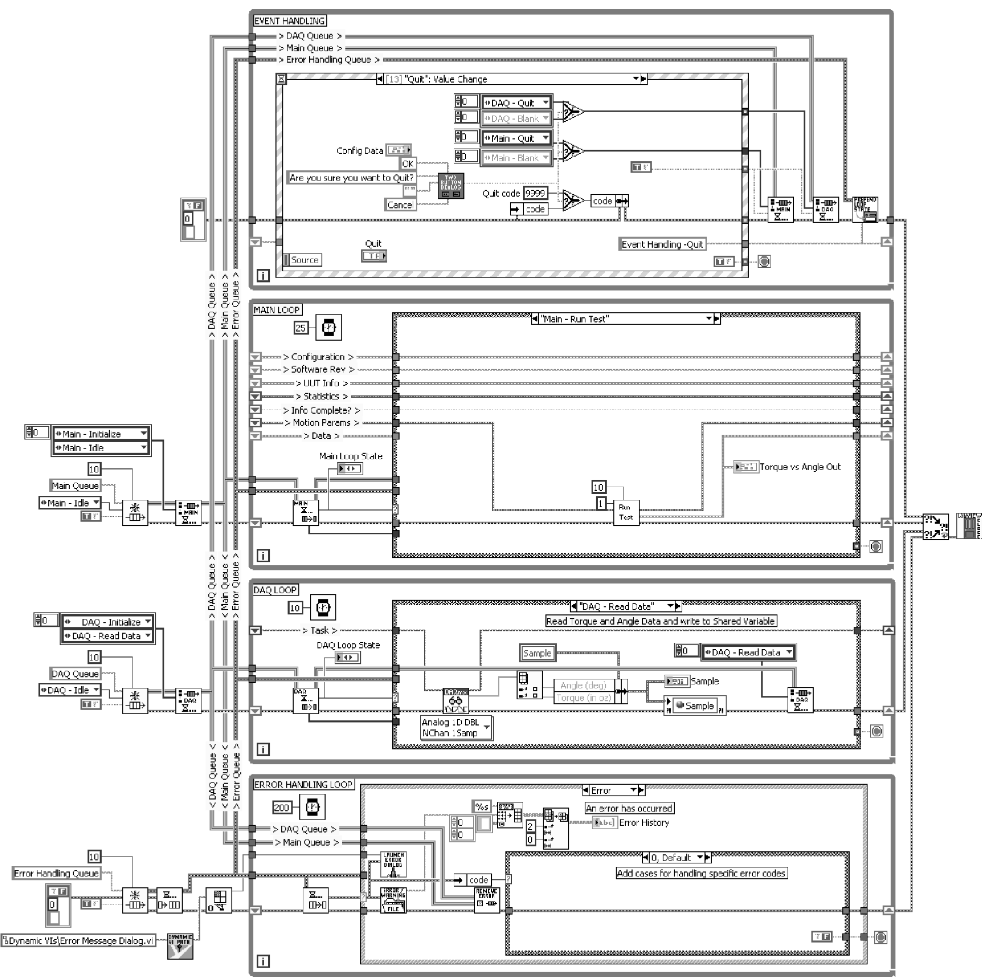


Рис. 8.23. Блок-диаграмма высшего уровня ВП Torque Hysteresis содержит отдельные циклы обработки сообщений, конечного автомата, сбора данных и обработки ошибок

о моменте вращения и угле, по аналогии с параллельными циклами на рис. 8.18. Цикл, отмеченный как Error Handling Loop, обрабатывает ошибки со всех циклов. Это позволяет приложению создавать отчеты об ошибках, восстанавливаться и продолжать работу. Обратите внимание на то, что основной конечный автомат, цикл DAQ и цикл обработки ошибок построены по шаблону конечного автомата с очередями. Для предотвращения сжатых циклов используются задержки, обмен сообщениями осуществляется через очереди.

Цикл DAQ Loop разработан для того, чтобы добавлять новое состояние в очередь при каждой операции чтения. Как видно, случай **DAQ–Read Data** добавляет новый элемент **DAQ–Read Data** в конец очереди каждый раз, когда случай выполняется. Это приводит к тому, что это состояние выполняется непрерывно, пока не завершится работа приложения или не произойдет ошибка. Соответственно, нужно добавить задержку, чтобы этот цикл не стал сжатым циклом. В качестве предупреждающей меры задержки добавлены и в другие циклы While.

8.4.3. Модульная объектная структура приложения со многими циклами

Казалось бы, объектную структуру приложения со многими циклами можно увеличивать в масштабе без ограничений. Функциональность увеличивается простым добавлением случаев в каждую конструкцию с одним циклом или добавлением новых циклов и очередей для каждого нового параллельного задания. Сложное приложение выглядит довольно впечатляюще, как например ВП Meticulous, описанный в главе 1. Однако объектная структура приложения со многими циклами нарушает несколько правил хорошего стиля программирования в LabVIEW. В главе 4 мы представили правила для разбиения блок-диаграмм на отдельные модули – ВПП и ограничения размеров блок-диаграммы одним экраном в разрешении 1280×1024 (см Правила 4.1, 4.4, 4.5, 4.6). Хотя объектная структура содержит множество ВПП, блок-диаграмма верхнего уровня занимает два и более экрана, и индекс вложенности (количество уровней вложения) часто ниже 3.0. *Модульная объектная структура приложения* со многими циклами является альтернативой обычной объектной структуре приложения с множеством циклов, в которой циклы выделены в отдельные модули – ВПП.

Правило 8.29 Передавайте ссылки на элементы управления в ВПП, используя кластер с определением типов

Поскольку каждый цикл объектной структуры приложения с множеством циклов очерчивает отдельную задачу, каждый цикл является кандидатом на выделение в ВПП. Это улучшит модульность структуры и снизит размер блок-диаграммы верхнего уровня. Однако в каждом цикле находятся терминалы объектов лицевой панели, о чем нужно подумать. Управляющие элементы и индикаторы ВП верхнего уровня считываются и записываются в ВПП с использованием ссылок на элементы и узлов Property Node, настроенных на чтение или запись свойства Value соответствующего элемента. Однако, поскольку это всего лишь образец, многие элементы управления еще не созданы. Тогда как же мы можем разбить объектную структуру на модули? Просто передайте ссылки на элементы управления ВП верхнего уровня, используя кластер, сохраненный как тайпдеф, и по мере появления новых элементов на блок-диаграмме добавляйте в этот кластер новые ссылки. Таким образом, вам нужно поддерживать только тайпдеф. Назначения на соединительной панели ВПП останутся прежними. Или вы можете сделать модульным пользовательский интерфейс ВП верхнего уровня, используя лицевые панели ВПП и субпанели, по аналогии с динамической объектной структурой, которую мы обсуждали ранее.

Правило 8.30 Избегайте использования утилиты SubVI from Selection (ВВП из участка кода) с непрерывными циклами

Давайте обратимся к конструкции, состоящей из цикла высокого уровня, превращенного в ВПП как цикличный ВПП (**loop-subVI**). Кто-то может захотеть быстро выделить циклы в ВПП, используя утилиту subVI from Selection (ВПП из участка кода). Это можно сделать, выбрав цикл и опцию **Edit ⇒ Create SubVI**. Однако утилита SubVI from Selection не очень хорошо обрабатывает терминалы лицевой панели. По умолчанию она создает константы указателей на все элементы управления и индикаторы внутри выбранной области и передает их в ВПП, используя отдельный соединительный терминал для каждого. Кроме того, данные, записанные на индикаторы, возвращаются через соединительные терминалы ВПП, соединенные с индикаторами ВП верхнего уровня. При таком подходе возникает три проблемы:

1. Если в цикле больше пары терминалов на лицевой панели, то у ВПП появятся лишние терминалы и возникнет путаница в проводниках данных снаружи и внутри ВПП.
2. Обновление индикаторов будет происходить только после того, как завершится выполнение ВПП.
3. Поддержка становится очень утомительной.

К тому же добавление или удаление ссылок на элементы для каждого нового управляющего элемента или индикатора, необходимого в ВПП, влечет за собой изменения терминала, соединительной панели, проводников данных. Вместо этого следует всегда передавать ссылки на элементы, используя кластер ссылок, сохраненный как тайпдеф, и избегать использования утилиты SubVI from Selection.

Правило 8.31 Цикл обработки сообщений оставляйте на верхнем уровне

Правило 8.32 Высокоскоростное обновление экрана следует также оставлять на верхнем уровне

Цикл обработки сообщений обычно очень жестко связан с графическим интерфейсом пользователя. И в самом деле, многие терминалы лицевой панели ВП с графическим интерфейсом поддерживаются этой структурой. Техническая поддержка упрощается, если этот цикл поддерживается на блок-диаграмме верхнего уровня, а не выделяется как модуль в циклический ВПП. Кроме того, операции чтения и записи через узел Property Node в несколько раз медленнее, чем чтение и запись непосредственно с терминалов элементов. Таким образом, производительность увеличивается, а необходимость техподдержки уменьшается, если цикл обработки событий остается на блок-диаграмме главного ВП с графическим интерфейсом, обычно это ВП верхнего уровня. Аналогично следует поступать со всеми циклами, осуществляющими высокоскоростное обновление экрана, чтобы увеличить частоту обновлений. Или нужно распределить пользовательский интерфейс ВП верхнего уровня между лицевыми панелями циклических ВПП и загружать их через субпанели.

На рис. 8.24а показана блок-диаграмма ВП Torque Hysteresis, выполненная с использованием модульной объектной структуры приложения со многими циклами. На левой части блок-диаграммы происходит инициализация трех очередей и кластера ссылок на элементы управления. Эти очереди передаются в структуру событий и циклические ВПП для осуществления обмена данными. Ссылки на элементы управления используются в циклических ВПП для чтения и записи данных с элементов ВП верхнего уровня через узлы Property Node. Когда приложение завершается, то ошибки объединяются, и LabVIEW закрывается, для этого используется ВПП справа.

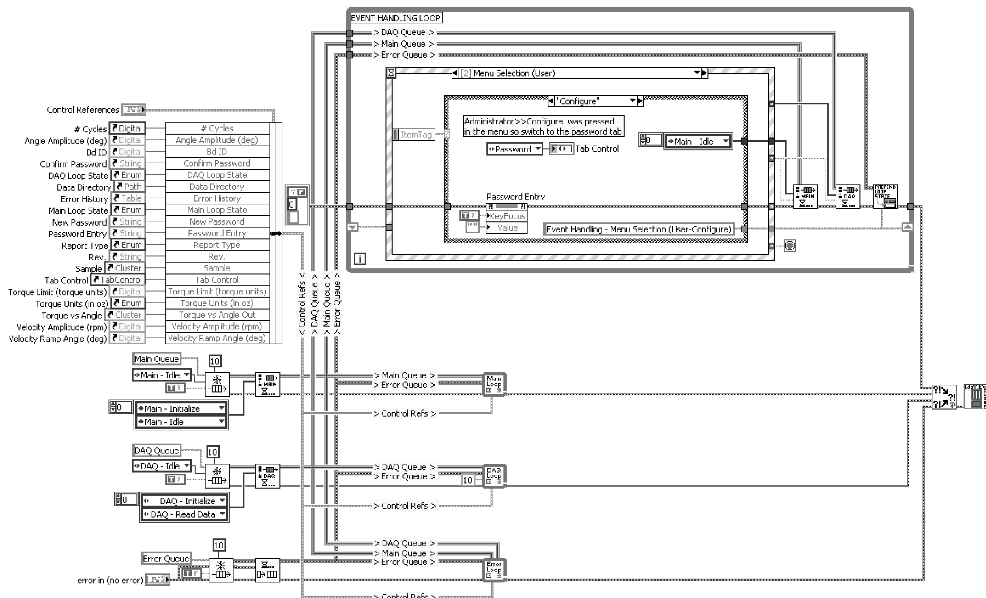
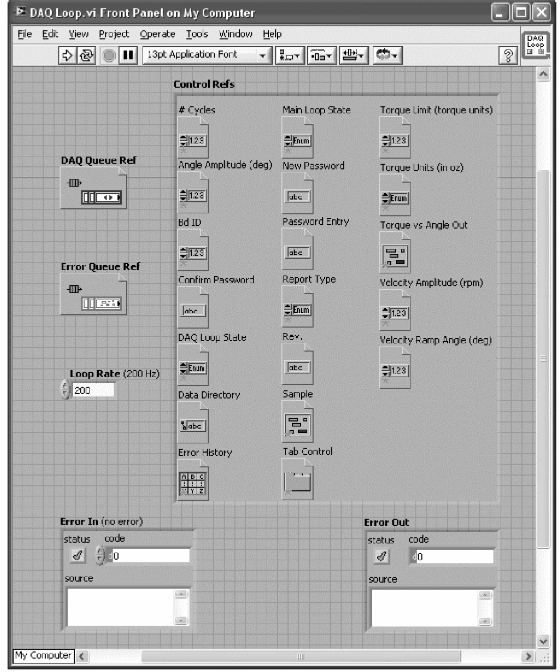


Рис. 8.24а. Блок-диаграмма верхнего уровня ВП Torque Hysteresis разбита на модули – три циклических ВПП и один цикл обработки событий

Лицевая панель и блок-диаграмма ВП DAQ Loop показаны на рис. 8.24б. Лицевая панель содержит ссылки на очереди **DAQ Queue** и **Error Queue**, а также кластер (тайпдеф) ссылок на элементы управления. В буфере **DAQ Queue** содержится перечень, который управляет селектором случаев ВПП. **Error Queue** сообщает обо всех ошибках в ВП Error Loop. Ссылки на элементы управления позволяют блок-диаграмме обновлять индикаторы на лицевой панели ВП с графическим интер-



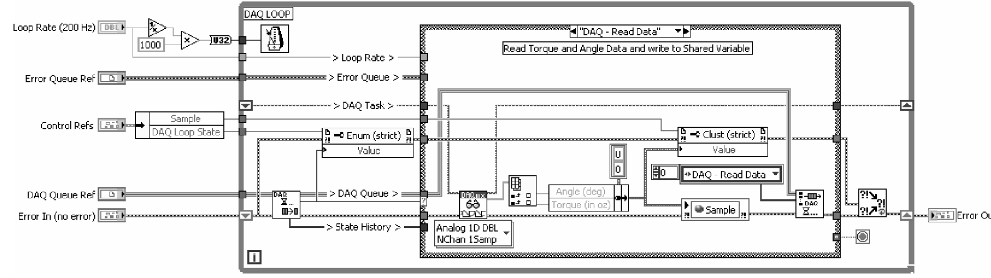


Рис. 8.24б. ВПDAQ Loop – это циклический ВПП, содержащий процедуру сбора данных. Лицевая панель содержит ссылки на очереди и кластер (тайпдеф) ссылок на элементы управления. Эта блок-диаграмма использует узлы Property Node для того, чтобы непрерывно обновлять индикаторы в ВП верхнего уровня

фейсом, в том числе перечень **DAQ Loop State** и кластер **Sample**. Блок-диаграмма состоит из конечного автомата с очередями. Слева от структуры случаев вызывается ВПП, который удаляет состояние, которое должно выполняться следующим, из очереди и обновляет **DAQ Loop State**, используя узел Property Node, настроенный на свойство Value слева от структуры случаев. Соответствующий индикатор находится на вкладке диагностики лицевой панели ВП с графическим интерфейсом. Это позволяет пользователю с соответствующими правами просматривать активное состояние ВП DAQ Loop. В состоянии **DAQ–Read Data**, ВПП считывает образец данных и обновляет индикатор кластера **Sample** и общую переменную и добавляет другое состояние **DAQ–Read Data** в очередь. Ограничение времени выполнения цикла выполнено через функцию Wait Until Next mS Multiple и основано на программируемой частоте цикла.

8.5. Примеры

В этом разделе содержится еще несколько примеров приложений, использующих шаблоны и объектные структуры.

8.5.1. ВП Elapsed Time

Показанный на рис. 8.25 ВП измерения прошедшего времени (Elapsed Time) является глобальным функционалом, который хранит стартовую временную отметку в памяти сдвигового регистра и вычисляет прошедшее с момента запуска приложения время. Когда терминал **Mode (Update)** установлен в состояние **Init**,

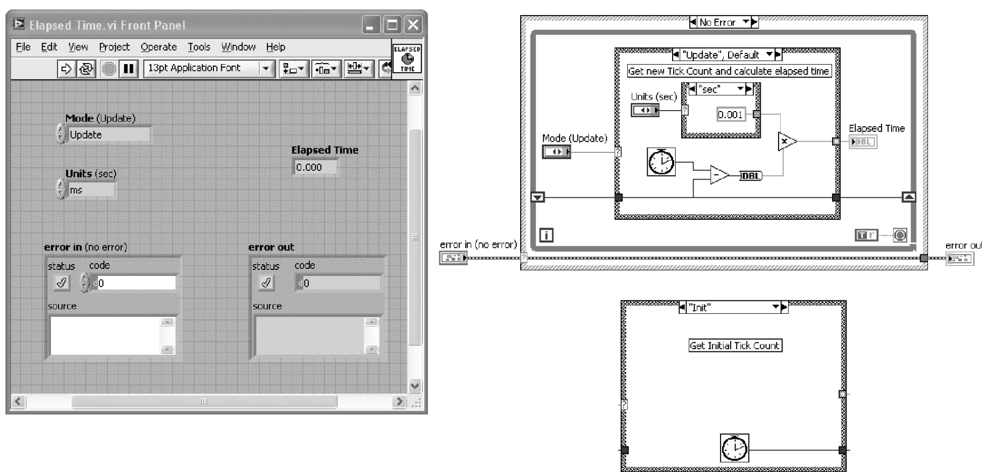


Рис. 8.25. ВПElapsed Time является функциональной глобальной переменной, которая хранит стартовую временную отметку в памяти сдвигового регистра и вычисляет прошедшее с момента запуска приложения время

то случай **Init** основной структуры случаев вызывает функцию Tick Count (ms), чтобы получить стартовую временную отметку и записать ее в сдвиговый регистр. Последующие вызовы ВП Elapsed Time через терминал **Mode** в значении **Update** приводят к выполнению случая **Update**, который получает данные о текущей временной отметке и вычисляет прошедшее время в выбранных единицах. Этот ВП может быть полезен для наблюдения или обнуления прошедшего времени из нескольких мест в приложении или просто для уменьшения путаницы на блок-диаграмме, вызывающей ВП.

8.5.2. ВП Poll Instrument Response

ВП Poll Instrument Response является ВПП, который периодически считывает данные с устройства. Этот ВП полезен в приложениях, работающих с последовательно подключенными устройствами, и является наглядным примером. Лицевая панель на рис. 8.26а содержит элементы управления **resource name**, **bytes to read**, **timeout**, r**esponse** и **total bytes**, вдобавок к стандартному кластеру ошибок. Блок-диаграмма на рис. 8.26б считывает доступное число байт с последовательного порта и строит ответную строку и число байт, которые хранятся в сдвиговом регистре. Функция Wait (ms) добавляет задержку в 25 микросекунд. Прошедшее время вычисляется и сравнивается с условием истечения времени через функцию Time Count и несколько арифметических функций. Цикл повторяется до тех пор, пока число считанных байтов не достигнет значения **bytes to read** или не произойдет ошибка, или не будет выполнено условие истечения времени.

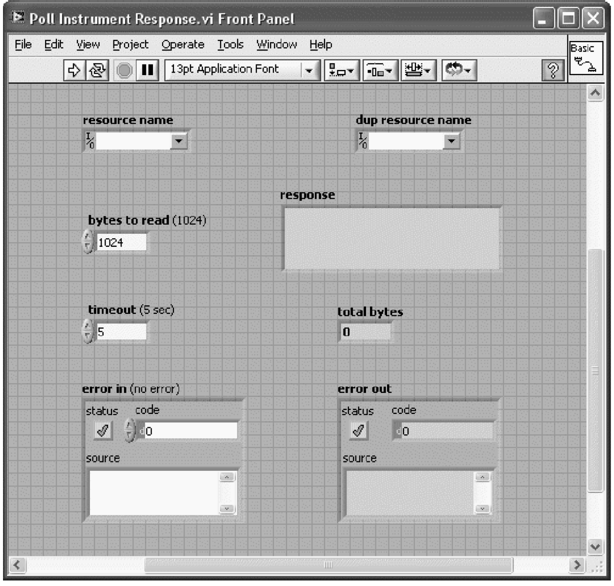


Рис. 8.26а. Лицевая панель ВП Poll Instrument Response

Обратите внимание на то, что этот ВП является гибридом шаблонов непрерывного цикла и начального ВПП. Он содержит цикл While с множеством условий выхода из цикла, включая истечение времени. Также он содержит процедуру отслеживания ошибок, включая кластеры ошибок и структуру Error Case. На рис. 8.26в представлена альтернативная блок-диаграмма с незначительными улучшениями. Так ВП Elapsed Time заменяет арифметические функции определения времени, которые находятся внизу блок-диаграммы на рис. 8.26б. Кроме этого, кластер ошибок через туннели в нижней части вертикальной границы проходит через структуру случаев и цикл while. Измененная ВПП стилистически превосходит оригинальную.

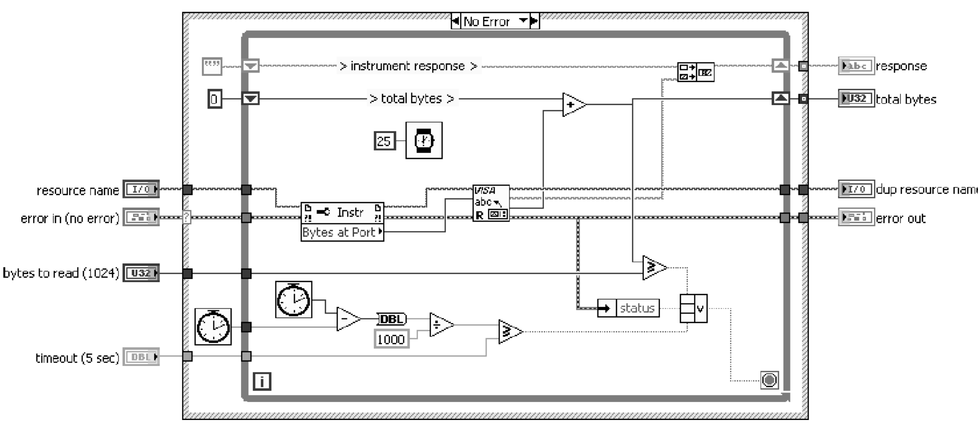


Рис. 8.26б. Блок-диаграмма является гибридом шаблонов непрерывного цикла и ВПП Immediate

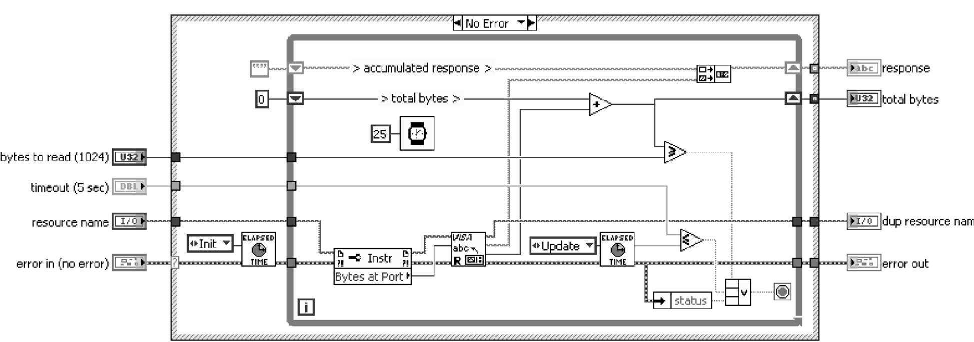


Рис. 8.26в. Блок диаграмма ВП Poll Instrument Response улучшена функциональной переменной ВП Elapsed Time

8.5.3. Нетрадиционный конечный автомат

ВП на рис. 8.27 содержит несколько черт, общих с шаблоном классического конечного автомата, но в целом ВП выполнен нестандартно и с множеством нарушений правил стиля. К общим чертам можно отнести цикл While, содержащий структуру Case, сдвиговый регистр и меню логических элементов управления. На блок-диаграмме присутствуют следующие нарушения:

- проводники входят в структуру через горизонтальную границу (см. Правило 4.13);
- поток данных справа налево (см. Правило 4.22);
- нет меток на проводниках данных, выходящих из сдвиговых регистров (см. Правило 4.37);
- метки функций видимы (см. Правило 4.45);
- ошибки не отслеживаются (см. Правило 7.3);
- используется ВП Simple Error Handler вместо ВП General Error Handler (см Правило 7.7);
- кластер ошибок вводится не через нижний туннель (см. Правило 7.17);
- опрос терминалов графического интерфейса вместо структуры событий (см. Правило 8.4);
- не используются перечни для селекторов (см. Правило 8.12);
- две параллельные структуры событий (см. Правило 4.13);
- метки элементов управления не видны (см. Правило 9.1).

Логическое меню используется для выбора случая верхней структуры событий. Нижняя структура событий контролируется кнопкой **Run VI**. Об этом не так легко догадаться, потому что метка элемента управления не видна. Поскольку обе структуры событий управляются логическими элементами, обе они могут быть объединены в одну структуру событий. Это повлечет за собой добавление логических элементов **Run VI** и **Quit** в массив, в котором осуществляется поиск значения TRUE и простое добавление случаев для соответствующих состояний в структуру случаев.

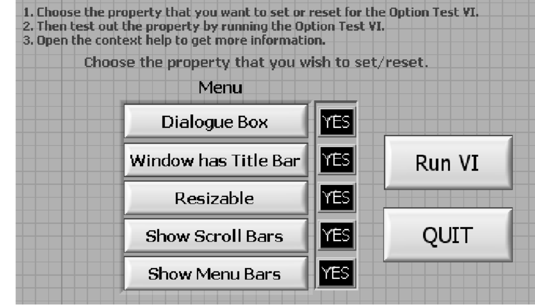


Рис. 8.27а. Лицевая панель ВП Set Window Options состоит из логического меню и нескольких светодиодов

Обратите внимание, что сдвиговый регистр поддерживает только предыдущие значения логического меню и не передает никакой информации о состоянии. Только логическое меню контролирует выбор случаев в верхней структуре случаев. Поэтому никакой код, выполняющийся в структуре случаев, не может повлиять на выбор следующего состояния. Это противоречит шаблону классического конечного автомата, в котором следующее желаемое состояние передается через сдвиговый регистр и соединяется с селектором структуры Case.

Первый вариант, представленный на рис. 8.27в, использует шаблон классического конечного автомата. Перечень, как тайпдеф, поддерживает состояния, и одна структура Case содержит случаи для всех состояний. Кнопки **Run VI** и **Quit** добавлены в логическое меню и опрашиваются в кадрах **Idle** и **Default** структуры Case. Выбранное состояние передается через сдвиговый регистр на следующую итерацию цикла. Метки функций не видны, к проводникам данных, выходящим из сдвиговых регистров, добавлены свободные метки. Помимо состояний **Idle** и **Default** в конечном автомате появились состояния **Initialize**, **Shutdown**, **Blank** и **Run**. Все метки элементов управления стали видимыми.

В этом примере примечательно то, что существует однозначное соответствие между логическими элементами управления и состояниями приложения, за исключением состояний: **Initialize**, **Blank** и **Idle**, **Default**. Каждое состояние вызывается в ответ на изменение значения логического управляющего элемента. Поэтому этот ВП эффективнее было бы реализовать через структуру событий, как показано на рис. 8.27г. Структура событий является наиболее эффективным способом обработки событий пользовательского интерфейса. Случаи структуры событий настроены так, чтобы отвечать на событие Value Change (Изменение значения) каждого логического элемента управления, и содержат код из соответству-

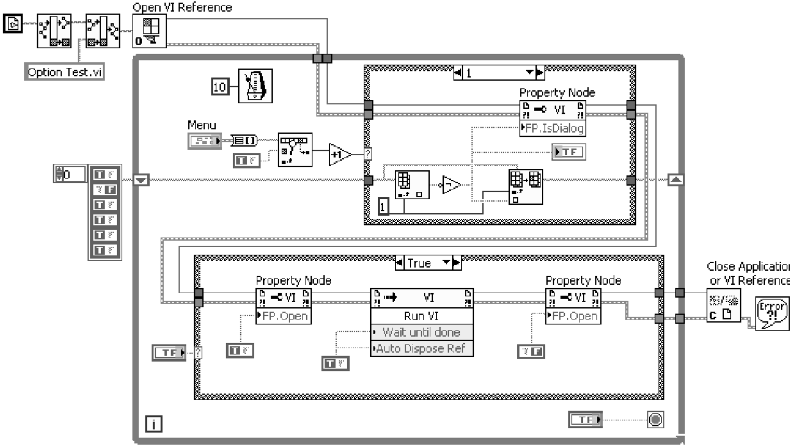


Рис. 8.27б. Блок-диаграмма содержит несколько компонентов шаблона конечного автомата, но нарушает множество правил стиля, касающихся как конечных автоматов, так и блок-диаграммы в целом

ющих состояний классического конечного автомата. Состояния **Idle**, **Default**, которые в схеме классического конечного автомата опрашивали логические элементы управления, устранены, так как введен цикл обработки событий. Состояние **Initialize**, содержащее процедуру инициализации, заменено случаем **Timeout** цикла обработки событий. Однако по Правилу 8.8 следует избегать событий типа Timeout. На рис. 8.27г сдвиговый регистр соединен с терминалом **Timeout** структ-

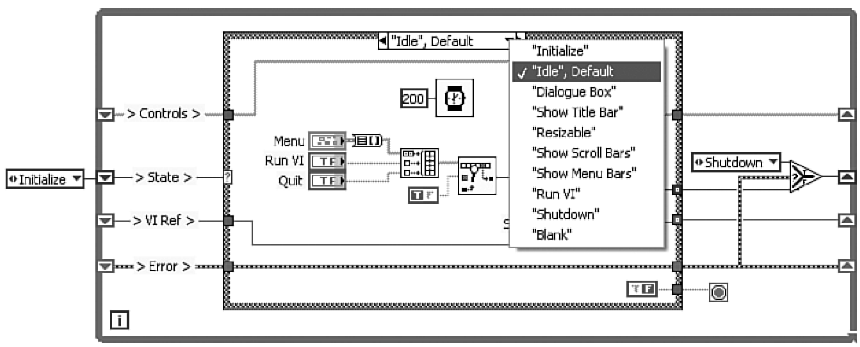


Рис. 8.27в. Блок-диаграмма содержит некоторые элементы шаблона классического конечного автомата

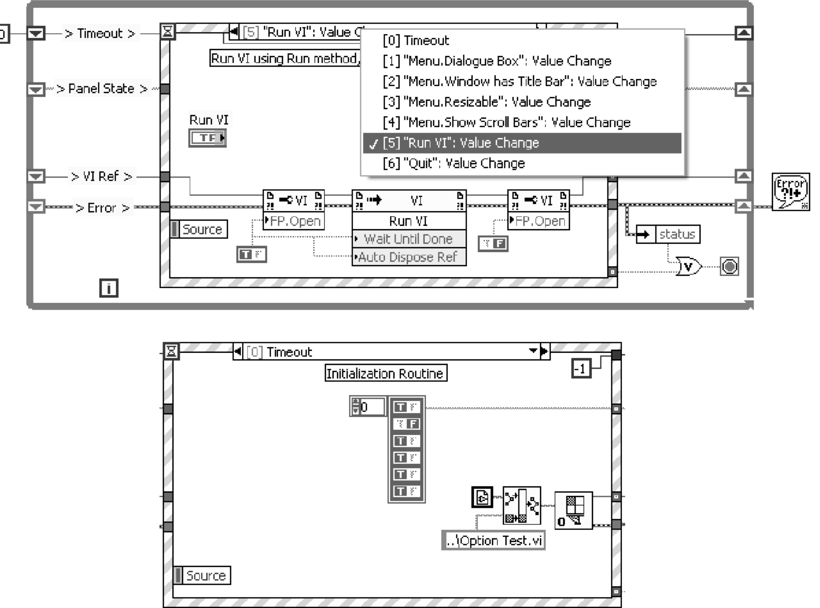


Рис. 8.27г. ВП выполнен с использованием шаблона цикла обработки событий. Состояния заменены событиями. Для осуществления процедуры инициализации используется событие ***Timeout***

туры событий. Значение сдвигового регистра равно 0, то есть событие **Timeout** запускается немедленно. Когда в рамках события **Timeout** выполнится процедура инициализации, то сдвиговый регистр устанавливается на значение –1, и событие **Timeout** отключается. Таким образом, случай **Timeout** похож на состояние **Initialize**.

8.5.4. ВП Centrifuge DAQ

Основной цикл ВП Centrifuge DAQ мы рассматривали в главе 4. На общей схеме, показанной на рис. 8.28, виден составной шаблон, состоящий из конечного автомата с очередями в верхней части, цикла обработки событий в нижней и очереди для обмена данными. Цикл обработки событий нарушает несколько правил, в том числе Правило 8.8 о событиях типа Timeout, Правило 8.18 о разных горизонтальных размерах циклов, и содержит незавершенную процедуру обработки ошибок (см. главу 7).

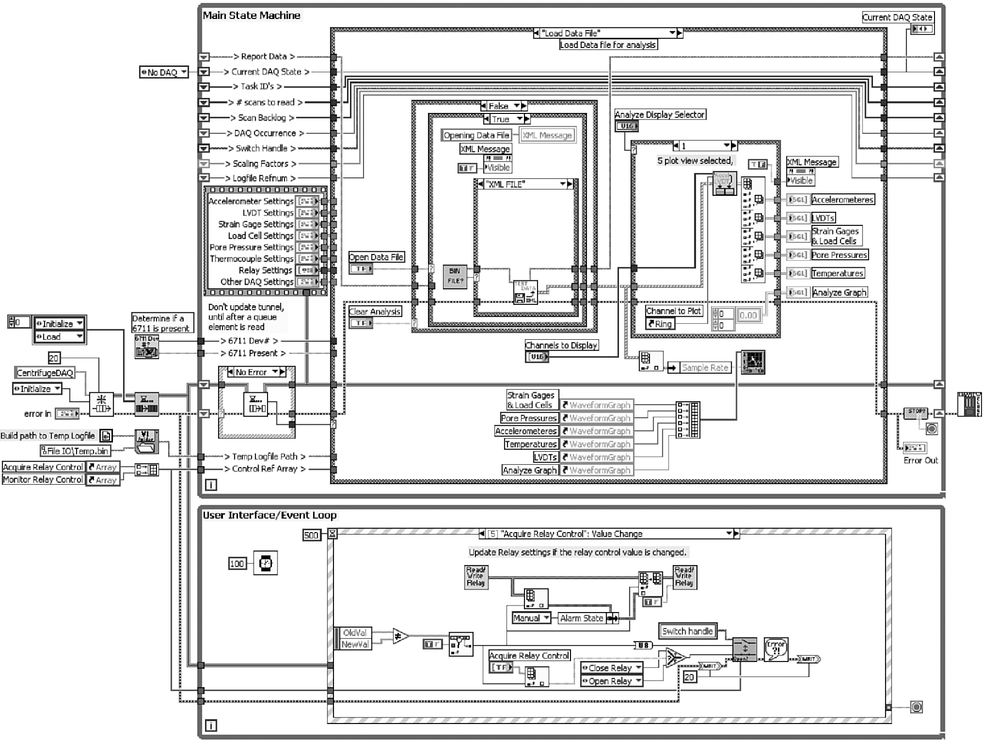


Рис. 8.28. ВП Centrifuge DAQ является составным шаблоном, состоящим из конечного автомата с очередями в верхней части, цикла обработки событий в нижней и очереди для обмена данными

8.5.5. Утилита управления датчиком

Приложение на рис. 8.29 обладает полным набором средств управления и объединено с драйвером цифрового датчика давления. Управляющая утилита настраивает, получает и записывает данные и предоставляет интерактивный контроль над любым количеством датчиков. Блок-диаграмма состоит из составного шаблона, основанного на объектной структуре приложения с многими циклами. Также она содержит цикл обработки событий графического интерфейса пользователя, конечный автомат с очередями для передачи состояний и непрерывный цикл обработки ошибок, что позволяет нескольким задачам выполняться параллельно. Например, несколько датчиков могут получать и записывать данные, в то время как настраивается другой датчик, без прерывания процедуры получения и записи данных.

Рис. 8.29а. Главная лицевая панель управляющего датчиком приложения состоит из простого логического меню

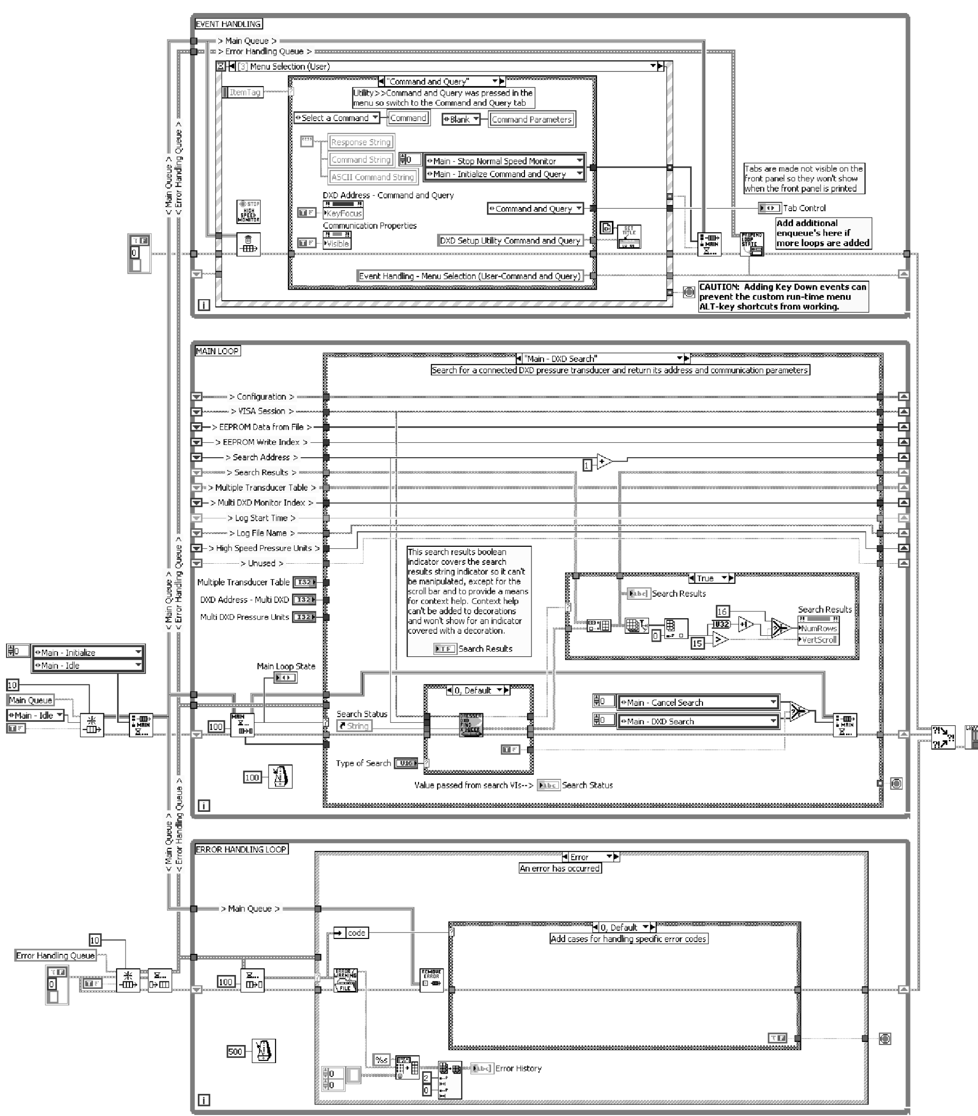


Рис. 8.29б. Блок-диаграмма использует объектную структуру приложения с многими циклами

8.5.6. Распределенная управляющая система

На рис. 8.30 показана блок-диаграмма верхнего уровня распределенной управляющей системы, выполненная с использованием LabVIEW RT. Это модульная объектная структура приложения со многими циклами. Блок-диаграмма цикли-

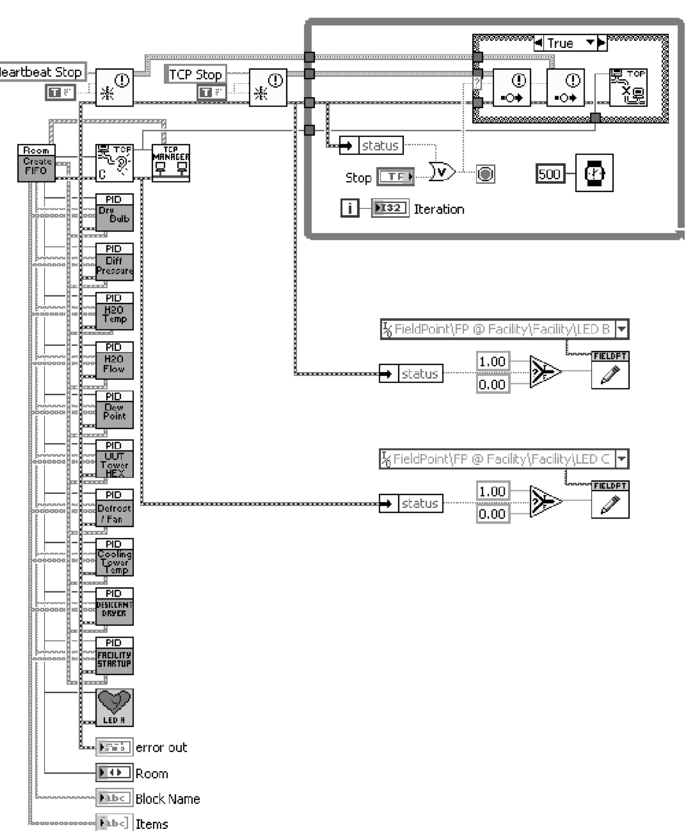


Рис. 8.30. Блок-диаграмма верхнего уровня распределенной управляющей системы использует модульную объектную структуру приложения со многими циклами

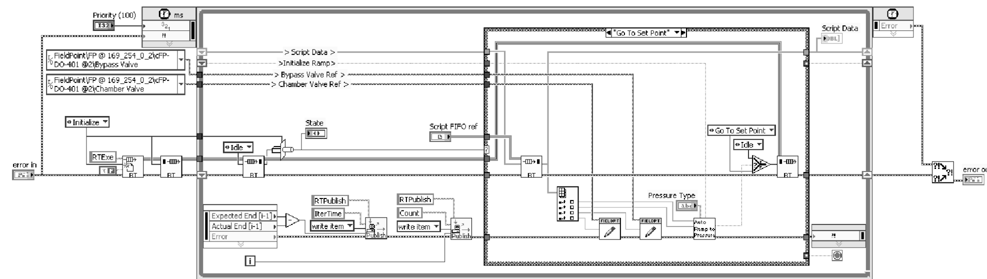
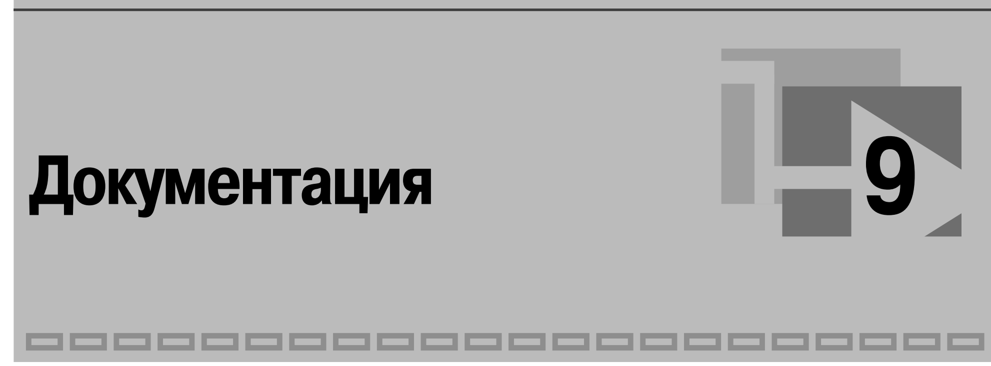


Рис. 8.31. Блок-диаграмма одного из ВПП содержит цикл Timed Loop. Обмен данными между ВП осуществляется с использованием буферов RT FIFO

ческого ВПП содержит цикл Timed Loop, изображенный на рис. 8.31. Обмен данными между ВП осуществляется с использованием буферов RT FIFO.

Ссылки

1. Образцы нескольких шаблонов, которые мы обсуждали в этой главе, можно скачать с www.bloomy.com/lvstyle.
2. "Using LabVIEW to Create Multithreaded Applications for Maximum Performance and Reliability" и "LabVIEW and Hyperthreading" доступны для свободного скачивания в NI Developer Zone (Зона разработчика NI) по адресу www.zone.ni.com.



9 Документация

Документация – это любое описание структуры, компонентов или операций системы, приложения или исходного кода. Документация включает в себя спецификацию, проектную документацию, документы по исходному коду, руководства пользователя и online-поддержку. Спецификация и проектная документация обсуждались нами в главе 2 «Приготовьтесь к хорошему стилю». Документация по исходному коду состоит из комментариев, описаний и текста, полезных для разработчиков для объяснения исходного кода. Руководство пользователя – это документ, в котором описано, как пользоваться приложением с точки зрения конечного пользователя. Online-поддержка – это документация, встроенная в приложение.

Все эти типы документации применимы к приложениям LabVIEW. Требования к документации сильно различаются в зависимости от типа приложения, разработчика, конечного пользователя, организации и индустрии. Например, коммерческое приложение может требовать развернутого руководства пользователя и online-поддержки, тогда как приложение для эксперимента в одной лаборатории может вообще не нуждаться в документации. Многие организации имеют внутренние стандарты, которым должны удовлетворять системы и программное обеспечение. Структуры с государственным управлением обычно имеют строгие требования к документации, введенные управляющим комитетом. В этой главе мы сосредоточимся на документации исходного кода LabVIEW, составляющей хорошего стиля программирования вне зависимости от разработчика, конечного пользователя, организации или индустрии.

Миф 9.1 *Блок-диаграммы LabVIEW являются самодокументирующимися.*

Графическая природа LabVIEW дает огромное преимущество по сравнению с текстовыми языками программирования с точки зрения понятности кода. Лучше один раз увидеть, чем сто раз прочитать. Однако это миф, что исходный код LabVIEW сам себя документирует. Поскольку документация используется для описания исходного кода, она не может быть такой же, как исходный код. Блок-диаграммы LabVIEW в конечном счете разработаны для графической компиля-

ции, тогда как документация разработана исключительно для лингвистического выражения. В основном документация – это текст, а поскольку код LabVIEW графический, то он определенно не текстовый. Документация используется для поддержки и усиления читабельности графического кода.

> ***Теорема 9.1*** *Хороший стиль разработки уменьшает усилия по созданию документации.*

Читабельность – это один из ключевых атрибутов хорошего стиля, о чем говорится на протяжении всей книги. Чем более удобочитаем и понятен исходный код, тем меньше необходимых усилий для создания документации. В самом деле, есть некоторое наложение между читабельным исходным кодом, содержащим интуитивно понятный встроенный текст, и документацией. Например, в главе 3 «Стиль лицевой панели» подчеркивается важность лаконичных и понятных меток элементов управления. В разделе 9.2 обсуждается связь между метками видимых элементов управления и читабельностью блок-диаграммы. Множество конструктов в LabVIEW используют текст, чтобы улучшить читабельность, например функции разделения кластера по имени и пронумерованный тип данных.

Другой пример, если архитектура вашей блок-диаграммы верхнего уровня использует один из шаблонов, которые мы обсуждали в главе 8 «Шаблоны», тогда вам нет необходимости писать длинное описание того, как работает этот шаблон. Для этого существуют другие ссылки. Стандартные шаблоны рекомендованы из-за их высокой читабельности, производительности и простоты поддержки. Однако если вы хотите создать свою собственную архитектуру ВП, тогда вам понадобится документация. Чем сложнее и уникальнее будет эта архитектура, тем больше понадобится ее объяснять. Правила стиля в этой книге направлены на то, что бы облегчить читабельность исходного кода. Поэтому чем лучше ваш стиль как разработчика, тем меньше дополнительных усилий вам понадобится для создания документации по исходному коду. Более того, когда вы используете функцию LabVIEW Print VI Documentation (Печать документации ВП), ваш исходный код и документация будут автоматически преобразованы в документ. Удобочитаемый исходный код, хорошая практика документирования и встроенные инструменты LabVIEW помогают упростить процесс создания документации. Особенности Print VI Documentation обсуждаются в разделе 9.4.

Документация напрямую связана с читабельностью, простотой поддержки и использования. Таким образом, документация – это неотъемлемая часть хорошего стиля. Часто документацию приносят в жертву скорости разработки приложения. Как уже обсуждалось в главе 1 «Значимость стиля», короткие пути не уменьшают время разработки, когда рассматривается весь жизненный цикл приложения. Скорее, хороший стиль разработки снижает время и усилия, потраченные на разработку приложения.

В этой главе представлены правила создания документации исходного кода, встроенной в лицевую панель, блок-диаграмму, иконку и свойства VI, и элементов управления. Кроме того, мы рассмотрим правила из предыдущих глав, которые напрямую связаны с читабельностью, такие как соответствующее использование текста. Если вы овладели правилами из предыдущих глав, то ваши ВП уже удобочитаемы и обладают отличным основанием для оптимального процесса документирования.

9.1. Документация лицевой панели

Документация лицевой панели состоит из меток элементов управления, описаний и свободных меток. Эти элементы важны для пользователей точно так же, как и для разработчиков. Правила стиля и примеры, относящиеся к тексту на лицевой панели, представлены в главах 3 и 6. На всякий случай в этой главе мы повторим эти правила и приведем примеры.

> ***Правило 3.17 Минимизируйте текст на лицевой панели***

> *Правило 3.18 Удаляйте шаблонные инструкции сразу после того, как внесены изменения*

Избегайте длинных комментариев в качестве постоянных меток на лицевой панели. Это особенно важно для ВП с графическим интерфейсом. Интуитивно понятный графический интерфейс – графический, не текстовый. Длинные параграфы текста должны остаться в документации и online-поддержке. Исключением являются ВПП, содержащие заметки для разработчика. Большинство шаблонных ВП содержат инструкции по редактированию в свободных метках. Всегда удаляйте эти метки, после того как закончили редактирование. Это значительно упростит жизнь.

> ***Правило 3.21 Используйте лаконичные, интуитивно понятные метки для элементов управления и встроенный текст***

> *Правило 3.23 Указывайте значения по умолчанию и единицы измерения в скобках, в конце собственных меток*

Метки элементов управления – самый заметный текст в приложениях LabVIEW. Очень важно, чтобы они были лаконичными и понятными. Добавляйте значения по умолчанию и единицы измерений в скобках для всех элементов управления, представляющих физические параметры. Используйте подписи, когда необходимо добавить длинную фразу к элементу графического интерфейса. Подписи – альтернатива меткам, они видны на лицевой панели, только когда выбрана опция **Visible Items ⇒ Caption**.

Правило 6.5 Вводите описание элементов управления

Правило 6.23 Вводите описания для составляющих массивов и кластеров

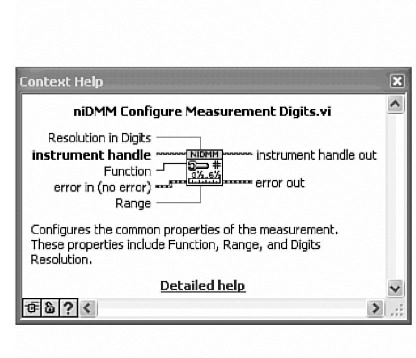
Введите одну или две строки описания назначения каждого элемента управления, типа данных, значения по умолчанию, диапазона; за исключением случаев, когда это понятно из метки элемента. Массивы и кластеры зачастую формируют инфраструктуру данных в приложении. Полезно ввести описание ячеек и элементов, прежде чем они распространятся по всему приложению. Кроме того, разместите небольшие ярлыки с короткими фразами – описаниями элементов управления ВП с графическим интерфейсом. Эти всплывающие подсказки становятся видны, когда вы наводите курсор на элемент управления. Длинные фразы и описания приберегите для руководства пользователя.

Правило 3.39 На ВП всегда должна быть справочная кнопка или пункт меню

Всегда включайте меню или кнопку **Help** (Помощь) в ВП с графическим интерфейсом, с элементами, позволяющими пользователю получить помощь. К таким элементам относятся **Show Context Help** (Показать контекстную справку), online-документация или скомпилированные файлы подсказок. Элемент **Show Context Help** открывает окно контекстной справки, которое отображает все описания, доступные для элементов управления, индикаторов и ВП по мере того, как пользователь наводит курсор на них. Кроме того, LabVIEW может открыть online-документы многих форматов, используя браузер, приложения чтения текста. Online-документы обсуждаются в разделе 9.4. Меню прогона программы может быть настроено через **Edit** ⇒ **Run-time Menu**.

Меню не подходят для диалоговых ВП. Вместо этого лучше разместить кнопки для требуемых действий, например кнопку **Help** или **?**, которая вызовет желаемую документацию. Создайте также меню быстрого доступа для всех действий, связанных с определенными элементами управления. Меню быстрого доступа создаются выбором **Advanced** ⇒ **Run-Time Shortcut Menu** ⇒ **Edit** из меню быстрого доступа элемента.

На рис. 9.1а приведена лицевая панель **niDMM Configure Measurement Digits VI**, ВПП, который настраивает модульный инструмент. В открытом окне текстовой справки видно описание ВП, иконка, терминалы. Метки элементов управления лаконичны и понятны. Только кластер **error in** содержит значение по умолчанию в скобках. Значение по умолчанию и диапазон **Resolution in Digits** (Разрешение) и **Range** (Диапазон) зависят от значения **Function** и не могут быть указаны в метке. Однако значение по умолчанию **Function** известно и должно быть указано. Значения по умолчанию и размерности особенно важны в ВПП-драйверах, потому что помогают разработчику быстро решить – соединить ли соответствующие терминалы или оставить значение по умолчанию.



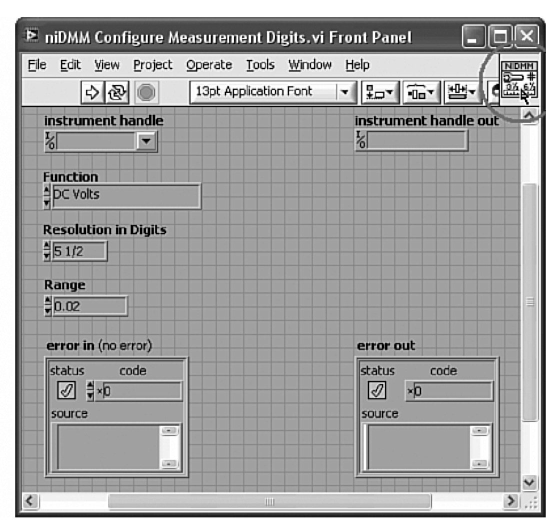


*Рис. 9.1а. **niDMM Configure Measurement Digits VI** – ВПП, который настраивает модульный инструмент. Метки элементов управления лаконичны и понятны. Значения по умолчанию зависят от значения **Function** и не могут быть указаны в метке*

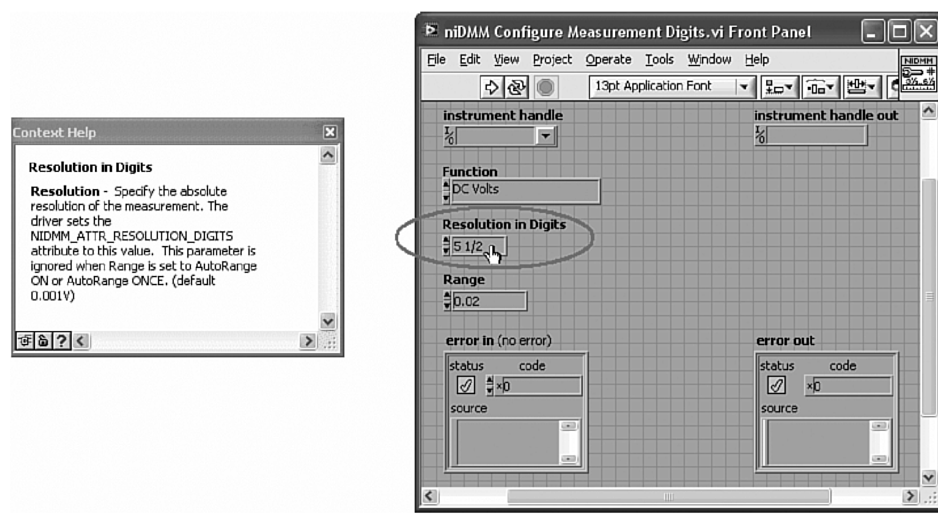


*Рис. 9.1б. Описание **Resolution in Digits** показано в окне контекстной справки. Все элементы управления имеют описания*

На рис. 9.1б показана лицевая панель **niDMM Configure Measurement Digits VI** с окном контекстной справки, показывающим описание **Resolution in Digits**. Окно контекстной справки обновляется по мере движения курсора. Все элементы этого ВПП имеют описания. Это помогает разработчикам понять программируемые функции устройства.

Рисунок 9.2а содержит лицевую панель **Configure Test Instrument VI**, диалоговое окно, которое позволяет пользователю настроить мультиметр. Метки объектов лицевой панели лаконичны и понятны, за исключением **Please Select Instrument to Configure**. Такая длинная фраза, скорее всего, должна была стать подписью, а не меткой. Оставляйте метки лаконичными по Правилу 3.21. Обратите внимание, что ни у одной метки нет значения по умолчанию. Значения по умолчанию не нужны для меток ВП с графическим интерфейсом, потому что элементы обычно уже содержат значения по умолчанию, когда приложение загружается, если только они программно не переписаны с помощью Property Node или локальных переменных. Однако полезно иметь эти значения записанными в метки управляющих элементов. Все элементы управления на лицевой панели **Configure Test Instrument VI** имеют описания и всплывающие подсказки. Однако описания не доступны пользователям до тех пор, пока они не узнают, как открыть окно контекстной справки комбинацией клавиш **Ctrl+H**. Так что работа с этим приложением требует навыков работы в среде LabVIEW.

На рис. 9.2б расположена лицевая панель того же приложения, что и на рис. 9.2а, но несколько улучшенная. Длинная фраза **Please Select Instrument to Configure** (Выберите, пожалуйста, прибор, который нужно настроить) заменена на лаконичное **Instrument** (Прибор). Длинные фразы, в метках или подписях обычно не нужны в диалогах. Стиля лицевой панели и элементов управления должно быть достаточно для того, чтобы пользователь понял, что панель интерактивна и от него требуются действия. Наконец, добавлена логическая кнопка **Help**. Это элемент используется для запуска окна контекстной справки, что дает пользователю возможность увидеть описание элементов ВП. Блок-диаграмма Configure Test Instrument VI представлена в разделе 9.2.

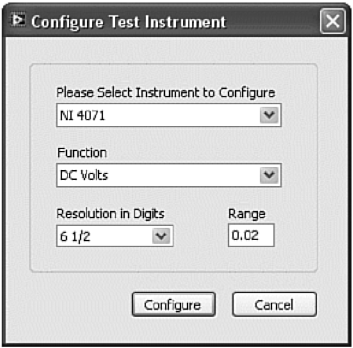
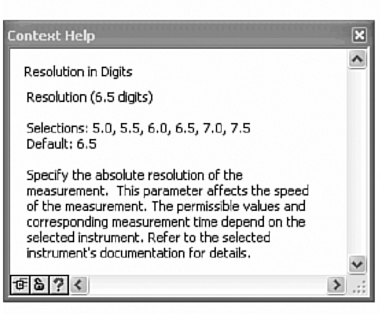


*Рис. 9.2а. Диалог позволяет пользователю настроить устройство. Надпись **Please Select Instrument to Configure** слишком длинная для метки. Нет очевидного способа открыть окно контекстной справки*



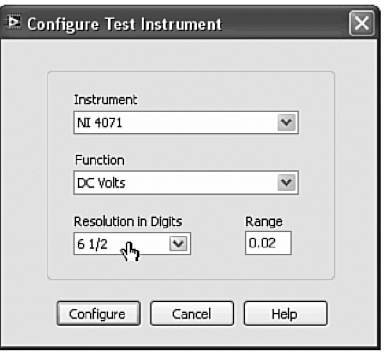


*Рис. 9.2б. **Configure Test Instrument VI** была улучшена введением лаконичных меток и кнопки **Help**, позволяющей вызвать окно контекстной справки*

9.2. Блок-диаграмма

Документация *блок-диаграммы* состоит из меток элементов, свободных меток и конструктов, содержащих текст. В этом разделе рассматривается техника документирования блок-диаграммы.

Правило 9.1 Оставляйте метки терминалов видимыми на блок-диаграмме

Единственный недооцененный источник документации – собственные метки элементов управления и индикаторов. В предыдущем разделе была показана важность создания лаконичных и понятных меток для всех элементов управления и индикаторов. Эти метки являются основным источником документации на блок-диаграмме, но только если они видны. Огромное преимущество концепции потока данных LabVIEW состоит в том, что вы можете проследить путь данных от источника до пункта назначения. Если вы не можете определить терминалы, значительная часть этого преимущества пропадает. Рассмотрение блок-диаграммы влечет за собой переключение между лицевой панелью и блок-диаграммой в попытках определить каждый терминал и зафиксировать метки в памяти. В случае блок-диаграммы, удовлетворяющей правилам хорошего тона, это упражнение становится абсолютно ненужным.

Можно подумать, что поддержка видимых меток происходит автоматически. В конце концов, собственные метки видны на блок-диаграмме по умолчанию. Однако из того, что я видел, могу сделать вывод, что многие разработчики прячут метки или даже оставляют их пустыми. Такая практика ставит меня в тупик, потому что LabVIEW автоматически присваивает имя каждому элементу. Я подозреваю, что разработчики не хотят, чтобы метки занимали место на блок-диаграмме, и прячут их или оставляют пустыми. Поэтому очень важно делать метки лаконич-

ными. В главе 3 содержатся примеры лаконичных и понятных меток, которые облегчают создание хорошей документации.

Правило 9.2 Используйте комментарии в свободных метках

Правило 4.18 Добавляйте метки для длинных проводников

Правило 4.37 Добавляйте метки-пояснения к левым терминалам сдвиговых регистров

Правило 8.20 Помечайте каждый цикл в левом верхнем углу

Правило 9.3 Помечайте каждую вложенную диаграмму каждой структуры с вложенными диаграммами

Правило 9.4 Помечайте алгоритмы, константы и узлы Call Library Function Nodes

Правило 9.5 Используйте простой черный шрифт размера 13 по умолчанию для всего текста на блок диаграмме

Свободные метки являются аналогом комментариев в обычных текстовых языках программирования. Используйте их, выбрав место на ваших диаграммах и просто щелкнув маркировочным инструментом или щелкнув дважды автоматическим инструментом в любой пустой области лицевой панели или блок-диаграммы. К свободным меткам применимомо одно общее правило: маркируйте каждую вложенную диаграмму структуры со многими вложенными диаграммами, за несколькими исключениями вроде структуры Case, но об этом позже. Кроме того, полезно прийти к соглашению о месте расположения метки. В Bloomy Controls мы пришли к соглашению помещать метки в левом верхнем углу большинства структур, как это видно на многих иллюстрациях в этой книге. Мы используем шрифт по умолчанию (простой шрифт размера 13) в наших приложениях и слегка оттененный желтый фон. Применяйте свободные метки для маркировки длинных проводников данных. Используйте больше чем один знак для указания направления потока данных, а фон установите белым. Похожим образом маркируйте проводники данных, выходящие из левого сдвигового регистра.

Маркируйте математические алгоритмы и константы. Обратите внимание, что математически алгоритмы – первые кандидаты в ВПП. Константы могут иметь собственные метки, как элементы управления или индикаторы. Выберите в контекстном меню (щелчок правой кнопкой мыши на константе) **Visible Items ⇒ Label** (Видимые элементы ⇒ Метка) и введите желаемый текст. По умолчанию иконка и описание Call Library Function Node (Узел вызова динамической библиотеки) не очень информативны. Полезно пометить каждый узел Call Library Function Node именем вызываемой DLL. Или можно выбрать формат **Names** вызовом **Name Format ⇒ Names** в контекстном меню. Тем самым изменится размер иконки, на которой появится имя вызываемой DLL. До этого разработчикам приходилось просматривать конфигурационный диалог, что бы узнать имя вызываемой функции.

Правило 9.6 Оставляйте заметки для команды разработчиков

Другой способ хорошего использования свободных меток – оставлять заметки другим членам команды разработчиков. Коммерчески доступные инструменты управления исходным кодом очень полезны в проектах с большим числом разработчиков. Эти инструменты обычно напоминают о комментариях, когда проверяют исходные файлы. Кроме того, инструмент LabVIEW VI Differencing tool может определить изменения в коде между последними редакциями. Однако использование свободных меток позволяет размещать комментарии непосредственно на блок-диаграммах, вблизи от исходного кода, к которому вы хотите привлечь внимание. Эта техника дополняет и улучшает традиционный исходный код.

В этом сценарии данная техника используется для улучшения взаимодействия разработчиков, когда они работают на одном компьютере по очереди, вместо того чтобы работать параллельно. Такое случается, когда только один компьютер имеет доступ к какому-то важному инструменту или оборудованию и защищен брандмауэром (firewall). И в самом деле, я работал над проектами, которые разрабатывались несколькими людьми как в рабочие, так и в выходные дни. Каждый разработчик пишет замечания, описывая серьезные изменения, произошедшие в разрабатываемом сегменте приложения. Метки могут быть выделены цветом, в зависимости от смены, сделавшей их. Автор также добавляет к метке данные и свои инициалы.

Некоторые источники рекомендуют свободно применять метки. Я бы все-таки использовал термин *выборочно* вместо *свободно*. Свободные метки никогда не должны доминировать на блок-диаграмме. В некоторых ситуациях слишком большое количество текста может ухудшить наглядность блок-диаграммы. Например, если вы используете стандартный шаблон, как Classic State Machine, и добавите параграфы текста, объясняющего принципы его работы, в метки на блок-диаграмме, вы за меткой можете не увидеть самого шаблона. Вместо этого я рекомендую использовать относительно лаконичные комментарии в выбранных местах, как описано выше. Длинные объяснения должны находиться в файлах документации, руководстве пользователя, файлах справки или во внешних источниках. Если же длинные комментарии на блок-диаграмме необходимы, то разместите их так, чтобы не закрывать блок-диаграмму, или используйте строковые константы с полосой прокрутки. Строковые константы могут быть использованы так

же, как и свободные метки, с той разницей, что у них есть вертикальная полоса прокрутки. Выберите в выпадающем меню **Visible Items ⇒ Vertical Scrollbar**.

Правило 6.16 Свободно используйте перечни во всех приложениях

Правило 9.7 Используйте пронумерованные типы данных со структурами Case

Пронумерованные элементы управления, или перечни, ставят в соответствие с текстовыми элементами численные значения, по аналогии с текстовыми списками или меню. Однако в отличие от списков, когда перечень соединяется с селектором структуры Case, метки селектора заменяются на текстовые метки, соответствующие элементам перечня. Поэтому перечни описывают функции каждого случая в структуре Case понятными словами. Раздельные метки для документирования каждого случая структуры Case согласно Правилу 9.3 становятся при использовании перечней ненужными.

В примере, представленном на рис. 9.3, **Test to Run** – это численный элемент управления, используемый для выбора запускаемого теста. Он соединен с селектором структуры Case. Каждый случай структуры Case содержит одну или более ВПП, которые осуществляют выбранный тест. На рис. 9.3а **Test to Run** является текстовым списком, соответствующие метки селектора – целыми числами. Свободные метки используются в каждом случае Case, чтобы связать номер с названием теста. Это требует определенных усилий. Поскольку структура Case изменилась, то необходимо настроить несколько элементов. На рис. 9.3б в качестве селектора используется перечень. Метки селектора содержат имена тестов, свободные метки не нужны. Кроме того, структуру Case легко поддерживать, просто изменяя значения перечня. Таким образом, перечень является центральным определением структуры случаев и документации.

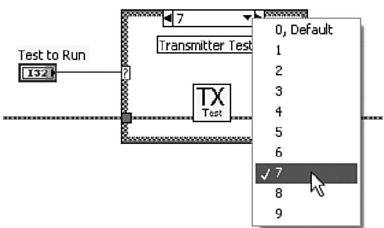


Рис. 9.3а. Структура Case имеет численный селектор, разработчику требуется переводить числа в названия тестов

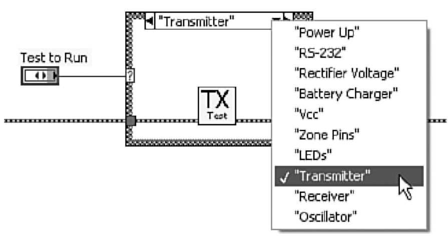


Рис. 9.3б. Пронумерованная структура Case отображает имена тестов в селекторе

Правило 6.17 Сохраняйте перечни в качестве определения типов (typedef)

Обычно используется несколько копий перечня на блок-диаграмме в качестве как констант, так и элементов управления, и текстовые элементы могут изменяться. Поэтому я рекомендую сохранять все перечни как typedef (тайпдеф). В этом случае изменения, сделанные через окно Control Editor, будут автоматически применены ко всем копиям перечня. Списки необходимо сохранять как строгие тайпдефы. Более детально эти вопросы обсуждаются в главе 6. Пронумерованная структура Case также является основным элементом шаблона конечного автомата, обсуждавшегося в главе 8.

В LabVIEW есть опция **Show subVI names when dropped**, доступная из **Tools ⇒ Options** (Инструменты ⇒ Опции) выбором **Block Diagram** из списка **Category**. Я рекомендую по умолчанию отключать эту опцию. Все ВПП должны быть легко узнаваемы по уникальной иконке и имени, видимому в окне контекстной справки. Я рекомендую держать это окно открытым во время настройки, потому что в нем содержится ценная информация, а именно – описание ВП, входов, выходов, свойств и типов данных, помимо имени ВП. Скрытие имен функций и ВПП уменьшает количество ненужного текста на блок-диаграмме, что экономит место и улучшает читабельность приложения.

Формат имени узлов свойств Property Node и методов Invoke Node влияет на размер иконки на блок-диаграмме. По умолчанию используются короткие имена, эта опция определяется через выпадающее меню выбором **Name Format ⇒ Short Names**. По аналогии с ВПП, длинное имя свойства или метода может быть определено через окно контекстной справки. Поэтому лучше использовать короткие имена, а не удалять их совсем.

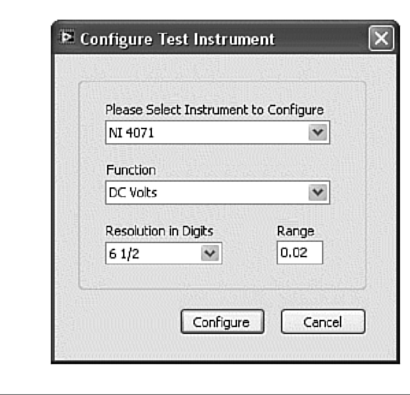
Правило 6.21 Используйте массивы для данных с множеством значений; используйте кластеры для объединения множества различных элементов

Когда вам требуется тип данных с множеством значений, чаще всего лучше подходит кластер, а не массив. Массивы обычно используются, когда все элементы – одного типа, тогда как кластеры используются для определения структур, состоящих из данных различного типа. Однако вы не можете документировать отдельные элементы массива. На блок-диаграмме все функции работы с массивами получают доступ к элементам массива только по его номеру, тогда как каждый элемент кластера имеет свою собственную независимую метку и описание. Проводник данных типа кластер также может быть объединен и разъединен по имени элементов соответствующими функциями. Если номер элемента массива полностью определяет данные, с которыми вы работаете, тогда используйте массивы. Если же требуются уникальные метки и описания для каждого элемента – используйте кластеры. Более подробно разница между кластерами и массивами обсуждается в главе 6.

Правило 6.26 Всегда используйте функции Bundle и Unbundle by Name

На блок-диаграмме используйте функции Bundle и Unbundle by Name, чтобы получить доступ к элементам кластера. Эти функции предоставляют больше возможностей и лучшую читаемость по сравнению с функциями Bundle и Unbundle. Так вы можете получить доступ к любому элементу кластера в любом порядке. Также эти операции не приведут к ошибке, если кластер изменится, если только не изменятся сами элементы, к которым вы получаете доступ таким образом. Самое важное, метки с именем однозначным образом определяют элементы кластера и дают полезную информацию. Это еще одно доказательство того, что метки являются важным источником документации на блок-диаграмме. Поскольку функции Bundle и Unbundle by Name имеют размер такой же, как и самая длинная метка элемента кластера, сохраняйте метки лаконичными и понятными.

Configure Test Instrument dialog VI рассматривается дальше на рис. 9.4. Блок-диаграмма на рис. 9.4а вообще не содержит документации. Поскольку все метки



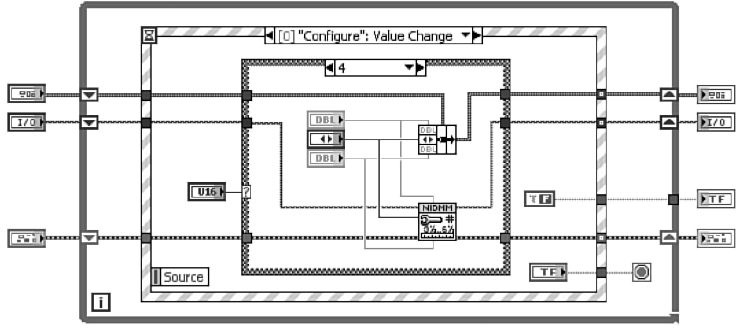


Рис. 9.4а. Блок диаграмма и лицевая панель Configure Test Instrument dialog VI, на блок-диаграмме нет документации

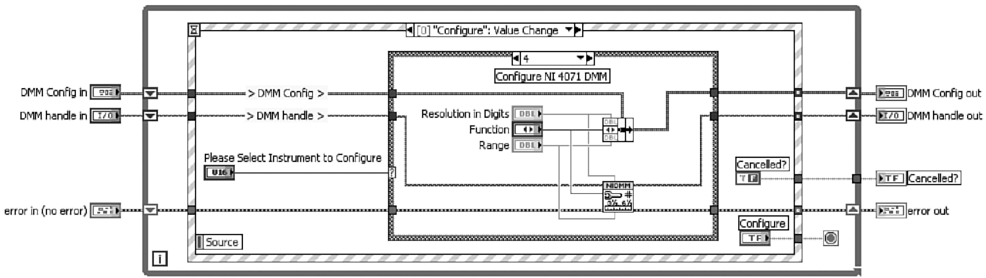


Рис. 9.4б. Этот вариант содержит документацию, состоящую из видимых меток терминалов, меток на проводниках данных у сдвиговых регистров и меток в каждом случае Case. Однако блок-диаграмма стала шире из-за слишком длинных меток

терминалов на блок-диаграмме не видны, свободные метки в структуре Case и около сдвиговых регистров отсутствуют. Также функция Bundle содержит загадочный терминал. Блок-диаграмма на рис. 9.4б содержит множество улучшений по сравнению с рис. 9.4а. Во-первых, видны метки терминалов на блок-диаграмме. Появились метки на проводниках данных возле сдвиговых регистров и в каждом случае структуры Case. Обратите внимание, что блок-диаграмма на рис. 9.4б значительно шире, чем на рис. 9.4а. Это объясняется длинными метками терминалов. Возможно, что это одна из причин, по которой разработчики прячут метки. Помните о том, что метки должны быть лаконичными и понятными. В частности, метка **Please Select Instrument to Configure** слишком длинная. Кроме того, обратите внимание, что стиль меток на переднем плане по сравнению с прозрачным – менее удобный.

На рис. 9.4в показана оптимальная документация для этого ВП. Текстовый список **Please Select Instrument to Configure** был заменен перечнем **Instrument**. Перечень поддерживает те же имена текстовых элементов, что и список до этого. Однако теперь имена инструментов появляются в области селектора структуры Case. Таким образом, осуществляется документация случаев структуры Case и снимается необходимость в свободных метках в каждом случае. Все метки лаконичные и занимают мало места на блок-диаграмме. Функция Bundle заменена функцией Bundle by Name. Имена терминалов упрощены, что облегчает поддержку изменений в кластере и работу с проводниками данных. Наконец, все метки на блок-диаграмме имеют прозрачный фон.

На рис. 9.4г показана измененная лицевая панель и блок-диаграмма, появились кнопка **Help**, событие **Help Value Change** и окно контекстной справки. Когда пользователь нажимает кнопку **Help**, то запускается событие **Help Value Change** и открывается окно контекстной справки, в котором отображается описание всех элементов на блок-диаграмме по мере того, как пользователь водит курсором по объектам.

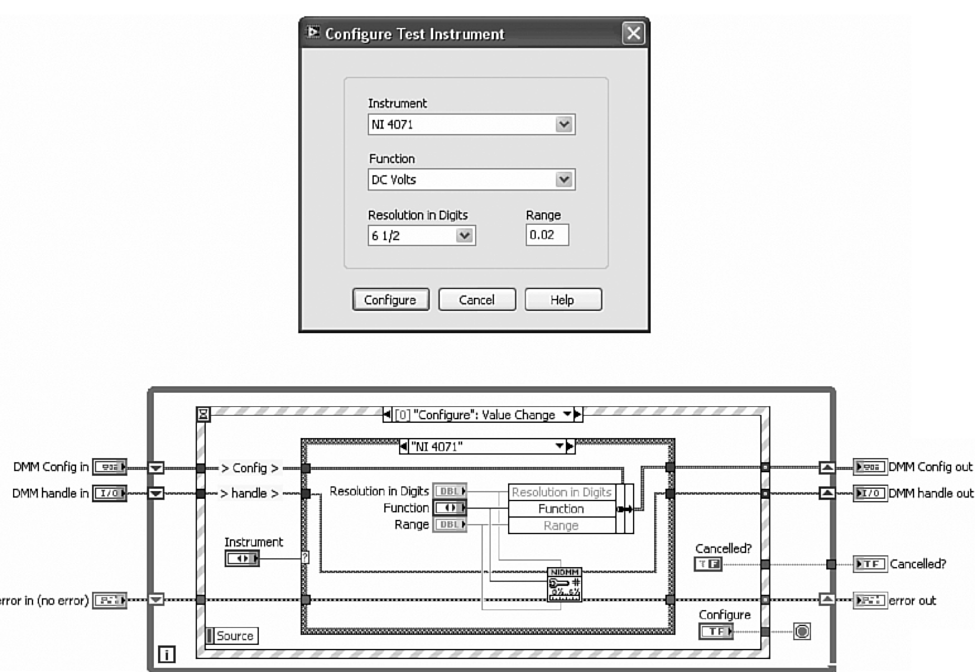


Рис. 9.4в. Сделано несколько улучшений, включая лаконичные метки, пронумерованную структуру Case и функцию Bundle by Name

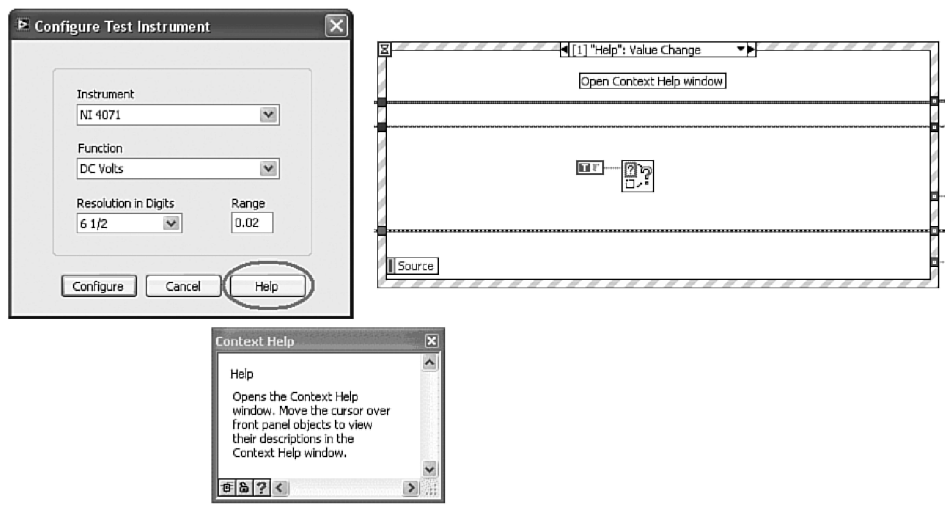


Рис. 9.4г. Кнопка **Help** запускает событие **Help Value Change**. Появляется окно контекстной справки с описанием элементов блок-диаграммы

9.3. Описание иконки и ВП

Иконки и описания ВП составляют главную форму документации в LabVIEW. Эти важные элементы усиливают привлекательность использования этих ВП, используя графику и текст. Иконка графически представляет ВПП на блок-диаграмме. Для лучших результатов создавайте иконки, сочетающие легко узнаваемые символы с цветом и текстом. Описания появляются в окне контекстной справки, когда пользователь наводит курсор на иконку ВПП. Создавайте описания минимум в два-три предложения, которые объясняют назначение каждого ВПП.

Правило 4.10 Создавайте смысловые иконки и связные описания для каждого ВПП

Правило 9.8 Создавайте иконки и описания во время разработки исходного кода

Многие разработчики считают, что разработка иконок ВП и описаний имеет низкий приоритет по сравнению с разработкой работающего исходного кода. У большинства есть намерения вернуться к этим элементам позже, но всегда откладывают: другая задача кажется более важной. Как результат, используются иконки по умолчанию, а не значимые, описания опускаются. Когда же программное обеспечение удовлетворяет операционным требованиям, то настройка иконок и добавление описаний становится еще менее важной задачей по сравнению со следующим проектом. Многие разработчики используют стандартные иконки LabVIEW и не пишут описания потому, что не знают, войдет ли этот ВП в финальную версию приложения. Зачем же тратить время на иконки и описания, если в результате ВП окажется в корзине? На практике, начиная с иконок по умолчанию и пустых описаний, никогда не остается времени на то, чтобы вернуться и настроить их. Читабельность и простота поддержки в таких приложениях приносится в жертву, как и возможность повторного использования этих ВПП. Более того, нужно гораздо больше времени, чтобы создать значащую иконку и связное описание задним числом, чем сразу после создания. Поэтому, создавайте понятные иконки и описания по мере того, как разрабатываете исходный код, пока он еще у вас в голове. Более того, иконки и описания должны рассматриваться как неотъемлемая часть вашего исходного кода. Используйте короткие пути, чтобы ускорить процесс создания иконок, но никогда не жертвуйте уникальностью, значимостью иконок и связностью описаний ВП.

Отношения между связанными описаниями ВПП и ВП обсуждаются в главе 4 «Блок-диаграмма». Более подробное обсуждение иконок, включая примеры и короткие пути создания, находится в главе 5 «Иконка и контакты».

9.4. Online-документация

Интегрируйте ваши одиночные документы, такие как файлы справки и руководства пользователя, в приложение LabVIEW как online-документацию. Online-документы включают в себя HTML, PDF и CHM-файлы. Эти файлы могут быть программно запущены из меню **Help** ВП верхнего уровня с графическим интерфейсом или через логическую переменную **Help**, или **?** из диалогового ВП. Используйте структуру событий, чтобы программно открыть документ, когда выбрана соответствующая опция меню или изменяется значение кнопки **Help**. Это очень похоже на запуск окна контекстной справки, который мы обсуждали ранее.

Язык разметки гипертекста (**HyperText Markup Language – HTML**) – это универсальный язык для создания Интернет-страниц. Сегодня большинство текстовых редакторов и приложений могут создавать HTML-документы. Например, встроенные функции LabVIEW или ВП из набора Report Generation toolkit могут создавать HTML-файлы. Создавайте ваши собственные online-документы, используя HTML-формат, если вы хотите просматривать их через браузер или хотите обеспечить к ним доступ через Интернет. В частности, HTML –это самый простой способ создать документ с множеством перекрестных ссылок, используя гиперссылки. Также, если ваша документация нуждается в постоянных обновлениях, вы можете реализовать ее в форме Интернет-страницы, вместо того чтобы поставлять документацию с приложением. Вы можете запустить HTML-документ в браузере по умолчанию, используя Open URL in Default Browser VI, расположенную в палитре ВПП Help, с помощью команды **Programming ⇒ Dialog & User Interface ⇒ Help**.

Формат переносимого документа (**Portable Document Format – PDF**) был создан компанией Adobe Systems для представления двух- и трехмерных документов в фиксированном расположении и кроссплатформенном формате. PDF-файлы кодируют вид документа вне зависимости от устройств. Их легко просматривать и печатать, используя любой тип компьютеров. Используйте PDF-формат для файлов фиксированного содержания и формата, таких как руководство пользователя. PDF-документы можно просматривать с помощью Adobe Acrobat Reader и в большинстве браузеров. Используйте System Exec VI, расположенный в палитре **Connectivity ⇒ Libraries & Executables** (Взаимодействие ⇒ Библиотеки и исполняемые файлы), чтобы открыть документ в желаемом приложении. Вы должны указать соответствующий путь к приложению и документу или можете использовать Open Acrobat Document VI, расположенный в <vi.lib>\Platform\browser.llb. Этот ВП автоматически сформирует командную строку для использования программы чтения PDF-документов по умолчанию. К несчастью, этот ВП не документирован и не удовлетворяет принципам хорошего стиля и к нему нельзя получить доступ через палитры функций.

Для коммерческих приложений стоит подумать об использовании *скомпилированного файла справки*. **Microsoft Compressed HTML Help – CHM**) является стандартом файлов справки в Windows XP. CHM-файлы жестко индексированы и содержат таблицу содержания, которая обычно находится вне самого текста. Сначала вы создаете HTML-документ, используя ваш любимый текстовый или HTML-редактор, а затем вы компилируете его в CHM-файл, используя компилятор сторонней фирмы, такой как Microsoft HTML Help Workshop или Adobe's Macromedia RoboHelp. Вы можете программно запустить такой файл справки, используя функцию Control Online Help. Соедините вход **String to search for**, чтобы открыть файл справки на определенной теме. Требуемое содержание скомпилированного файла сильно отличается в зависимости от приложения и находится за гранью обсуждаемого в этой главе.

Windows Vista использует Microsoft Assistance Markup Language (AML), новое поколение справки, в которой документы определяются по контексту. Что касается записи в формате AML, то пока она невозможна. Поэтому CHM-формат предпочтительнее в коммерческих приложениях.

Несколько вариантов online-документации, встроенной в Torque Hysteresis VI, показаны на рис. 9.5. К ним относятся контекстная справка, руководство пользователя в формате PDF, HTML и CHM. Эти источники могут быть выбраны в меню **Help**, как показано на рис. 9.5а. На рис. 9.5б показана блок-диаграмма, поддерживающая выбор опций в меню. Каждая выбранная в меню опция соответствует определенному случаю в структуре Case, который запускает соответствующее приложение. Более конкретно, **User Manual (HTML)** использует Open URL in Default Browser VI, чтобы открыть HTML-документ в Интернет-браузере по умолчанию. **User Manual (PDF)** использует System Exec VI, чтобы открыть руководство пользователя в PDF в Adobe Acrobat Reader. **User Manual (CHM)** использует функцию Control Online Help, чтобы открыть CHM-файл. Наконец, опция **Context Help** открывает окно контекстной справки для просмотра описаний эле-

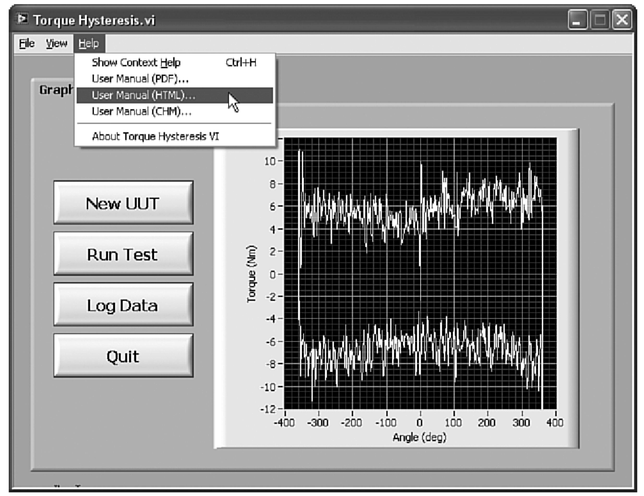


Рис. 9.5а. Лицевая панель Torque Hysteresis VI содержит меню **Help** с возможностью просмотра руководства пользователя в формате PDF, HTML и CHM в дополнение к контекстной справке

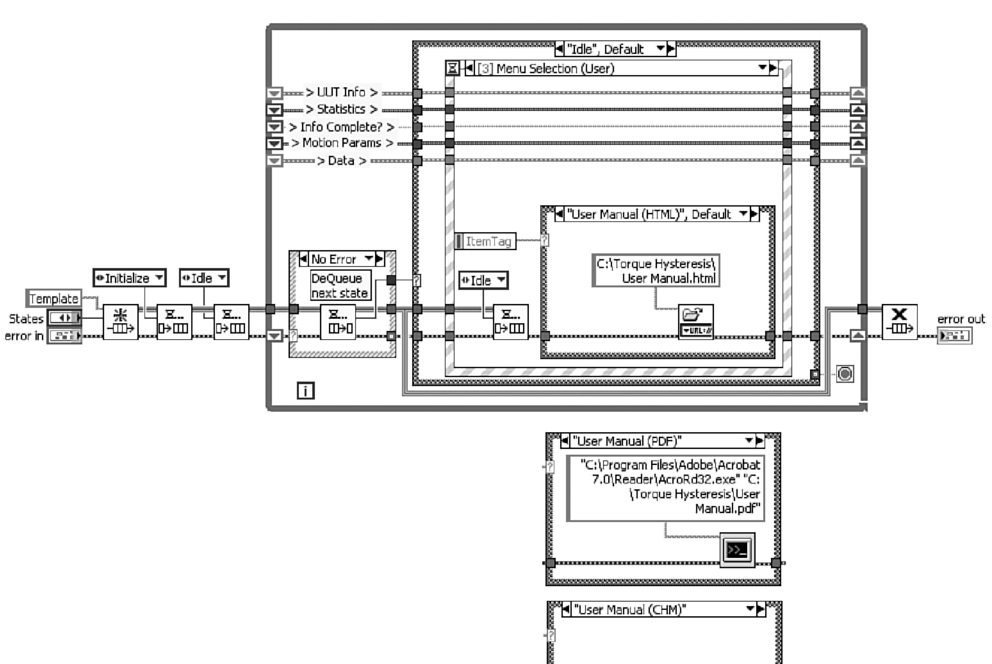


Рис. 9.5б. Блок-диаграмма Torque Hysteresis VI отслеживает события выбора опций в меню, используя структуру событий, и открывает руководство пользователя в соответствующей программе

ментов управления. Эта операция встроенная и для нее не требуется никакого кода на блок-диаграмме.

Правило 9.9 Предоставляйте online-документацию вместе с разрабатываемым приложением

Рассмотрим гипотетическую ситуацию: Вы только что завершили проект для требовательного клиента. Множество изменений в требованиях отразились на увеличении срока разработки значительнее, чем вы предполагали. Проект сильно вышел за рамки бюджета и выполнен гораздо позже срока. Вы проводите окончательную проверку и согласование приложения, и клиент спрашивает: «А где же документация?» Это то, что никогда ранее не обсуждалось, как одно из требований, но часто считается частью приложения. Звучит знакомо? Не паникуйте! Ваш исходный код хорошо документирован согласно правилам стиля из этой главы.

Вы начали с документа спецификации и проектной документации, согласно правилам главы 2. У вас есть детальные описания всех ВП и элементов управления во всех ВП. Ваши диаграммы имеют соответствующие метки, комментарии и текст. Все, что вам нужно сделать, – это выполнить Print VI Documentation. Почти...

Печать документации к ВП (**Print VI Documentation**) – это встроенный инструмент по созданию документации к VI. Он извлекает различные кусочки документации из исходного кода, в зависимости от выбранных параметров, и создает документ в формате HTML, RTF или TXT. Или отправляет документ на печать. Этот инструмент можно запустить, выбрав **File ⇒ Print ⇒ VI Documentation** или программно используя ВП с палитры **VI Documentation**, расположенной в палитре **Programming ⇒ Report Generation**. Также программный доступ ко всем свойствам каждого ВП дает сервер ВП.

Есть несколько особенностей, касающихся использования этого инструмента. Во-первых, множество элементов не относится к руководству пользователя. Если это ваша цель, тогда вам действительно нужно выбрать только те элементы документации, которые относятся к графическому интерфейсу пользователя, включая описание ВП, лицевую панель, элементы управления, их описания и метки. Вы можете отключить все остальное. Также соберите информацию только для ВП с графическим интерфейсом, включая ВП верхнего уровня. Пользователям может не понадобиться видеть ваши внутренние ВПП, пока они не захотят тоже поддерживать исходный код или если они должны иметь возможность видеть какие-то математические алгоритмы. Также существуют некоторые дополнительные элементы, которые не должны попасть в документацию, так как не будут полезны конечному пользователю. Например, Print VI Documentation печатает документацию для каждого элемента на лицевой панели. К этим элементам может относиться кластер ошибок и любой из тех элементов управления или индикаторов, которые вы использовали при разработке приложения в целях диагностики, и теперь они не видны или убраны из видимой области лицевой панели. Наконец, терминалы элементов управления, представленные в документации так, как они обычно появляются на блок-диаграмме. Эти элементы не важны для типичного пользователя. Таки образом, вам может потребоваться произвести значительную чистку документации, связанной с кластерами ошибок, невидимыми элементами и терминалами.

Альтернативным методом автоматизации процесса документирования является использование Control References для получения изображения элементов управления с лицевой панели вместо терминалов и разработка собственной процедуры создания документации через сервер ВП. Этот подход описан в статье, вышедшей в LabVIEW Technical Resource Volume 10, Number 3[1]. Основной момент в том, чтобы тщательно документировать все ваши ВП, тогда эта техника автоматизации будет вам доступна. Я советую всегда предоставлять руководства пользователя или другую форму документации и делать ее доступной через меню **Help** в ВП верхнего уровня во всех разрабатываемых вами приложениях. И вы будете на шаг впереди ваших клиентов.

9.5. Примеры

Как уже говорилось во вступлении к этой главе, читабельность, простота использования и поддержки – вот основные положения, на которые опираются правила стиля и которые отражены в примерах в этой книге. Из-за сильной связи между хорошим стилем, удобочитаемым исходным кодом и документацией примеры в каждой главе тоже связаны. Ниже приведено еще несколько примеров, таких как: ВПП from Selection VI, Filter Test VI, Meticulous Control Descriptions и Temperature Profile Illustration.

9.5.1. ВПП из участка блок-диаграммы

На рис. 9.6 приведено окно контекстной справки для ВП subVI from Selection, который мы обсуждали ранее в главах 3, 4 и 5. Как уже говорилось, этот ВП никогда не будет удовлетворять правилам хорошего стиля без нашего вмешательства. Помимо неподходящих меток терминалов и иконок, отсутствует описание ВП. Тем самым нарушены правила 3.21, 4.10, 5.2, 5.3, 9.7. На рис. 9.6б показан subVI from Selection w Cleanup VI, улучшенная версия того же ВП, в котором стоят соответствующие метки терминалов и добавлена значащая иконка и описание. Но более важно то, что нет способа понять назначение этого ВП из окна контекстной справки или с лицевой панели. Всегда помните о том, что при завершении разработки любых ВП необходимо следовать правилам хорошего стиля.

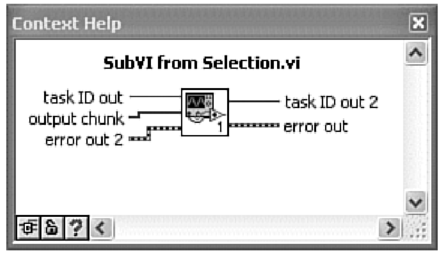


Рис. 9.6а. Окно контекстной справки ВПП from Selection VI. Этот ВП нарушает множество правил, включая неподходящие метки терминалов, неинформативную иконку и отсутствие описания

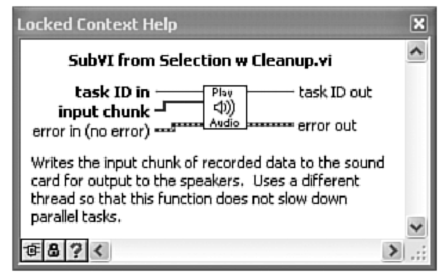


Рис. 9.6б. ВПП from Selection w Cleanup VI обладает значащей иконкой, описанием и подходящими метками терминалов

9.5.2. ВП Filter Test

Приложение для тестирования оптического фильтра, которые мы рассматривали в главе 4, представлено на рис. 9.7. Видно, что процедура, управляющая лазером, правильно использует принцип потока данных, проводники данных удобочитаемы, но документация не адекватна (рис. 9.7а), а именно – кластер разделяется функцией Unbundle, драйвер, управляющий лазером, имеет случайную иконку, и нет свободных меток. Просто посмотрев на ВП, вы не сможете понять, что же он делает. На рис. 9.7б показана улучшенная версия, использована функция Unbundle by name, иконки обрели смысл, добавлено несколько свободных меток. Иконка драйвера лазера содержит стандартный символ светового пучка и текст, описывающий функцию каждого ВП. Также под каждым ВП появились свободные метки, описывающие операции, осуществляемые этим ВП. Функция Unbundle by Name содержит терминалы с понятными метками, соответствующими элементам, входящим в кластер. Число видимых терминалов уменьшено до числа соединенных. Кроме того, добавлены метки на длинном проводнике данных (кластер) и над арифметическими функциями, вычисляющими число шагов в длинах волн. Таким образом, значительно улучшена общая читаемость приложения.

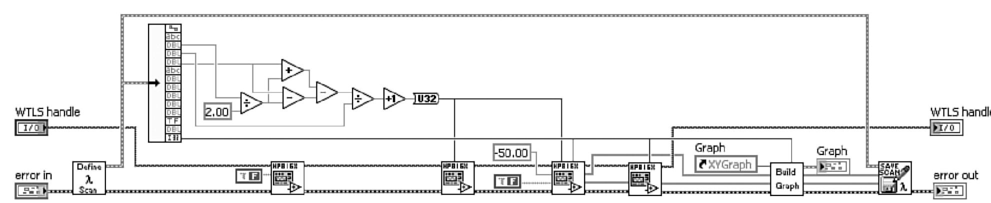


Рис. 9.7а. Процедура измерений параметров оптического фильтра правильно использует принцип потока данных, проводники данных удобочитаемы, но документация не адекватна

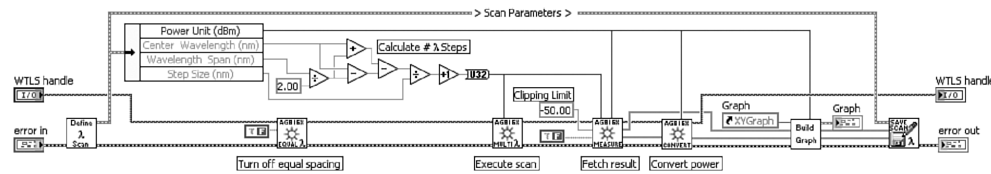


Рис. 9.7б. Читабельность приложения улучшена использованием функции Unbundle by Name, значащих иконок и свободных меток

9.5.3. Тщательное описание элементов управления

Рисунок 9.8 содержит кластер элементов управления, которые используются для настройки последовательных соединений с одним или более датчиком давления. Каждый элемент очень хорошо документирован, метки и описания удовлетворяют всем правилам хорошего стиля. Во-первых, каждый элемент управления имеет лаконичную и понятную метку, значение по умолчанию указано в скобках. Кроме того, описана каждая ячейка кластера. Все описания элементов управления со-

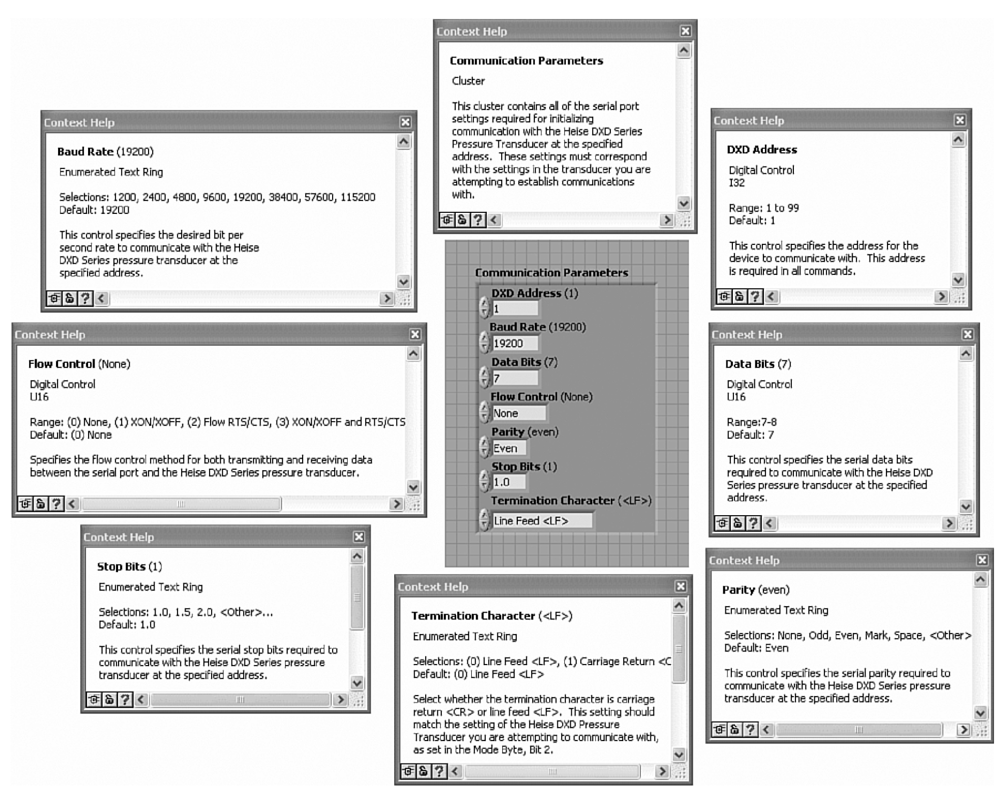


Рис. 9.8. Этот кластер содержит элементы, управляющие последовательными соединениями с датчиком давления. Каждому элементу дано подробное описание

держат информацию о типе данных, допустимом диапазоне, значении по умолчанию, назначении элемента.

9.5.4. Профиль температуры

Документация не ограничивается текстом. Иногда иллюстрации дают больше информации. На рис. 9.9 показана блок-диаграмма ВП, который контролирует температурный профиль и содержит иллюстрацию этого профиля. Эта иллюстрация была создана в графическом редакторе и сохранена как файл изображения. Этот файл затем был размещен на блок-диаграмме с помощью меню: **Edit** ⇒ **Import Picture to Clipboard** (Редактировать ⇒ Импортировать картинку в буфер), а затем **Edit** ⇒ **Paste** (Редактировать ⇒ Вставить).

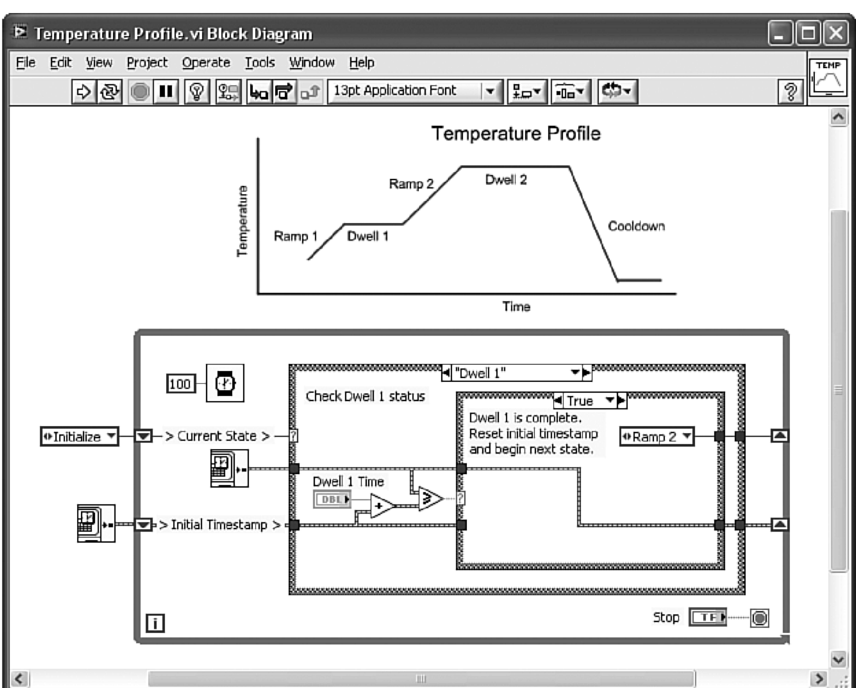


Рис. 9.9. Блок-диаграмма ВП, который контролирует температурный профиль и содержит иллюстрацию этого профиля

Ссылки

1. Chris Megandoff, «One-Click Documentation—Automating Your User Manual Development» LabVIEW Technical Resource 10 (3), 12-15.

Экспертная оценка программы

10

В этой книге представлено много правил стиля, которые могут помочь выработать некий стандарт стиля. Однако эти правила, описания, примеры нуждаются в адаптации. В этой главе мы выясним, как усилить стандартизацию стиля программирования внутри организации. На практике далеко не все согласны со всеми правилами стиля, а даже те, которые согласны, необязательно будут последовательно и эффективно применять их без системы сдержек и противовесов. *Экспертная оценка программы* – это систематические методы критического анализа качества исходного кода, опирающиеся на один или несколько независимых критериев. Пересмотр кода имеет три основные задачи: определение проблем, обмен знаниями и идеями, что улучшает навыки разработчика, введение и разработку стандартов во всей организации. И что более важно – экспертная оценка программы помогает убедиться в высоком качестве программного обеспечения, которое легко понять, поддерживать и которое устойчиво к ошибкам.

Вторичный эффект от пересмотра кода состоит в привлечении дополнительных разработчиков к работе над приложением, что дает некоторую глубину знаний о возможностях применения и развития приложения. Как специалист по интеграции систем, я выяснил, что возможности дальнейшего развития отличают хорошую работу с клиентами от отличной. Когда в будущем понадобится внести какие-то изменения, то очень важно знать, что исходный код – высокого качества и у вас осталось множество вариантов дальнейшего развития. И шансы удовлетворить новые запросы клиентов, без остановки работы надо другими проектами, невероятно возрастают.

Правило 10.1 Усильте стандарты стиля программирования в вашей организации, используя экспертные оценки программы

Правило 10.2 Используйте комбинацию самостоятельных экспертных оценок и экспертных оценок, выполненных компетентными сотрудниками, для получения лучших результатов

Для экспертной оценки вам потребуется один или более компетентный сотрудник, чтобы отдача была эффективной. Экспертные оценки включают координацию и встречи с опытными в программировании на LabVIEW коллегами, знакомыми с принятыми в организации стандартами. Это обычный метод проведения экспертных оценок. Однако объективную информацию можно получить и другими способами. Самостоятельная экспертная оценка программы осуществляется с использованием альтернативных ресурсов, таких как итоговые правила в Приложении Б «Сводка основных правил стиля» и набор инструментов LabVIEW VI Analyzer Toolkit. Самостоятельные экспертные оценки и экспертные оценки коллег дополняют друг друга, для лучших результатов лучше использовать обе эти техники.

10.1. Самостоятельная экспертная оценка программы

Как следует из названия, это техника экспертной оценки вами вашего же кода. Это поможет вам подготовиться к обсуждению кода с коллегами, определить и исправить возможные недочеты, перед тем как отдать код на рецензию другим разработчикам. Автоматический метод опирается на использование дополнения от NI под названием LabVIEW VI Analyzer Toolkit. Кроме того, приведенная в Приложении Б «Сводка основных правил стиля» может быть использована как контрольный список. Просто пройдитесь по списку, что бы убедиться, что ваш ВП удовлетворяет всем правилам.

Правило 10.3 Осуществляйте самостоятельные экспертные оценки программы перед экспертной оценкой

Проведите как минимум одну проверку кода перед каждой экспертной оценкой. Это поможет вам подготовиться к обсуждению, определить и устранить возможные недочеты. Тем самым увеличится и продуктивность экспертной оценки.

И еще кое что о самостоятельных проверках: многие разработчики программ на LabVIEW работают в одиночку. Может случиться так, что вы единственный разработчик в компании или работаете как «вольный стрелок». Тогда самостоятельные проверки – ваш единственный шанс. Такие проверки вместе с жесткой самодисциплиной дадут вам возможность овладеть хорошим стилем программирования.

10.1.1. ВП Analyzer Toolkit

Дополнение от NI под названием LabVIEW VI Analyzer Toolkit (Анализатор ВП) автоматически анализирует ВП с точки зрения стиля и производительности. В нем есть и интерактивный и программный интерфейс. Интерактивный интер-

фейс позволяет вам настроить желаемый тест, записать конфигурацию в файл, запустить тест, просмотреть и сохранить результаты. Кроме того, если вы щелкнете по ошибке в окне **Results**, то ВП Analyzer откроет блок-диаграмму и высветит проблемный участок. Таким образом, вы сможете по очереди разобраться со всеми проблемами. Вдобавок, программный интерфейс позволяет вам разработать приложение, которое осуществляет желаемый тест, с использованием ВП с палитры **VI Analyzer**. Например, вы можете написать приложение, которое проверит все ВП в исходном коде и создаст отчет с результатами проверки. Вы можете настроить приложение на периодическую проверку, а результаты рассылать по электронной почте или публиковать на сайте автоматически.

VI Analyzer содержит более 60 тестов, разбитых по категориям: **Block Diagram** (Блок-диаграмма), **Documentation** (Документация), **Front Panel** (Лицевая панель) и **General** (Общая). Тесты из категории **Block Diagram** проверяют соблюдение нескольких правил из главы 4 «Блок-диаграмма», такие как направление потока данных, наложение объектов и проводников данных. Кроме того, несколько тестов проводят более узкие проверки, не связанные напрямую со стилем. Сюда относится поиск не исполняющихся элементов (так называемого «мертвого кода») и спрятанных в структурах объектов. Тесты из раздела **Documentation** проверяют соблюдение правил из главы 9 «Документация», включая описания ВП и элементов управления, комментарии в свободных метках. Дополнительно VI Analyzer может проверить орфографию текста на лицевой панели и блок-диаграмме. Тесты **Front Panel** проверяют выполнение некоторых правил главы 3 «Стиль лицевой панели», включая выравнивание и наложение элементов. Дополнительно VI Analyzer может проверить значения по умолчанию для элементов списков и массивов, которые обычно являются источниками не эффективного использования памяти. Категория **General** содержит тесты иконки и соединительной панели, свойств файлов и ВП. Тесты иконки и соединительной панели проверяют соблюдение некоторых правил главы 5 «Иконка и контакты». Тесты свойств файлов и ВП включают проверку версии LabVIEW, в которой они созданы, и совместимость свойств и методов в приложении.

LabVIEW VI Analyzer Toolkit был впервые представлен для LabVIEW 7.0. К тому времени в Bloomy Control уже были устоявшиеся стандарты стиля программирования и система экспертных оценок программы, и мы упорно держались за эти стандарты. До тех пор, пока я не нанял и не обучил новую сотрудницу, только что закончившую обучение, я был в стороне от возможностей ВП Analyzer. Обычно экспертная оценка программы новых инженеров занимает много времени, потому что с ними приходится обсуждать множество аспектов стиля. Однако качество кода, разработанного новым членом моей команды, невероятно улучшилось. Удивительно, но все проводники данных в ее коде были четко расположены, не налагались друг на друга, поток данных шел слева направо. Ее код напоминал код куда более опытных разработчиков. Эффективность ее экспертных оценок также заметно улучшилась, а необходимый уровень тренировки уменьшился. Ее секретом был ВП Analyzer Toolkit. И теперь некоторые из моих инженеров используют ВП Analyzer перед экспертной оценкой программы.

Правило 10.4 Используйте VI Analyzer для автоматизации тщательных проверок

Основные плюсы использования VI Analyzer следующие:

1. Он помогает проверять и улучшать стиль программирования.
2. Использовать его может каждый.
3. Процесс автоматизирован.

Важно понимать, что ВП Analyzer может куда больше, чем просто проверить стиль приложения. Множество тестов в ВП Analyzer направлены на поиск ошибок, по аналогии с процессом повторной компиляции. Так, ВП Analyzer содержит категорию **Warnings**, включая **Bundling Duplicate Names**, **Typedef Cluster Constants**, **Hidden Tunnels**, **Reentrant VI Issues**, и многое другое. Все это – неуловимые источники ошибок и некорректной работы, которые трудно определить при ручной проверке.

Правило 10.5 Настройте критерии тестирования в VI Analyzer

1. Установите максимальное число точек приведения типов (**Coercion Dots**) на одном проводнике равным **0**.
2. Установите на время разработки значение Fail Test if Debugging равным **Disabled** (тест **Enabled Debugging**).
3. Число глобальных и локальных переменных задайте равным **0**.
4. Установите опцию **Comment Usage** на **Ensure subdiagrams contain at least one comment each**.
5. Установите опцию **Control Alignment для Pixel Tolerance** на **1**.
6. Отключите тесты диалогов **Dialog Controls** для ВПП и промышленных ВП с графическим интерфейсом.
7. Значение **Connector Pane Pattern** установите на **32** и **0**.

Используйте диалог **Select Tests**, чтобы настроить критерии тестов в ВП Analyzer под свои нужды, и сохраните эту конфигурацию в файл для повторного использования. Установите максимальное число точек приведения на проводнике равным 0, как показано на рис. 10.1. Этот тест, настраиваемый через **Block Diagram ⇒ Performance ⇒ Coercion Dots**, усиливает Правило 4.24 «Избегайте точек приведения типов для массивов и кластеров» и Правило 6.3 «Выбирайте элементы и тип данных, которые облегчают создание согласованной структуры данных в приложении». Значение по умолчанию равно 2, поэтому его необходимо изменить, чтобы отследить каждое нарушение. Если вы не собираетесь скомпилировать исполняемое приложение, отмените также тест **Enable Debugging**, установив значение **Disable**. Большинство экспертных оценок осуществляется на стадии разработки, и отслеживание ошибок необходимо. Во время хотя бы одного прогона кода установите максимальное число локальных и глобальных переменных равным 0, это поможет проверить каждую и понять, действительно ли она нужна. Если ваш ВП большой и сложный и содержит много глобальных или ло-

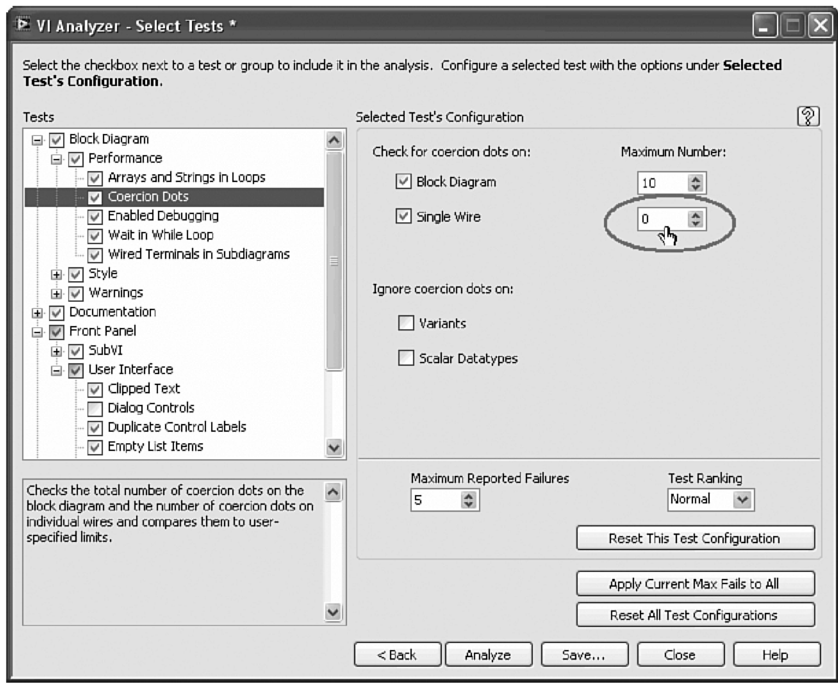


Рис. 10.1. Интерактивное окно VI Analyzer, настроенное на запуск теста с определенными параметрами. Максимальное число точек схождения установлено равным 0, чтобы определить каждое нарушение правила стиля

кальных переменных и точек схождения, подумайте о том, чтобы создать отдельные файлы конфигурации для тестирования только переменных или только точек схождения. Так вы сможете сфокусироваться на одной задаче.

Под панелью **Front Panel ⇒ SubVI**, в тесте **Control Alignment** есть параметр **Pixel Tolerance** со значением по умолчанию **10**. Если вы используете инструменты выравнивания, как рекомендуется в главе 3, вы можете уменьшить это значение до **1**. Также в VI Analyzer есть тест **Dialog Controls**, в категории **User Interface**, который по умолчанию включен. Этот тест проверяет все элементы управления в ВП, чтобы определить, все ли они из палитры **System**. Диалоги применимы только в ВП с графическим интерфейсом, для приложений, которые имеют внешний вид, аналогичный операционной системе. Убедитесь в том, что отключили этот тест для всех ВПП и промышленных приложений, которые используют элементы управления стиля Modern и Classic.

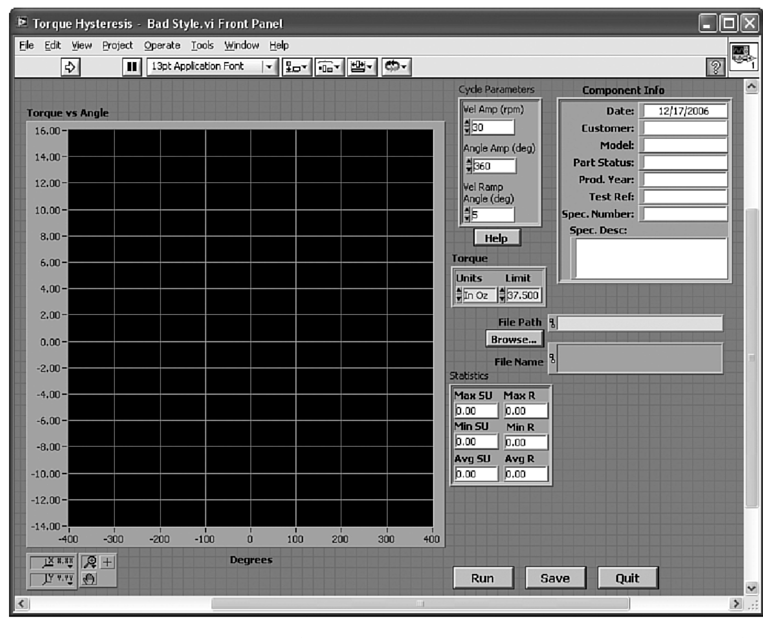
Дополнительно в VI Analyzer есть тест **Comment Usage**, в категории **Documentation ⇒ Developer**. Выберите **Ensure subdiagrams contain at least one comment each**, чтобы убедиться в выполнении Правила 9.3 «Ставьте метки на всех вложенных диаграммах структур со многими вложенными диаграммами». Наконец, в **General ⇒ Connector Pane Pattern** выберите шаблон 4×2×2×4 и один из шаблонов соединительной панели, соответствующий числам от **32** до **0**. Номер 32 поддерживает Правило 5.21 «Используйте шаблон 4××2×2×4 для большинства ВП». Номер шаблона 0 предотвращает появление ошибки, когда вы тестируете ВП верхнего уровня, разработанный в более ранних версиях LabVIEW. До версии 8.0 по умолчанию для большинства ВП использовался шаблон с один терминалом.

Правило 10.6 Устанавливайте очередность каждого теста согласно приоритету каждого правила

Установки очередности тестов определяют порядок, в котором появляются сообщения о нарушениях в окне результатов и символ рядом с именем теста. Установите ранг тестов согласно приоритету правил: для правил с высоким приоритетом ранг **High**, для правил с обычным приоритетом – **Medium**. Поскольку ВП Analyzer предназначен для тестирования не только стиля, но и производительности приложения, по умолчанию ранги тестов значительно отличаются от приоритетов правил, представленных в этой книге. Например, тест **Error Cluster Wired** имеет ранг **Low**. Этот тест проверяет, соединены ли терминалы **error out**, то есть соблюдено ли Правило 7.5 «Отслеживайте все ошибки от всех узлов, имеющих терминалы ошибок». Как уже обсуждалось в главе 7 «Обработка ошибок», отслеживание ошибок – ключевой момент для их обработки, а обработка ошибок влияет на устойчивость приложения к ошибкам. Установите высокий **(High)** ранг этого теста, что отражает важность правила 7.5. Вы можете скачать настройки конфигураций тестов, содержащие рекомендуемые параметры и ранги с www.bloomy.com/lvstyle.

На рис. 10.2а представлена примитивная версия ВП Torque Hysteresis, которую мы обсуждали в предыдущих главах. Эта версия была разработана в предыдущей версии LabVIEW по смягченным стандартам стиля. В этой главе мы постепенно переделаем этот ВП через процедуру экспертных оценок. ВП Analyzer настроен согласно правилам 10.5 и 10.6. На рис. 10.2б представлены результаты тестов в окне VI Analyzer. Результаты отсортированы сверху вниз по приоритету: **!** означает высокий ранг теста, средний ранг никаким символом не обозначается, а символ **i** означает низкий ранг теста. На рис. 10.2в–10.2к содержатся детальные результаты конкретных тестов, включая все ошибки в случае 1 структуры Case, и исправления этих ошибок.

На рис. 10.2в–10.2ж представлены ошибки тестов с высоким рангом. На рис. 10.2в тест **Backwards Wires** определяет направление потока данных (должно быть слева направо). На рис. 10.2г тест **Error Cluster Wired** определяет функции с неподсоединенными терминалами **error out**. На рис. 10.2д тест **Unused Code** определил неиспользуемую константу типа перечень. Она осталась от предыдущей версии функции File Dialog, которая имела дополнительный входной терминал для выбора операционного режима. На рис. 10.2е кнопка **Save** не имеет описания. Проводник данных кластера на рис. 10.2ж возникает внутри структуры кадров (Sequence) слева, передается через туннель, проходит под этой структурой, преж-



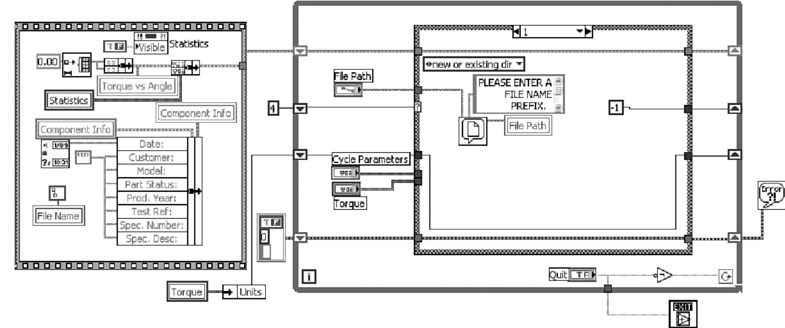


Рис. 10.2а. Лицевая панель и блок-диаграмма примитивной версии ВП Torque Hysteresis перед экспертной оценкой программы

де чем соединиться со сдвиговым регистром цикла While. Это приводит к ошибке в тесте **Wires Under Objects**.

Рисунки 10.2з–10.2к содержат информацию о тестах среднего ранга. На рис. 10.2з ВП Analyzer обнаружил неудобно расположенные комментарии. Короткие комментарии добавлены в каждый случай структуры Case. Кроме того, добавлены два комментария, описывающие функции двух основных структур, как показано на рис. 10.2л. На рис. 10.2и ВП Analyzer обнаружил неправильное написание слова **velocity** (написано как **vel**) в двух метках. И хотя это сокращение намеренно введено разработчиком, сокращение важных терминов в метках элементов ВП с графическим интерфейсом не рекомендуется. Проблема решается расширением

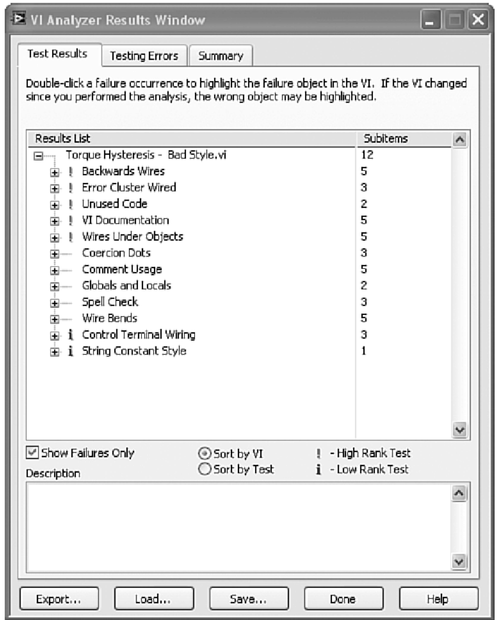


Рис. 10.2б. В окне результатов VI Analyzer показаны результаты проверки и отсортированы сверху вниз по приоритету: **!** означает высокий ранг теста, средний ранг никаким символом не обозначается, а символ **i** означает низкий ранг теста

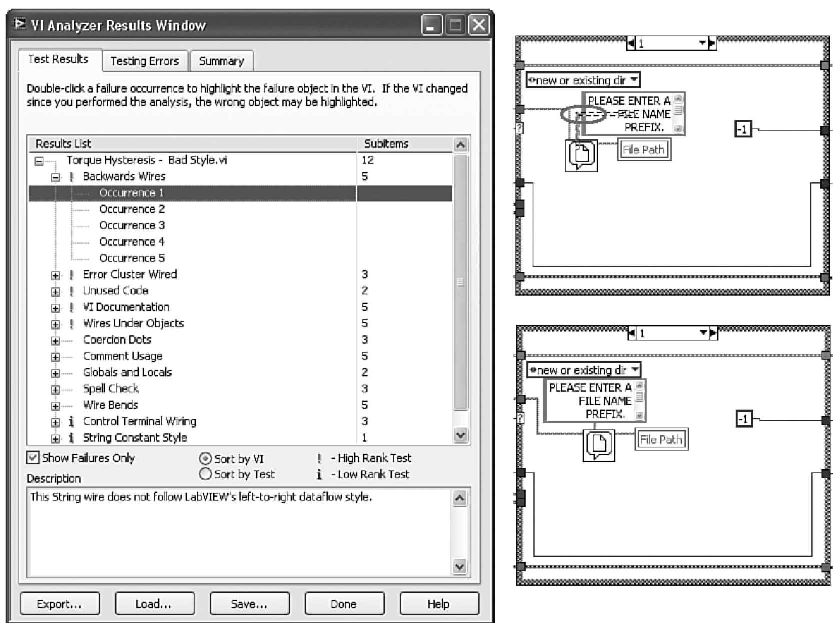


Рис. 10.2в. Строковая константа расположена вверху и несколько правее функции File Dialog, что приводит потоку данных справа налево. ВП Analyzer сигнализирует об ошибке в тесте ***Backwards Wire***

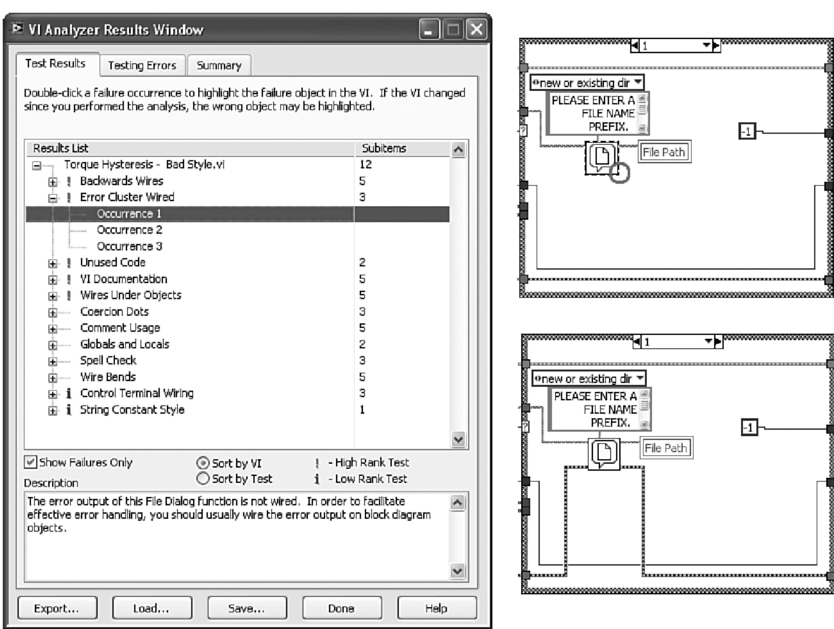


Рис. 10.2г. Терминал ***error out*** функции File Dialog не соединен, что приводит к ошибке в тесте ***Error Cluster Wired***

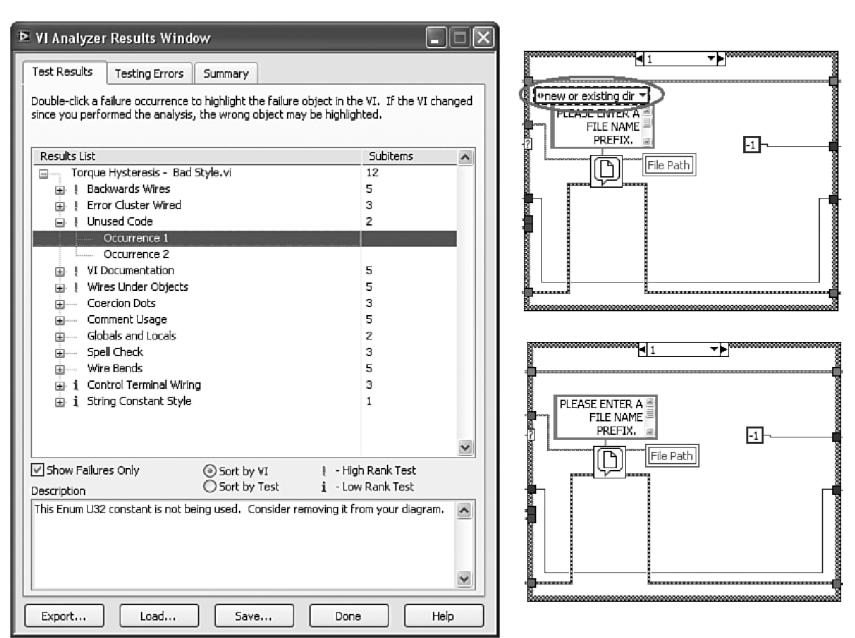


Рис. 10.2д. Случай 1 структуры Case содержит перечень, ни с чем не соединенный, о чем говорит ошибка в тесте ***Unused Code***

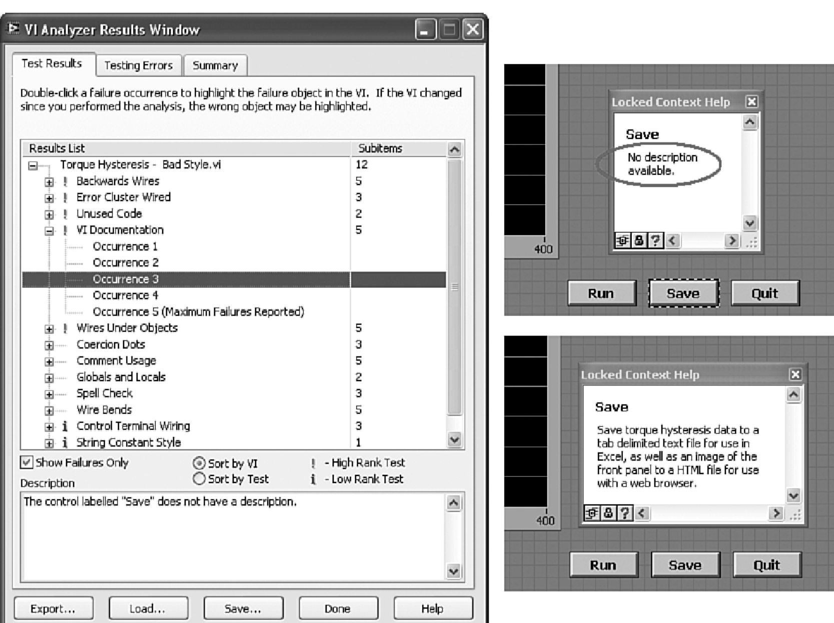


Рис. 10.2е. Несколько элементов не имеют описаний, в том числе кнопка ***Save***, что приводит к множеству ошибок в тесте ***VI Documentation***

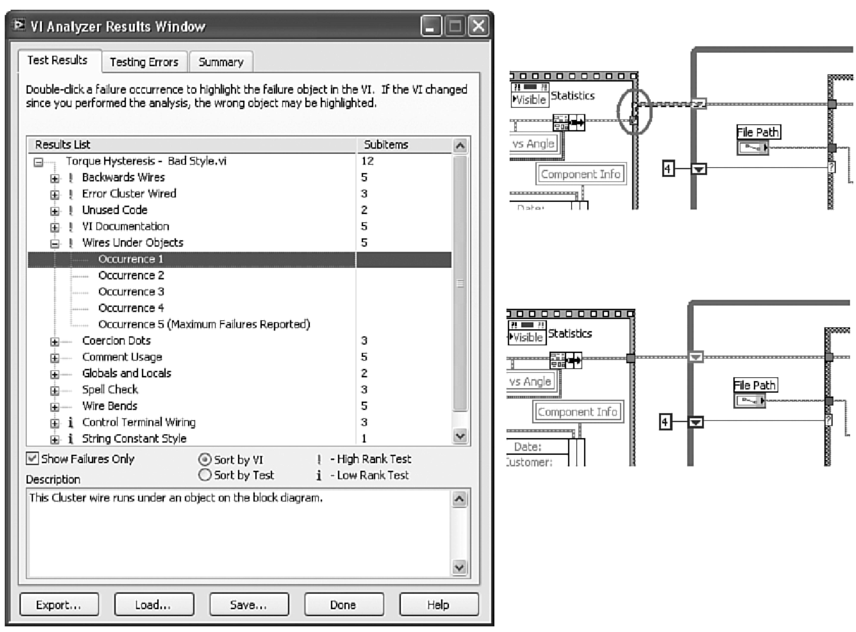


Рис. 10.2ж. Часть проводника данных проходит под структурой Sequence, возникает ошибка теста ***Wires Under Objects***

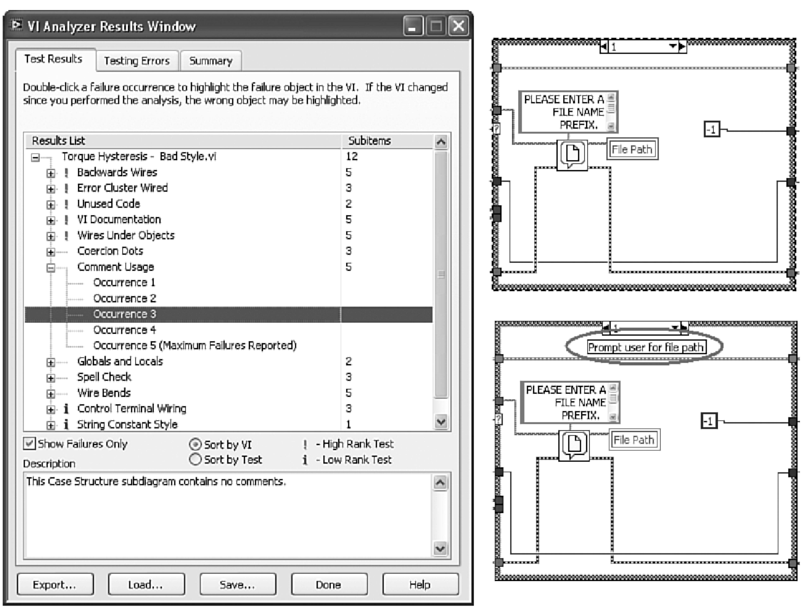


Рис. 10.2з. На блок-диаграмме вообще нет комментариев, в том числе в структуре Case, как результат множество ошибок в тесте **Comment Usage**. *Добавлен короткий комментарий в случай 1*

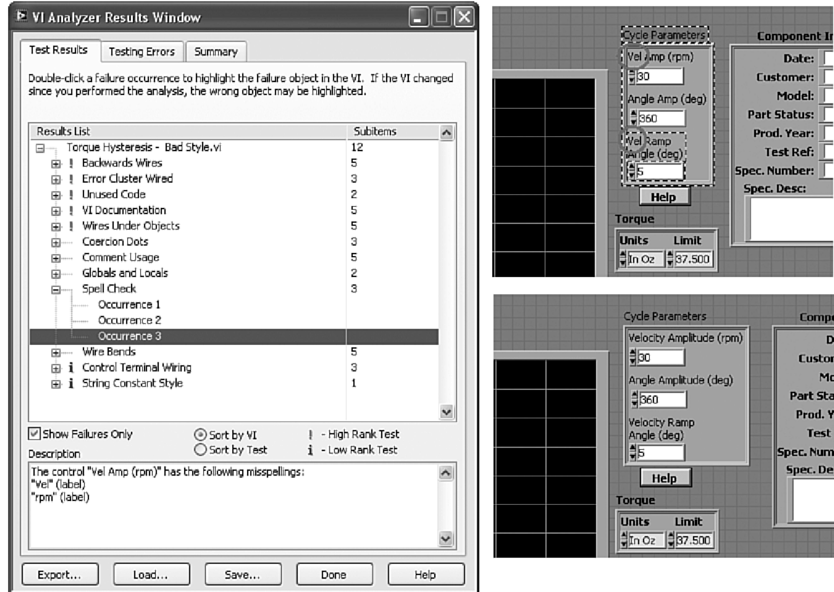


Рис. 10.2и. Уведомление о неправильном написании меток элементов. Проблема в сокращении термина **velocity**

сокращения до **velocity**. По аналогии расширены все метки элементов в кластере **Statistics**, как показано на рис. 10.2л. На рис. 10.2к показаны нарушения теста **Wire Bends**. Очень короткий сегмент, известный как перегиб, определен и исправлен. Другой проводник также имеет 4 изгиба, что превышает максимальное число в 3 изгиба. Эти изгибы по замыслу разработчика позволяют проводнику следовать по границе структуры Case. Поскольку изгибы проводников данных зачастую не случайны, тест **Wire Bends** имеет средний ранг. Однако в данном случае эти изгибы не служат какой-либо цели и просто исправлены на рис. 10.2л.

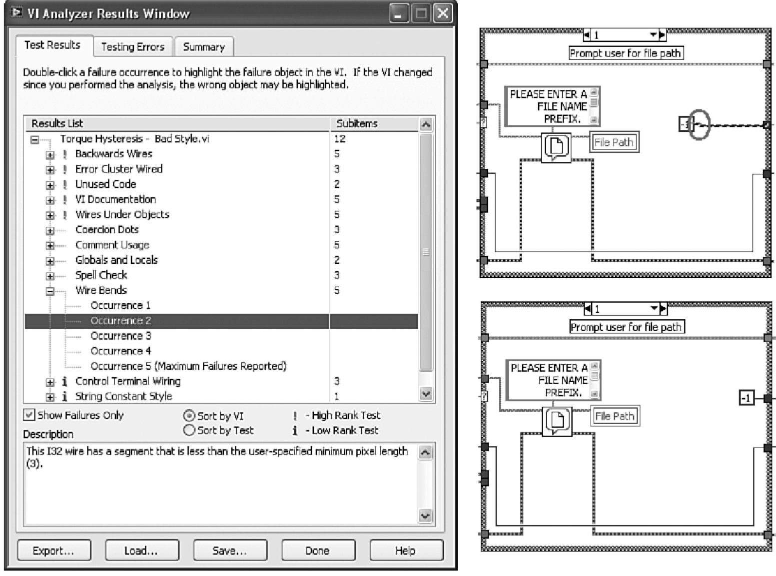


Рис. 10.2к. Тест **Wire Bends** *определил участок проводника с меньше чем тремя пикселями, такой участок называется перегибом*

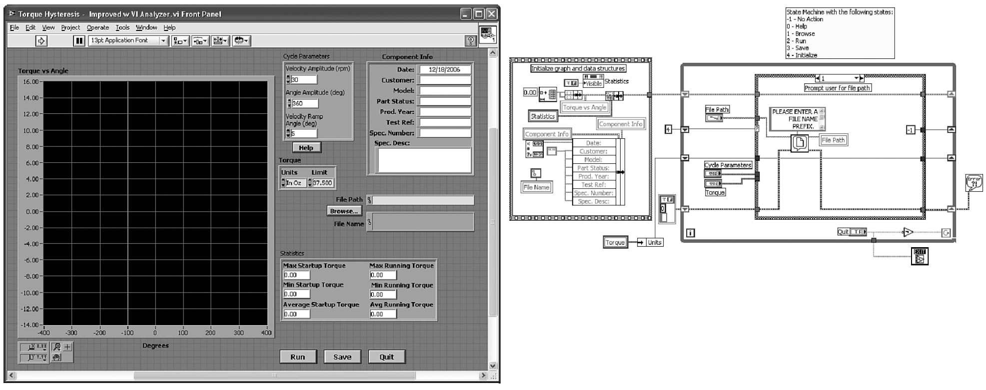


Рис. 10.2л. ВП Torque Hysteresis исправлен согласно рекомендациям ВП Analyzer

10.1.2. Контрольный список ручной проверки

Как было показано в предыдущем разделе, ВП Analyzer является эффективным инструментом для быстрой проверки собственного кода. Однако ВП Analyzer не учитывает многих соображений, касающихся стиля. Например, вы могли заметить, что можно внести множество дополнительных изменений на диаграмме на рис. 10.2 помимо тех, что рекомендует ВП Analyzer. И в самом деле, ВП Analyzer осуществляет примерно 60 тестов, 40 из них относятся к стилю, но эта книга содержит более 200 правил стиля.

Правило 10.7 Используйте контрольный список, чтобы провести всесторонний анализ кода

Контрольный список – это наиболее полный метод пройти по всем правилам, оценить важность конкретного правила и проверить ваш код на соответствие этим правилам. Вы можете использовать сводку правил из Приложения Б в качестве контрольного списка или разработать свой собственный список. Понятно, что просто не хватит времени в ходе экспертной оценки, чтобы оценить все 200 с лишним правил. Подход с использованием контрольного списка достаточно полный и имеет еще одно преимущество – вы можете еще раз пересмотреть каждое правило стиля. Если в вашей организации стандарты отличаются от тех, что представлены в Приложении, то можете создать свой контрольный список.

Давайте теперь устроим самопроверку ВП Torque Hysteresis, вручную применив правила из контрольного списка к улучшенной с помощью ВП Analyzer. Лицевая панель и блок-диаграмма улучшенной Torque Hysteresis Improved w VI Analyzer VI представлены на рис. 10.3а и 10.4а соответственно. Мы начнем с анализа лицевой панели. Лицевая панель на рис. 10.3а нарушает следующие правила из контрольного списка в Приложении Б:

Правило 3.1	Разделяйте логически не связанные элементы с помощью закладок, кластеров, элементов оформления и промежутков
Правило 3.2	Придерживайтесь симметричного расположения элементов
Правило 3.9	Более важные элементы управления в промышленных приложениях должны быть больше по размеру и располагаться ближе к центру окна
Правило 3.10	Ограничивайте количество информации на лицевой панели: на панели не должно быть больше 7 групп из 7 различных элементов; следите за наличием пустого места между группами
Правило 3.19	Не злоупотребляйте форматированием текста
Правило 3.33	Ограничьте количество активных элементов управления
Правило 6.4	Задайте соответствующее значение по умолчанию для каждого элемента управления

Унаследованный приложением графический интерфейс (рис. 10.3а) состоит из одной большой панели, на которой расположены все элементы управления и индикаторы, которые всегда видны. Расположим графический индикатор слева, кластер элементов управления и индикаторов справа, элементы, контролирующие и показывающие путь к файлу, – в вертикальном центре, кнопки управления вдоль нижней линии, кнопки **Help** и **Browse** вместе с кластерами. Приложение было создано до появления современного трехмерного стиля элементов и поэтому использует классические элементы управления и индикато-

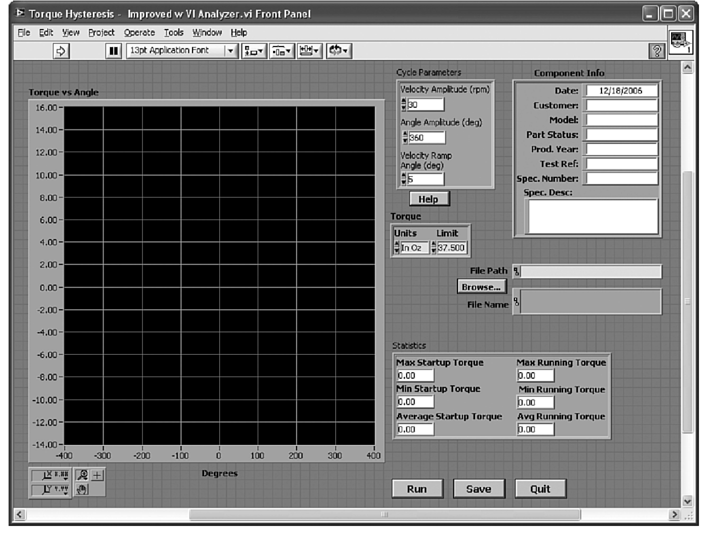
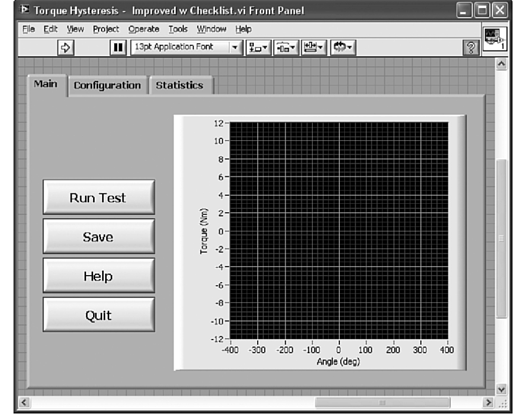


Рис. 10.3а. Лицевая панель Torque Hysteresis – Improved with VI Analyzer VI содержит много нарушений правил хорошего стиля



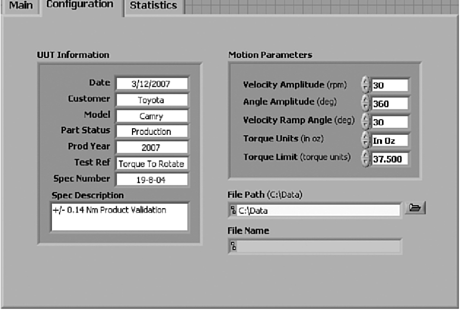


Рис. 10.3б. Лицевая панель изменена в соответствии с правилами из Приложения Б

ры. На рис. 10.3б измененная лицевая панель использует вкладки для группировки наиболее часто используемых элементов на вкладке **Menu**, элементы, используемые для настройки последовательности измерений, расположены на вкладке **Configuration**, а индикаторы, суммирующие полученные результаты, на вкладке **Statistics**. Поскольку это промышленное приложение, классические элементы заменены на современные трехмерные. Элементы на вкладке **Main** увеличены для наглядности и простоты использования. Помимо группировки связанных элементов, вкладки позволяют спрятать неиспользуемые в данный момент элементы, тем самым ограничивается поток единовременно отображаемой информации. На вкладке **Configuration** для меток элементов управления использован полужирный шрифт размера 13, в скобках обычным шрифтом указаны единицы измерения. Также указано значение по умолчанию для элемента выбора пути к файлу.

На рис. 10.4а расположена блок-диаграмма Torque Hysteresis – Improved with VI Analyzer VI. Если просмотреть контрольный список, то можно увидеть нарушения следующих правил:

Правило 4.11	Избегайте петель и изгибов проводников
Правило 4.13	Туннели должны быть расположены на вертикальных границах структур
Правило 4.27	Избегайте структур последовательности без необходимости
Правило 4.34	Избегайте переменных, если задачу можно решить с помощью проводников
Правило 4.37	Добавляйте метки-пояснения к левым терминалам сдвиговых регистров
Правило 7.7	Используйте ВП General Error Handler, а не ВП Simple Error Handler
Правило 8.4	Избегайте опрашивать объекты графического интерфейса
Правило 8.12	Используйте перечень в качестве селектора случаев
Правило 8.13	Минимизируйте код, внешний по отношению к структуре Case
Правило 8.14	Добавьте состояния Initilaize (Инициализация), Idle (Бездействие), Shutdown (Выключение) и Blank (Пустое)

Блок-диаграмма унаследовала шаблон, сочетающий структуру кадров и цикл While. Структура кадров инициализирует несколько элементов лицевой панели. Цикл While содержит структуру Case и похож на шаблон классического конечного автомата, с тем исключением, что в селекторе использован целочисленный тип данных. Случай **Default** опрашивает логические элементы управления и проверяет, изменилось ли значение **Units** кластера **Cycle**. Эти особенности и нарушают правила, связанные с шаблоном конечного автомата, которые мы обсуждали в главе 8 «Шаблоны», в разделе 8.2 «Конечные автоматы».

Улучшенная блок-диаграмма показана на рис. 10.4б и использует правильный шаблон конечного автомата, основанный на структуре событий (Event), включая перечень в качестве селектора и структурой событий, расположенной в случае **Idle** структуры Case. Конечный автомат содержит стандартные состояния **Initialize** (Инициализация), **Idle** (Простой), **Shutdown** (Выключение), **Blank** (Пустое состояние). Структура событий заменяет собой постоянный опрос логических переменных меню и кластера **Torque**. Локальная переменная и сдвиговый регистр для распространения **Units** больше не нужны. Код инициализации элементов управления перенесен в состояние **Initialize** конечного автомата, структура кадров уничтожена. Метки состояний в перечне интуитивно понятны. Все проводники данных, выходящие из левого сдвигового регистра, промаркированы. Наконец, вместо Simple Error Handler VI используется General Error Handler VI. В результате блок-диаграмма стала компактной, читаемой, аккуратной и понятной.

Самостоятельные экспертные оценки программы дают вам определенную возможность пересмотреть, проверить, повысить качество вашего исходного кода. Таким образом, вы можете повторно ознакомиться с кодом, который разрабатывали в спешке или изменяли несколько раз; подумайте, понятен ли код, хорошо ли документирован и имеет ли логический смысл, обновите документацию по необ-

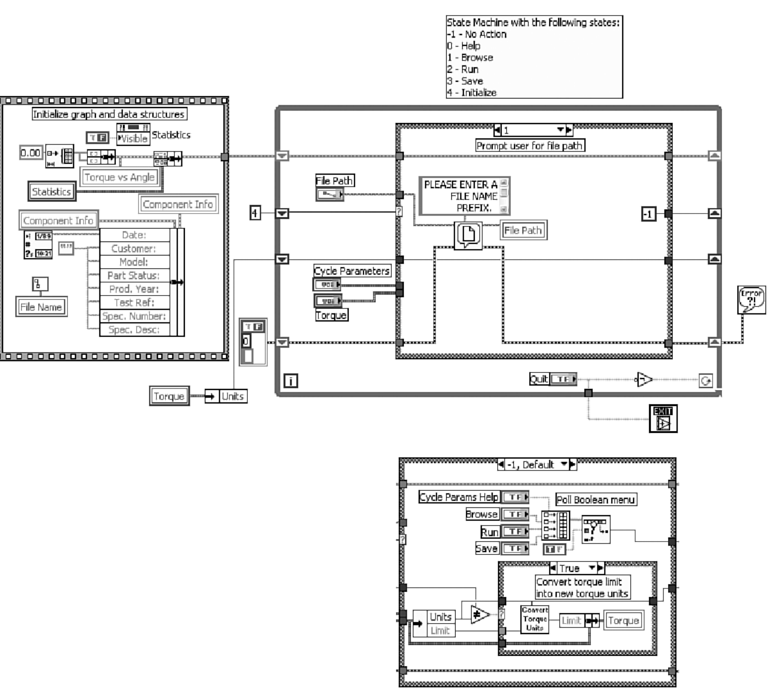


Рис. 10.4а. На блок диаграмме Torque Hysteresis – Improved with VI Analyzer VI нарушено множество правил

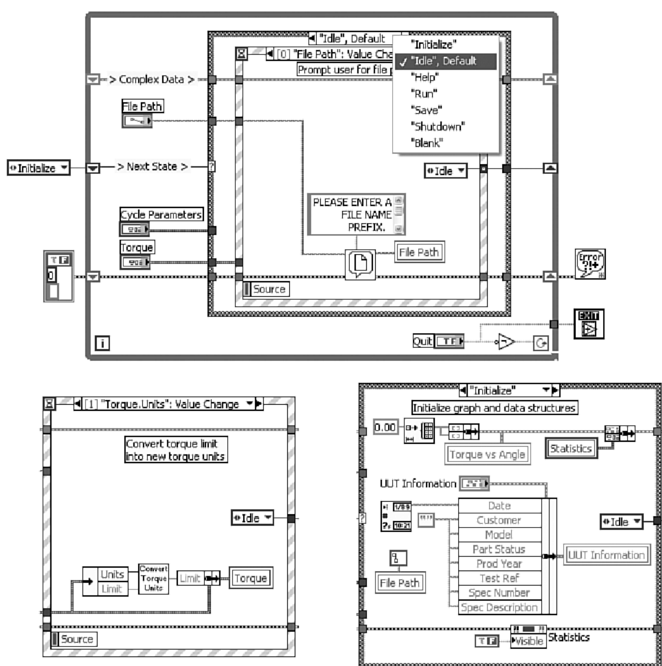


Рис. 10.4б. Улучшенная согласно правилам из Приложения Б блок-диаграмма использует правильный шаблон конечного автомата, основанный на структуре событий (Event)

ходимости, рассмотрите все аспекты, касающиеся стиля. Кроме того, это отличный способ подготовиться к экспертной проверке.

10.2. Экспертные проверки

Экспертные проверки – это экспертные оценки кода, выполненные одним или более экспертом. На практике множество затруднений разработчик может разрешить самостоятельно, когда попытается объяснить свой код коллегам. В тоже время некоторые методики разработки, используемые индивидуальными разработчиками, не известны широкой публике. В самом деле, в LabVIEW множество особенностей, и большинство разработчиков никогда не перестают учиться новым техникам. Как разработчик, так и участники обсуждения могут много почерпнуть во время критических обсуждений исходного кода с экспертами.

Теорема 10.1 *Взаимодействие с экспертами – это сильнейший стимул. И экспертные оценки являются эффективным методом усиления и развития стандартов стиля.*

Многие разработчики с радостью улучшают свой стиль программирования самостоятельно или с помощью учителя или экспертной группы. Обычно разработчик достигает некоторого плато, уровня, на котором он чувствует себя уверенно в рамках какого-то шаблона или набора инструментов, после этого он уже не так сильно хочет изучать альтернативные методы и стили. Общение с экспертами является мощным инструментом мотивации, и экспертные оценки – это отличный способ улучшить свои навыки, в том числе и стиль программирования. Более того, экспертные оценки являются ключевой техникой усиления стандарта стиля в организации.

Правило 10.8 Проводите хотя бы одну экспертную оценку каждого проекта

Правило 10.9 К обсуждению привлекайте следующих людей: менеджера проекта, ведущего разработчика, опытного разработчика и неопытного члена команды разработчиков

Правило 10.10 На обсуждение принесите список требований и контрольный список правил стиля

Оптимальное число и частота экспертных оценок могут сильно различаться. Вот несколько факторов, которые необходимо учитывать: уровень команды разработчиков (разработчика), размер и сложность приложения, основные цели экспертной оценки. Если вы являетесь членом организации с большим количеством разрабатываемых проектов, то проводите хотя бы одну экспертную оценку каждого проекта. К обсуждению привлекайте следующих людей: менеджера проекта, ведущего разработчика, опытного разработчика и неопытного члена команды разработчиков. Каждый из них имеет свои взгляды и приносит свою пользу. Менеджер проекта понимает требования к функциональности и может следить за тем, как соблюдаются эти требования. Это поможет определить любые функциональные ограничения конкретной схемы программного обеспечения. Ведущий разработчик – главный архитектор программного обеспечения, в его интересах и проводится оценка. Опытный разработчик знаком со стандартами стиля данной организации и обеспечит конструктивную критику. Неопытный член команды может привнести какие-то новые идеи и задать вопросы по существующим методам, заодно он будет учиться хорошему стилю. Дополнительно можно пригласить эксперта по какому-либо вопросу для оценки узкоспециализированных подсистем вашего приложения. Например, если ваш коллега – эксперт в разработке графического интерфейса, попросите его посмотреть ваш графический интерфейс. То же самое верно и для сетевых соединений, сбора данных, анализа изображений, контроля в реальном времени и т.д. Держите список требований и контрольный список правил стиля под рукой. Это важные материалы, и к ним вы будете обращаться во время встречи.

Начните с обзора критически важных требований, расставьте приоритеты. Затем обсудите общую архитектуру приложения, установите причины, почему она должна быть именно такой. Это может потребовать презентации блок-диаграммы или других моментов, после чего необходимо шаг за шагом разобрать исходный код верхнего уровня. Затем обсудить все подсистемы приложения, такие как высокоуровневые компоненты и низкоуровневые драйверы устройств. Необходимо затронуть все подсистемы и проанализировать их код. Необходимо также обсудить все структуры данных и все протоколы взаимодействия, используемые для связи всех высокоуровневых компонентов или подсистем. Также нельзя обойти вниманием стратегию обработки ошибок. Поищите обычные ошибки вроде незавершенной процедуры отслеживания ошибок, отсутствия описаний ВП, ненужные операции в циклах.

Правило 10.11 Назначьте нейтральную группу для записи желаемых изменений

Правило 10.12 Не изменяйте исходный код во время экспертной оценки

Лучше всего назначить нейтральную группу, не ведущего разработчика или менеджера проекта, делать записи во время экспертной оценки. В этом случае разработчики могут сконцентрироваться на представлении своего кода и критике, и встреча пойдет согласно программе. Также нейтральная группа сможет сделать беспристрастные выводы. Для удобства разработчик может захотеть добавить несколько комментариев в код с напоминаниями, как закладки. В каждый комментарий нужно включить термин для поиска, например Обзор + <дата>, для того, чтобы быстро найти комментарии после обзора, используя функцию поиска. Однако всегда следует избегать внесений изменений в исходный код во время экспертной оценки. Это замедляет встречу и меняет уровень перспективы.

Экспертная оценка ВП Torque Hysteresis была проведена последовательно для самостоятельно выполненной автоматической и ручной проверки. На лицевых панелях и блок-диаграммах на рис. 10.3б и 10.4б эксперты заметили следующие нарушения, которые остались не замеченными разработчиком:

Правило 3.37	Задайте порядок элементов (tabbing order) и пользуйтесь свойством Key focus (Фокус ввода)
Правило 6.2	Используйте эффективные с точки зрения использования памяти типы данных
Правило 6.27	Избегайте использования кластеров для интерактивных элементов управления с диалоговыми ВП
Правило 7.8	Используйте запись в лог-файл при внедрении приложения
Правило 7.9	Запрещайте диалоговые окна ошибок для удаленных или не требующих внимания приложений
Правило 8.4	Избегайте опрашивать объекты графического интерфейса

Дополнительно было рекомендовано повторно использовать некоторое программное обеспечение, созданное данной организацией, в том числе диалоговое окно в индустриальном стиле с двумя кнопками, похожее на обычный ВП LabVIEW, и процедуру записи ошибок. Были сделаны тщательные записи, и Torque Hysteresis VI была снова улучшена после экспертной оценки, как показано на рис. 10.5.

Рисунок 10.5а содержит измененную лицевую панель. Работа программы была улучшена добавлением логического элемента управления **New UUT** на вкладку **Menu**. Эта кнопка инициирует новую последовательность тестов, начиная с диалогового окна, в котором пользователю предоставляется информация о тестируе-

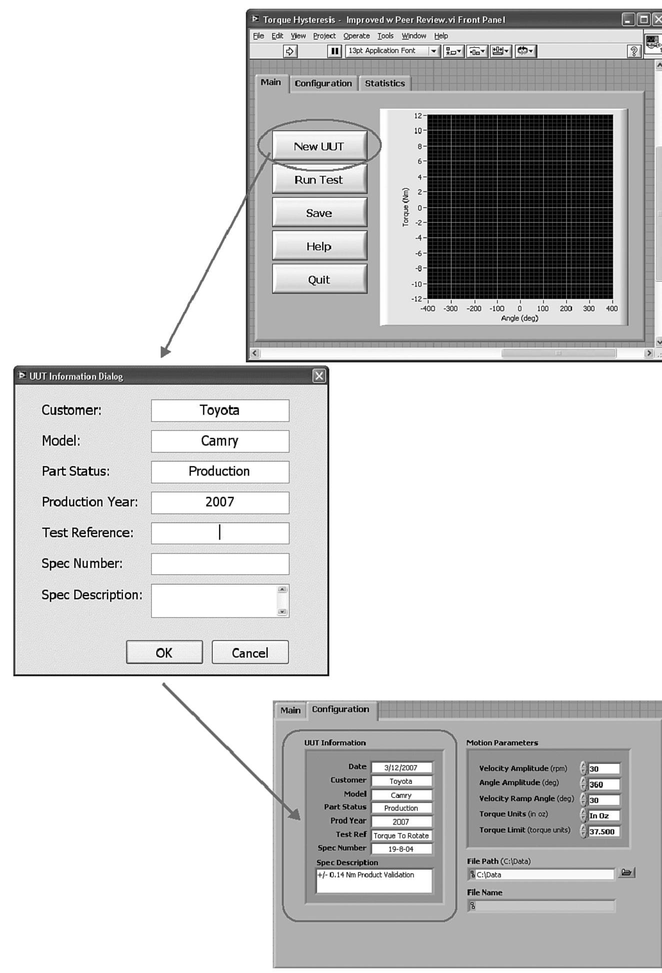


Рис. 10.5а. Окно лицевой панели Torque Hysteresis – Improved w Peer Review VI. Навигация улучшена добавлением новой кнопки ***New UUT*** *и диалогов, содержащих данные конфигурации*

мом объекте. Prompt UUT Information dialog VI использует большие индивидуальные элементы управления с навигацией через клавишу табуляции, что упрощает использование по сравнению с соответствующим кластером на вкладке **Configuration**. По аналогии Prompt Motion Parameters dialog VI представляет параметры цикла движения и обновляет соответствующий кластер на вкладке **Configuration**.

На рис. 10.5б содержится переделанная блок-диаграмма. В некоторых структурах данных используются сдвиговые регистры, расположенные у верхнего края цикла While. Кластеры UUT Information и Motion Parameters используют различные сдвиговые регистры, что уменьшает запутанность структуры данных. Кроме того, кластеры Torque vs Angel и Statistics передаются с использованием сдвиговых регистров. Это уменьшает количество локальных переменных и увеличивает гибкость приложения, так как данные доступны в каждом случае. Структуры данных и сдвиговые регистры улучшают поток данных и эффективность использования памяти.

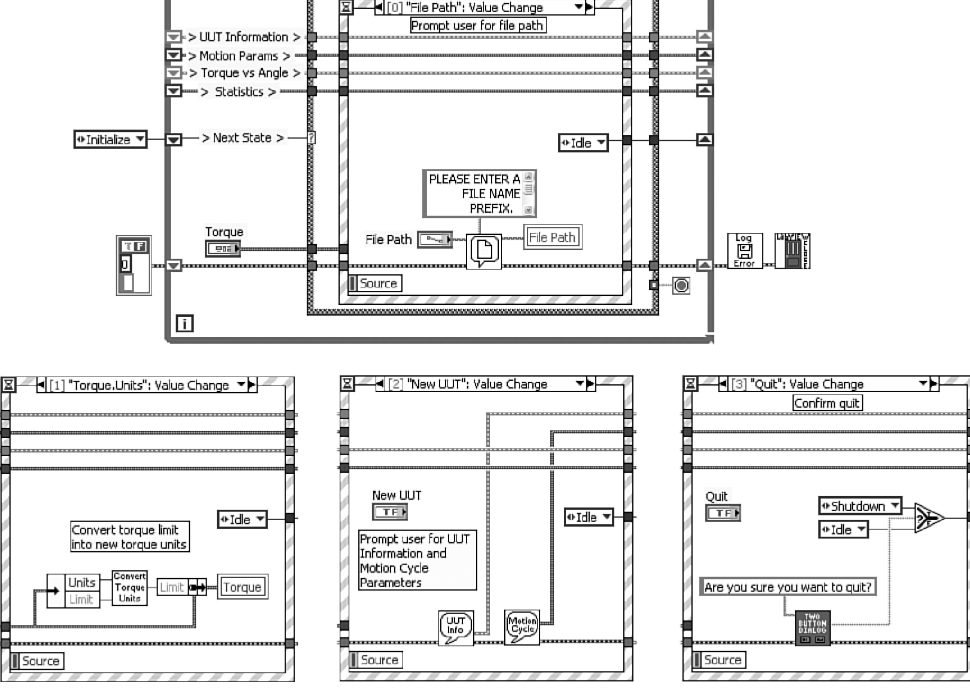


Рис. 10.5б. Блок-диаграмма Torque Hysteresis – Improved w Peer Review VI. Сдвиговые регистры снимают необходимость в локальных переменных и обеспечивают поток данных. Также используется несколько процедур из библиотеки организации для подтверждения выхода, записи ошибок и завершения сессии LabVIEW

Добавлены новые события в структуру Event, а именно **New UUT** и **Quit Value Change**. Новое событие **New UUT Value Change** вызывает диалоги Prompt UUT Information dialog VI и Prompt Motion Parameters dialog VI. Событие **Quite Value Change** заменяет постоянный опрос булиновской переменной **Quit**, которая до этого была соединена с терминалом условия выхода из цикла. События более эффективны, чем опрос переменных. Кроме того, диалоговое окно подтверждения выхода из приложения является стандартом в нашей организации. Дополнительно ВП General Error Handler заменен процедурой записи ошибок, тем самым диалоговое окно ошибки заменено записью в файл на заднем фоне. Эта техника обработки ошибок предпочтительнее, так как пользователя не будет смущать диалоговое окно ошибки. Наконец, вместо функции LabVIEW Quit используется Close LabVIEW Conditional VI. Эта процедура проверяет, запущен ли ВП как исполняемое приложение, прежде чем завершить сессию LabVIEW. Подтверждение выхода, процедура записи ошибок и завершения сессии LabVIEW являются стандартными ВП, доступными в библиотеке функций организации. Опытный разработчик нашел возможность повторного использования кода.

Короткое визуальное сравнение ВП Torque Hysteresis до экспертной оценки показано на рис. 10.2а, а улучшения в коде после всех трех экспертных оценок показаны на рис. 10.5, видна существенная разница в качестве программного обеспечения. Кроме того, экспертная оценка программы повышает уровень разработчика, глубину знаний и продуктивность работы. Более того, экспертные оценки укрепляют стандарты стиля, что улучшает простоту использования, эффективность, надежность, производительность и устойчивость к ошибкам приложений LabVIEW.

Ссылки

1. Настроенный файл конфигурации и контрольный список правил стиля в электронной форме можно скачать с **www.bloomy.com/lvstyle**.
2. «Advice for VI-Based Code Reviews», NIWeek2006 presentation by Doug Norman of National Instruments.

В словаре приведено значение терминов в том смысле, как они используются в данной книге. В большинстве случаев значение совпадает с документацией LabVIEW, а также с соответствующими промышленными стандартами программ и виртуальных приборов. Однако некоторые термины могут быть неоднозначными, другие изменяют свое значение со временем и могут иметь несколько значений. В этих случаях мне приходилось выбирать значение, наилучшим образом соответствующее материалу книги. Приведенные определения одинаковы, где бы они ни встретились в данной книге.

CHM. См. Компилированный файл справки

CHM компилированный файл справки. Стандартный файл справки Windows XP. У этих файлов есть подробный индекс и таблица гипертекстовых ссылок, обычно отдельно от основного файла

DOS (Disk Operating System). Основная операционная система персональных компьютеров, широко использовавшаяся в 90-х годах, предшественник MS Windows

LabVIEW Advanced Virtual Architects (LAVA). Сообщество, стремящееся к открытому и безвозмездному обмену идеями по вопросам программирования в LabVIEW среднего и высокого уровня сложности. Адрес – www.lavausergroup.org

NaN. Сокращение от Not a number (Не число), результат некорректных операций с числами с плавающей запятой, например извлечение корня из отрицательного числа. Эта константа широко используется для инициализации массивов, потому что ее легко найти при поиске в массиве, а на графике значение не отображается

NI. National Instruments, создатель LabVIEW. Адрес www.ni.com

OpenG. Организованное сообщество, разрабатывающее открытые программы и средства для LabVIEW. Адрес www.openg.org

PDF (Portable Document Format). Формат, предложенный компанией Adobe Systems для представления двух- и трехмерных документов в фиксированном кросс-платформенном виде. Вид PDF-файлов не зависит от какого-либо прибора

TBD (To be determed – подлежит определению). Технические требования, не определенные на данной стадии проектной документации

Variant (Произвольный). Тип данных, содержащий заголовок и тип данных, а также дополнительную информацию и атрибуты в общем формате. Можно рассматривать этот тип как формат, преобразующий данные к унифицированной форме

vi.lib. Папка со всеми стандартными ВП LabVIEW

А

Автомат событий (Event Machine). Шаблон на основе структуры событий внутри цикла While с заданными пользовательскими событиями и кадрами, обрабатывающими комбинацию как пользовательских событий, так и событий графического интерфейса. Этот шаблон рекомендуется, если в приложении требуется несколько структур событий

Асинхронные циклы (asinchronous loops). Два или более параллельных независимых цикла без какой-либо связи через тактирование или данные, из-за которых один цикл ожидает определенного момента другого цикла

В

Ввод/вывод (I/O). Операции ввода/вывода, которые обращаются к драйверам приборов, библиотекам, операционной системе или другим приложениям и ресурсам вне среды LabVIEW, включая удаленные сессии LabVIEW. Операции ввода/вывода осуществляют все функции палитр ввода/вывода с файлами (File I/O), измерений (Measurement I/O), приборов (Instrument I/O) и сетевых подключений (Data Communication)

Вложенная структура данных (nested data structure). Конструкция из массивов или кластеров с несколькими слоями

ВП верхнего уровня (top-level VI). ВП на верхней ступени иерархической лестницы приложения, его лицевая панель обычно представляет собой основное окно приложения

ВП графического интерфейса (GUI VI). Любой ВП с открытой пользователю лицевой панелью, включая ВП верхнего уровня и диалога

ВП графического интерфейса пользователя (Desktop GUI VI). ВП с лицевой панелью, предназначенный для использования в офисе, лаборатории или решения других вычислительных задач под управлением пользователя. Эти ВП обычно выглядят и ведут себя аналогично другим приложениям операционной системы

ВП диалога (dialog VIs). ВПП пользовательского интерфейса, которые открывают свои лицевые панели с запросом на ввод информации. Их функциональность и количество ВПП гораздо ниже, чем у ВП верхнего уровня. Единственная их цель – общение с пользователем

ВП компонент (component VI). ВПП высокого уровня или динамически загружаемый дополнительный ВП, который выполняет значительную обособленную часть приложения или описывает подсистему

ВП промышленного графического интерфейса (industrial GUI VI). ВП с лицевой панелью, предназначенный для управления промышленным оборудованием. Эти ВП предназначены для промышленных приложений и оборудования, вне зависимости от вычислительной платформы

ВПП (subVI). ВП, который вызывается с блок-диаграммы ВП более высокого уровня. В главах 3 и 8 также предполагается, что лицевая панель ВПП не открывается во время работы

Время разработки (development time). Время, необходимое для создания, документирования, отладки, изменения и поддержки приложения во время всего цикла его работы

Вызов по ссылке (Call by Reference). Функция сервера ВП, которая используется для запуска динамически загруженного ВП. В модели динамической загрузки по ссылке вызываются компоненты приложения высокого уровня

Г

Гибкая последовательность (flexible sequencer). Конструкт для исполнения многокадровой последовательности в виде структуры варианта в цикле. Порядок исполнения и выход из структуры определяются динамически во время исполнения. Гибкую последовательность можно реализовать в рамках классического конечного автомата

Глиф (glyph). Узнаваемое графическое изображение, например указатель или дорожный знак

Глобальные переменные версии LabVIEW 2 (LabVIEW 2 style globals). Одно из названий глобальных функционалов. Такое название возникло потому, что до введения переменных в версии LabVIEW 3.0 это был единственный способ передачи данных в нарушение потока данных

Глобальный функционал (Functional Global). Шаблон для передачи данными между ВПП, можно использовать вместо переменных. Он состоит из цикла While, структуры варианта, кольцевого списка и элементов ввода и вывода данных. У цикла есть, по крайней мере, один неинициализированный сдвиговый регистр, к терминалу условия выхода – константа, останавливающая его после первой итерации. Цикл нужен только для хранения данных в сдвиговом регистре. Как и глобальная переменная, ВПП вызывается каждый раз при необходимости обращения данных

Д

Диалоговые функции (Dialog functions). Встроенные в LabVIEW функции и экспресс-ВП открытия различных диалоговых окон (от одной до трех кнопок, запрос ввода и сообщения), расположенные на палитре Dialog & User Interface (Диалог и интерфейс пользователя). Их просто использовать, а внешний вид соответствует стандартным системным сообщениям

Динамический (dynamic). Универсальный тип данных, который используется в экспресс-ВП. Очень гибкий тип, предназначен для начинающих разработчиков. Позволяет хранить и преобразовывать различные типы данных без углубления в детали происходящих действий

Динамический каркас (Dynamic Framework). Организация работы, при которой приложение динамически подгружает и запускает ВП высокого уровня с помощью функций сервера ВП

Документация (documentation). Любое текстовое описание структур, компонентов или назначения системы, приложения или исходного кода. В документацию могут входить технические характеристики, проектная документация, документация исходного кода, руководства пользователя и справочная система

Дополнительный (plug-in). ВП, который динамически подгружается динамическим каркасом, к ним могут относиться ВП как высокого уровня, так и ВПП низкого уровня

Дочерняя панель (child panel). Лицевая панель ВП-компонента, загруженная в управляющий элемент subpanel

Ж

Жесткий цикл (tight loop). Цикл, который исполняется с максимально возможной частотой, без средств тактирования и задержки, что приводит к неэффективному использованию процессорного времени. Такой цикл может полностью занять процессор. При использовании любой задержки, например Wait (ms), Wait Until Next ms Multiple и экспресс-ВП Time Delay, исполнение задачи прерывается, и у процессора есть время на обработку других задач

З

Зависимость данных (data dependency). Использование принципа потока данных для установки очередности исполнения узлов, соединенных одним или несколькими проводниками. А именно: любой узел с входными данными не может начать работу, пока не получит их все, то есть существует зависимость данных между приемником и источником

Задача (task). Набор операций, описанных в ВПП или цикле While

Зона разработчиков NI (NI Developer Zone). Форум всех нуждающихся в информации по LabVIEW: статьи, примеры и техподдержка. В сети драйверов зоны есть тысячи драйверов для различных приборов. Адрес www.ni.com/idnet

И

Изгиб (kink). Небольшая петля на проводнике

Имена ввода/вывода (I/O names). Ссылки на открытые ресурсы (приборы или инструменты). Функционально они похожи на указатели на структуру данных этого ресурса. В именах ввода/вывода содержится краткое описание ресурса. Например, в управляющих элементах VISA, IVI и DAQmx Name содержится информация об оборудовании

Инструмент анализа ВП (VI Analyzer Toolkit). Дополнительная библиотека NI, предназначенная для автоматического анализа стиля и производительности программы. В него входит более 60 тестов, которые можно настроить и запустить в ручном или автоматическом режиме. Это очень полезное средство для самостоятельной проверки

Интервал (time stamp). Тип данных, предназначенный для хранения временных интервалов и моментов времени с большой точностью: 8 байт для хранения целой и дробной части числа секунд

Искусственная зависимость данных (artificial data dependency). Соединение двух объектов проводником данных, при котором получающий данные объект их не использует. Соединение служит только для определения порядка выполнения объектов в соответствии с принципом потока данных

К

Каркас приложения (application framework). Среда, содержащая различные конструкты: стандартные шаблоны, ВПП, данные, информацию и определяющую структуру разрабатываемого приложения

Классический конечный автомат (Classic State Machine). Шаблон из структуры варианта в цикле по условию. В качестве переменной выбора варианта используется тип данных нумерованного списка. Данные между итерациями цикла передаются с помощью сдвиговых регистров. Метки вариантов описательные, содержатся в нумерованном списке. Следующий кадр варианта для исполнения определяется программно

Кластер ошибок (error cluster). Стандартные входные и выходные кластеры error in и error out, которые используются для передачи информации об ошибках

Код ошибки (error code). Элемент кода в кластере ошибки представляет собой знаковое 32-битное целое число. Каждое число соответствует своей ошибке, которая может произойти в функции или ВП. В базе данных LabVIEW содержится описание внутренних ошибок, соответствующих выделенным кодам

Кольцевые списки (ring controls). Элементы, которые устанавливают соответствие между выбранной строкой текста и числовым значением. Очень удобны, чтобы предоставить выбор конечного числа вариантов, которые лучше описать, чем перенумеровать

Коммерческий ВПП (commercial subVI). ВПП, являющийся частью коммерческого продукта: библиотеки разработчика или драйвера прибора

Конечный автомат (State Machine). Наиболее популярный шаблон всех времен, состоит из структуры варианта в цикле While со сдвиговым регистром или конструктом передачи сообщений, определяющим терминал выбора структуры варианта

Конечный автомат с событиями (Event-Driven State Machine). Шаблон на основе конечного автомата, структуры событий и очереди, обрабатывающий события пользовательского интерфейса и обеспечивающий буферизованную обработку состояний. Этот шаблон подходит для ВП верхнего уровня и ВП графического интерфейса в приложениях средней сложности

Конструкт (construct). Элемент структуры, созданный разработчиком, к ним относятся структуры данных, ВПП и шаблоны

Конструкт данных (data construct). Объединение одного или нескольких встроенных типов данных в новый тип. К ним относятся как созданные пользователем конструкты: массивы, кластеры, универсальный тип (variant), переменные и очереди, а также встроенные: матрицы, ошибки, осциллограммы и динамический тип

Контекстная справка (Context Help). Окно справки в LabVIEW с описанием ВП и их компонентов, над которыми в данный момент находится курсор мыши. Чтобы открыть окно контекстной справки, выберите пункт меню Help ⇒ Show Context Help (Справка ⇒ Показать контекстную справку) или нажмите клавиши Ctrl+H

КПК (Personal Digital Assistant – PDA). Электронный прибор небольшого размера, сочетающий в себе возможности компьютера, сотового телефона, плеера и камеры. Часто КПК пользуются на совещаниях для внесения заметок вместо ноутбуков

М

Матрица (matrix). Определение типа: двумерный массив чисел двойной точности с плавающей запятой или комплексных чисел. Матрицы широко используются в операциях линейной алгебры

Модальный (modal). Поведение ВП-интерфейса, при котором его окно расположено поверх остальных и запрещает переключаться на другие окна приложения. По умолчанию все диалоги являются модальными: File ⇒ VI Properties ⇒ Window Appearance ⇒ dialog

Н

Надежный (robust). Свойство программы корректно обрабатывать ошибки в случае их возникновения и спокойно завершать работу

Настольное приложение (desktop application). Приложение, запускаемое на персональном компьютере и не предназначенное для работы в режиме реального времени или на встраиваемых системах

Начальный ВПП (Immediate SubVI). Шаблон ВПП для локализации ошибок. Нет никаких циклов, диалоговых окон и графических панелей. Этот ВПП исполняется последовательно, в порядке, определенном кластером ошибок

Непрерывный цикл (Continuous Loop). Шаблон программирования на основе одного цикла While со сдвиговыми регистрами, средствами тактирования и обработкой ошибок. Используется как в ВП верхнего уровня, так и в ВПП

О

Обособленность (cohesion). Мера слитности функций в модуле или ВПП. ВПП можно назвать обособленным, если его назначение можно полностью описать в 2 или 3 предложениях

Объяснить ошибку (Explain error). Встроенная функция LabVIEW, отображающая информацию об ошибке в отдельном диалоговом окне. Чтобы получить объяснение ошибки, выберите пункт контекстного меню любого кластера ошибок Explain Error или команду меню Help ⇒ Explain error

Определение типа (type definition). Также иногда называется тайпдефом (typedef). Это пользовательский элемент управления, информация о типе которого хранится в файле CTL. При изменении типа данных определения типа все его копии автоматически обновляются

Опрос (polling). Периодическая проверка значения ресурса, ожидание определенного значения или состояния. Опроса элементов лицевых панелей лучше избегать, есть более эффективные средства

Очередь (queue). Буфер FIFO, в частности, используется для передачи данных между параллельными циклами без переменных

Ошибка (error). Невозможность функции или ВП выполнить поставленную задачу. Данные об ошибке обычно передаются с помощью кластеров error in и error out

П

Память блок-диаграммы (block diagram memory). Одна из 4 составляющих памяти ВП, хранит графические объекты и изображения блок-диаграммы. Используемую память блок-диаграммы можно посмотреть в категории Memory Usage (Использование памяти) окна свойств ВП: File ⇒ VI Properties (Файл ⇒ Свойства ВП).

Память данных (data memory). Одна из 4 составляющих памяти ВП со всеми используемыми данными, включая константы, значения по умолчанию элементов лицевой панели, а также копии данных при использовании переменных и индикаторов. Количество памяти, которую занимают данные, можно посмотреть в категории Memory Usage (Использование памяти) окна свойств ВП: File ⇒ VI Properties (Файл ⇒ Свойства ВП). Также в окне Profile Memory and Performance (Запуск через меню Tools ⇒ Profile ⇒ Performance and memory (Инструменты ⇒ Измерить ⇒ Производительность и память)) представлена полная информация о памяти, занимаемой ВП и его ВПП

Память кода, исходный код (code memory или code). Одна из 4 составляющих памяти ВП с компилированным исходным кодом ВП. Количество памяти кода можно посмотреть в категории Использование памяти окна свойств ВП: File ⇒ VI Properties (Файл ⇒ Свойства ВП)

Память лицевой панели (front panel memory). Одна из 4 составляющих памяти ВП со всеми графическими элементами и рисунками, составляющими лицевую панель. Количество памяти лицевой панели можно посмотреть в категории Использование памяти окна свойств ВП: File ⇒ VI Properties (Файл ⇒ Свойства ВП)

Персональный компьютер (PC). Компьютер под управлением ОС нереального времени, например MS Windows

Поддерживаемый (maintainable). Простота понимания программы другими программистами с целью изменения и добавления новых функций

Полиморфизм (polymorphism). По отношению к ВП означает, что один или несколько его терминалов могут принимать данные различных типов. Эти терминалы приспосабливаются к поданному типу данных, а не приводят тип и не разрывают проводник ошибкой

Пользовательское событие (user event). Определенное пользователем событие, которое обрабатывается в структуре последовательности. Пользовательские события создаются, регистрируются и генерируются программно с использованием функций палитры Events: Programming ⇒ Dialog & User Interface ⇒ Events (Программирование ⇒ Диалог и Интерфейс пользователя ⇒ События)

Предупреждение (warning). В случае предупреждения код в кластере ошибки не равен нулю, но статус ошибки FALSE. Предупреждения похожи на ошибки, но считаются не такими критичными

Префикс прибора (instrument prefix). Сокращение производителя и модели прибора в библиотеке драйвера прибора. Первые 2–3 символа обозначают производителя, еще 4 или 5 – модель и семейство. Этот акроним из 6–8 символов используется в именах файлов и на иконках ВПП

Программирование-отладка (Code and Fix). Модель разработки приложения, в соответствии с которой исходный код приложения постоянно пересматривается, изменяется и отлаживается без всякого предварительного планирования. Эта модель отличается отсутствием четких требований, непонятной архитектурой, запутанным исходным кодом, что в результате приводит к низкому качеству и малоэффективному использованию времени и усилий

Программируемая логическая интегральная схема (ПЛИС) – Field Programmable Gate Array (FPGA). Прибор с массивом программируемых логических вентилей. При программировании ПЛИС внутренние соединения устанавливаются таким образом, чтобы получившаяся схема реализовывала программу. В отличие от процессоров, аппаратная часть ПЛИС всегда соответствует решаемой задаче и не требует операционной системы

Программист (programmer). Разработчик приложения, пользующийся стандартным текстовым языком

Производительность (performance). Очень широкий термин, который включает эффективность приложения и его скорость (эффективность ВПП)

Производительность ВПП (subVI performance). Скорость исполнения ВПП

Производительность приложения (application performance). Эффективность, с которой приложение или ВП выполняет поставленную задачу

Простота (simplicity). Величина, обратно пропорциональная числу узлов и терминалов приложения, – антоним сложности. Чем меньше элементов на лицевой панели и узлов на блок-диаграмме, тем больше простота приложения

Простота использования (ease of use). Простота действий в приложении, которые необходимы для описания, запуска и получения результатов задачи. Относится в основном к графическому интерфейсу пользователя

Простые структуры данных (simple data structures). Структуры данных, которые хранят данные в непрерывных блоках памяти. К ним относятся все скалярные типы данных (логические, числа и строки), массивы логических и числовых значений, а также кластеры, содержащие вышеперечисленные типы простых данных

Простые элементы (simple controls). Понятные управляющие элементы, не требующие объяснений, содержащие простые типы данных. У простых элементов есть свойства, которые позволяют настроить проверку ввода данных, как программного, так и пользователем

Профилировщик (Profile window). Инструмент измерения занимаемой памяти и времени исполнения ВП и ВПП, загруженных в память. Открывается с помощью меню Tools ⇒ Profile ⇒ Performance and Memory (Инструменты ⇒ Измерить ⇒ Производительность и память)

Пустое место (white space). Свободное место в окне графического интерфейса

Р

Разработчик (developer). Создатель программ, приложений как на низком уровне, так и на высоком

Реальное время (real time). Режим работы приложения или операционной системы, отличающийся детерминированным временем исполнения операций

Рецензия кода (code review). Систематический анализ кода не его разработчиком на предмет улучшения его читабельности, простоты поддержки и надежности. Можно выделить три преимущества рецензирования кода: обнаружение и выделение неполадок, обмен знаниями и идеями между разработчиками, соответственно повышение их профессионального уровня и третье – поддержка единого стиля программ в пределах компании

С

Сбор данных (data acquisition – DAQ). Регистрация сигналов от различных источников и оцифровка сигнала для хранения, анализа и отображения. Этот термин обычно используется при работе с платами оцифровки для компьютера

Сдвиговые регистры (shift registers). Терминалы на границах циклов, передающие данные между итерациями. Функционально они представляют собой проводники, соединяющие конец одной итерации с началом следующей

Сеть инструментов для LabVIEW (LabVIEW Tools Network). Сайт с поддержкой NI в зоне разработчиков LabVIEW (LabVIEW Zone) с ресурсами для разработчиков, включая дополнительные библиотеки, книги, обучающие пособия и другие материалы

Служебный ВПП (utility VI). ВПП, который выполняет набор операций низкого уровня, расширяющий или дополняющий функции LabVIEW

Собственный элемент управления (custom control). Элемент, измененный разработчиком с помощью редактора элементов управления и сохраненный в файле CTL. Каждую копию собственного элемента управления можно редактировать в программе, не затрагивая внешний вид, свойства и тип данных в исходном файле

Составные шаблоны (compound design pattern). Структуры из двух или более простых шаблонов на основе одного цикла с системой передачи данных между ними

Список (enumeration, enum). Специальный тип данных, предоставляющий выбор одной или нескольких строк, с каждой из которых связано целое число. От кольцевого текстового списка отличается тем, что численные значения и метки строк входят в тип данных. Этот тип данных очень удобен, если нужно дать пользователю выбрать из нескольких вариантов, которые лучше описать, чем перечислить по номерам. От кольцевых списков отличаются также и тем, что данным соответствуют беззнаковые целые числа, причем последовательно и в указанном порядке

Ссылка (refnum). Указатель на открытый ресурс: файл, прибор или инструмент, сетевое соединение, изображение, приложение, ВП или элемент управления. Ссылка (Refnum) – аналог указателей на структуру данных ресурса

Стандартный ВПП (standard subVI). Пользовательский ВПП, не входящий в коммерческий продукт или библиотеку

Строгое определение типа (strict type definition). Пользовательский элемент управления, сохраняющий точный внешний вид, свойства и тип данных каждой копии

Структура варианта в цикле (Looped Case Structure). Конструкт из структуры варианта в цикле. Работает аналогично гибкой последовательности, когда участки кода для последовательного исполнения расположены в кадрах структуры варианта, а не последовательности

Структура варианта с обработкой ошибок (Error Case Structure). Структура варианта, терминал выбора которой соединен с входным кластером ошибок, в результате у структуры два кадра: No error (Нет ошибки) и Error (Ошибка). Эта структура широко применяется в ВПП

Структура данных (data structure). Любая структура или конструкт, которая используется в LabVIEW для хранения данных. Структуры данных определяются выбором управляющих элементов, массивов и кластеров на лицевой панели ВП, а также при операциях с данными на блок-диаграмме

Т

Тестируемый прибор (unit under test – UUT). Цель многих приложений LabVIEW – проверка и тестирование прибора

Тип данных (data type). Элемент данных, которому соответствуют определенный терминал и вид проводника. Типы данных отличаются занимаемым размером и возможными операциями.

Тип – осциллограмма (waveform data type – UUT). Специальный тип кластера из трех элементов и атрибутов: интервала с моментом начала данных (t0), интервала времени между точками и одномерного массива чисел – отсчетов аналоговой или цифровой осциллограммы. Атрибутами могут быть любые числа, метки или пары значений.

У

Удобочитаемость (readability). Простота восприятия исходного кода, термин применим как к лицевой панели, так и к блок-диаграмме.

Узел (node). Любой объект блок-диаграммы с терминалами входных и/или выходных данных, работающий при запуске ВП. К узлам относятся функции, ВПП и структуры.

Ф

Файл регистрации ошибок (error log file). Метод информирования о неполадках, который сохраняет информацию об ошибках в текстовый или бинарный файл

Файл хранения данных (datalog file). Бинарный формат файла, в котором записывается последовательность одинаковых элементов вместе с описанием типа данных. Для работы с этими файлами используются функции палитры Datalog. Тип данных определяется кластером, поданным на вход record type входного терминала функции Open/Create/Replace Datalog. Эти файлы нельзя прочитать или изменить в средах, отличных от LabVIEW

Ц

Целевое устройство (target). Любое вычислительное устройство, на котором может работать LabVIEW. К целевым устройствам относятся персональные компьютеры, встроенные контроллеры, КПК и ПЛИС

Цепочка ошибок (error chain). Метод локализации ошибок объединением узлов с кластерами ошибок в цепочку. Цепочкой ошибок также называют эту последовательность узлов

Цикл обработки событий (Event-Handling Loop). Шаблон, реализующий основы обработки событий в LabVIEW. Состоит из структуры событий внутри цикла While. В каждом кадре структуры расположен участок код, который исполняется при возникновении соответствующего события. Этот шаблон позволяет более эффективно и гибко обрабатывать события пользовательского интерфейса, чем опрос элементов в непрерывном цикле

Циклический ВПП (loop-subVI). Конструкт на основе цикла, который выполняет одну из основных задач приложения, оформленную в виде ВПП

Ш

Шаблоны (design patterns). Стандартные архитектуры ВП для решения типичных проблем в различных типах приложений. Они состоят из объединения структур, функций, управляющих элементов и средств обработки ошибок. Шаблоны используются для организации цикла работы, регистрации ошибок, управления состоянием и для хранения и инкапсуляции данных

Э

Экспертная оценка (peer review). Рецензия кода, выполненная одним или несколькими специалистами. При объяснении программы специалистам часто всплывают разные неточности и ошибки

Эффективность (efficiency). Качество использования приложением ресурсов: процессорного времени, памяти, устройств ввода/вывода. Эффективное приложение работает с нужной скоростью, причем без ненужных операций, особенно в циклах. Также эффективно используется память, ограничивая размер всех четырех компонентов программы: лицевой панели, блок-диаграммы, данных и кода.

В этом дополнении собраны все правила стиля, представленные в этой книге. Используйте их в качестве контрольного списка во время оценки кода, об этом более подробно было обсуждено в главе 10 «Обзор кода», раздел 10.1.2. Электронная копия этого списка доступна по адресу www.bloomy.com/lvstyle.

Глава 2

Правило 2.1 Ведите журнал проекта в LabVIEW.
Правило 2.2 Напишите техническое задание.
Правило 2.3 Тестирование блоков программы с помощью отдельных ВП положительно влияет на стиль.
Правило 2.4 Запишите опции LabVIEW и сохраните файл настроек (LabVIEW.ini).
Правило 2.5 Создавайте ВПП для повторного использования.
Правило 2.6 Библиотеки ваших приборов должны быть на палитре функций.
Правило 2.7 Скопируйте свои шаблоны в папку LabVIEW\templates.
Правило 2.8 Структурируйте место расположения файлов проекта.
Правило 2.9 Структура папок проекта должна соответствовать иерархии приложения.
Правило 2.10 Иерархия папок должна быть создана перед написанием программы.
Правило 2.11 Если это возможно, объединяйте исходные файлы проекта LabVIEW в библиотеки проекта.
Правило 2.12 Имена исходных файлов должны быть описательными и уникальными.
Правило 2.13 Не используйте аббревиатур.
Правило 2.14 Не пользуйтесь именами, которые LabVIEW дает по умолчанию.
Правило 2.15 Укажите ВП верхнего уровня.
Правило 2.16 Следуйте правилам внесения изменений, принятых в вашей компании.
Правило 2.17 Не стоит изменять расположение файлов.

Глава 3

Правило 3.1 Разделяйте логически не связанные элементы с помощью закладок, кластеров, элементов оформления и промежутков.
Правило 3.2 Придерживайтесь симметричного расположения элементов.
Правило 3.3 Размер объектов одного назначения должен быть одинаковым.
Правило 3.4 Разверните окно ВП верхнего уровня промышленного приложения на весь экран.
Правило 3.5 Диалоговые окна должны быть небольшими.
Правило 3.6 Располагайте диалоговые окна в центре экрана.
Правило 3.7 Пользуйтесь диалоговыми окнами LabVIEW для настольных приложений, избегайте их в промышленных задачах.
Правило 3.8 Системные элементы управления используются для настольных приложений, трехмерные элементы – для промышленных.
Правило 3.9 Более важные элементы управления в промышленных приложениях должны быть больше по размеру и располагаться ближе к центру окна.
Правило 3.10 Ограничивайте количество информации на лицевой панели.
Правило 3.11 Избегайте перекрытия объектов.
Правило 3.12 Скрывайте панель инструментов.
Правило 3.13 Во всех профессиональных приложениях есть логотип компании.
Правило 3.14 Для лицевой панели ВПП, ее объектов и большей части текста пользуйтесь стандартным стилем.
Правило 3.15 Расположение элементов должно соответствовать разъемам соединительной панели.
Правило 3.16 Размер панели должен быть удобным для расположения всех элементов.
Правило 3.17 Количество текста на лицевой панели должно быть минимальным.
Правило 3.18 Удаляйте временные замечания сразу после выполнения их инструкций.
Правило 3.19 Не злоупотребляйте форматированием текста.
Правило 3.20 Выберите один шрифт и подчеркивайте разный стиль его размером, толщиной и цветом.
Правило 3.21 Пользуйтесь интуитивно понятными метками и внедренным текстом.
Правило 3.22 Используйте стандартный шрифт (Application размера 13, черный) для большинства элементов ВПП.
Правило 3.23 В конце меток элементов указывайте значение по умолчанию.
Правило 3.24 В метках элементов используйте метки с полужирным шрифтом и пояснения стандартным шрифтом.
Правило 3.25 Увеличьте контраст между шрифтом и фоном.
Правило 3.26 Увеличьте размер шрифта кнопок и важных данных.
Правило 3.27 Увеличьте промежутки между метками и объектами многоплатформенных приложений.

Правило 3.28 Не злоупотребляйте цветами.
Правило 3.29 Разработайте цветовую схему приложения.
Правило 3.30 Следуйте общепринятым соотношениям при использовании желтого, зеленого и красного цветов.
Правило 3.31 Оставьте лицевую панель и объекты ВПП серыми.
Правило 3.32 Оставьте цветовую схему простой, не тратьте слишком много времени.
Правило 3.33 Ограничьте количество активных элементов управления.
Правило 3.34 Ограничьте диапазон изменения всех управляющих элементов.
Правило 3.35 Пользуйтесь свойством data range (Диапазон данных) для численных элементов управления.
Правило 3.36 Пользуйтесь списками выбора, а не строками там, где это возможно.
Правило 3.37 Задайте порядок элементов (tabbing order) и пользуйтесь свойством фокус ввода (Key focus).
Правило 3.38 Настройте меню ВП верхнего уровня.
Правило 3.39 На ВП должна быть справочная кнопка или пункт меню.
Правило 3.40 Будьте последовательны.

Глава 4

Правило 4.1 Установите разрешение 1280×1024.
Правило 4.2 Оставьте фон белым.
Правило 4.3 Располагайте объекты плотно.
Правило 4.4 Ограничьте размер блок-диаграммы одним экранам, по крайней мере, по ширине или высоте.
Правило 4.5 Создайте многоуровневую архитектуру ВПП.
Правило 4.6 Создавайте модульные программы с помощью ВПП.
Правило 4.7 Используйте ВПП в ВПП.
Правило 4.8 Не добавляйте ВПП просто для экономии места.
Правило 4.9 Избегайте тривиальных ВПП.
Правило 4.10 Иконка и понятное описание должны быть у каждого ВПП.
Правило 4.11 Избегайте петель и изгибов проводников.
Правило 4.12 Параллельные проводники должны проходить на одинаковом расстоянии.
Правило 4.13 Туннели должны быть расположены на вертикальных границах структур.
Правило 4.14 Не проводите через структуры лишний раз.
Правило 4.15 Никогда не закрывайте узлы и проводники.
Правило 4.16 Ограничивайте длину проводника: его начало и конец должны быть видны на экране.
Правило 4.17 Никогда не используйте переменные просто ради удобства соединений.
Правило 4.18 Добавляйте метки для длинных проводников.
Правило 4.19 Размещайте неиспользуемые терминалы элементов отдельно.

Правило 4.20 Объединяйте связанные данные в кластеры.
Правило 4.21 Сохраняйте кластеры как определения типа.
Правило 4.22 Всегда направляйте поток данных слева направо.
Правило 4.23 Пользуйтесь кластером ошибок.
Правило 4.24 Избегайте приведения типов у кластеров и массивов.
Правило 4.25 Создавайте константы и элементы управления из контекстного меню терминалов.
Правило 4.26 Отключите точки на узлах проводников.
Правило 4.27 Избегайте структур последовательности без необходимости.
Правило 4.28 Избегайте глубоких (больше 3) вложений.
Правило 4.29 Инициализируйте элементы управления с помощью локальных переменных.
Правило 4.30 Передавайте данные между простыми параллельными ВП с помощью глобальных переменных.
Правило 4.31 Структуры последовательности нужны, чтобы задать порядок действий, не связанных потоком данных.
Правило 4.32 При необходимости пользуйтесь только плоской структурой последовательности.
Правило 4.33 Избегайте опроса переменных в циклах.
Правило 4.34 Избегайте переменных, если задачу можно решить с помощью проводников.
Правило 4.35 Сдвиговые регистры предпочтительнее локальных и глобальных переменных.
Правило 4.36 Располагайте большинство сдвиговых регистров в верхней части цикла.
Правило 4.37 Добавляйте метки-пояснения к левым сдвиговым регистрам
Правило 4.38 Структура выбора с циклом предпочтительнее последовательности.

Глава 5

Правило 5.1 Получите удовольствие от создания иконки.
Правило 5.2 У каждого ВП должна быть своя, причем поясняющая, иконка.
Правило 5.3 Никогда не пользуйтесь стандартными иконками LabVIEW.
Правило 5.4 Сохраняйте ВП с отображением ВПП в виде иконок, а не контактов.
Правило 5.5 Пользуйтесь черной границей.
Правило 5.6 Лучшим стилем обладают цветные иконки с глифами и текстом.
Правило 5.7 Глифы должны быть описательными.
Правило 5.8 Набирайте текст размером 8 или 10.
Правило 5.9 Стиль взаимосвязанных ВП должен быть единообразным.
Правило 5.10 Время на создание иконки должно соответствовать планам на ВПП.
Правило 5.11 Быстрее всего сделать иконку с черной границей, цветным фоном и текстом.

Правило 5.12 Контраст между текстом и фоном должен быть высоким.
Правило 5.13 Выберите цветовую схему для целого набора ВП.
Правило 5.14 Создайте шаблон иконки для взаимосвязанных ВП.
Правило 5.15 Пользуйтесь одним глифом, цветовой схемой и шрифтом для взаимосвязанных ВП.
Правило 5.16 Копируйте графику.
Правило 5.17 Избегайте локализованных текста и графики.
Правило 5.18 Выберите расположение контактов и их назначение, чтобы обеспечить правильный поток данных и не усложнять соединения.
Правило 5.19 Оставляйте на шаблоне панели пустые разъемы.
Правило 5.20 Шаблон взаимосвязанных ВПП должен быть единообразным.
Правило 5.21 Для большинства ВП подойдет шаблон 4×2×2×4.
Правило 5.22 Водные данные должны быть слева, результаты справа.
Правило 5.23 Не допускайте пересечения проводников в окне контекстной помощи.
Правило 5.24 Расположение контактов и элементов лицевой панели должно соответствовать друг другу.
Правило 5.25 Кластеры ошибок должны быть снизу с краю.
Правило 5.26 Ссылки и имена ресурсов должны быть сверху с краю.
Правило 5.27 Контакты на вертикальных границах соединительной панели предназначены для основных данных.
Правило 5.28 Контакты на горизонтальных границах соединительной панели предназначены для дополнительных параметров.
Правило 5.29 Установите свойство «обязательный разъем» для критически важных данных.
Правило 5.30 Укажите свойство «дополнительный разъем» для необязательных параметров.

Глава 6

Правило 6.1 Выбирайте элементы управления, упрощающие работу с панелью.
Правило 6.2 Используйте эффективные с точки зрения использования памяти типы данных.
Правило 6.3 Используйте те элементы управления и типы данных, которые облегчат создание согласованных структур во всем приложении.
Правило 6.4 Задайте соответствующее значение по умолчанию для каждого элемента управления.
Правило 6.5 Вводите описания элементов управления.
Правило 6.6 Сохраняйте собственные элементы управления как строгие определения типов (strict type def, или тайпдеф).
Правило 6.7 Создавайте массивы и кластеры, которые объединят связанные элементы.
Правило 6.8 Используйте булиновские переменные, если два состояния логически противоположны.

Правило 6.9 Присваивайте имена, которые идентифицируют поведение значений ИСТИНА и ЛОЖЬ.
Правило 6.10 Используйте кнопки управления для действий, ползунковый переключатель – для установок параметров.
Правило 6.11 Отмечайте состояния ИСТИНА/ЛОЖЬ для ползунковых переключателей и тумблеров.
Правило 6.12 Избегайте использования кнопок и переключателей как индикаторов и светодиодов – как элементов управления.
Правило 6.13 Используйте представление I32 для целых чисел и DBL – для чисел с плавающей точкой.
Правило 6.14 Используйте автоматическое форматирование, если не требуется специальный формат.
Правило 6.15 Показывайте основание системы исчисления для данных в восьмеричной, шестнадцатеричной или двоичной системе.
Правило 6.16 Используйте перечни как можно больше во всех приложениях.
Правило 6.17 Сохраняйте перечни как тайпдеф.
Правило 6.18 Избегайте строковых элементов управления на лицевой панели.
Правило 6.19 Используйте перечень, кольцевые структуры и указатели пути вместо строковых элементов управления везде, где возможно.
Правило 6.20 Держите кнопку Browse видимой для указателей пути на лицевой панели.
Правило 6.21 Используйте массивы для данных с множеством значений; используйте кластеры для группировки различных типов данных.
Правило 6.22 Используйте массивы для хранения большого объема данных или наборов данных динамически изменяющейся длины.
Правило 6.23 Вводите описания для ячеек массивов и кластеров и элементов управления.
Правило 6.24 Используйте инструменты выравнивания, чтобы кластеры выглядели компактно и аккуратно.
Правило 6.25 Сохраняйте все кластеры как тайпдефы.
Правило 6.26 Всегда используйте Bundle By Name и Unbundle By Name.
Правило 6.27 Избегайте использования кластеров для интерактивных элементов управления с диалоговыми ВП.
Правило 6.28 Упорядочивайте сложные данные, используя вложенные структуры данных.
Правило 6.29 Избегайте манипулирования вложенными структурами во время критических операций.
Правило 6.30 Ограничивайте размер массивов, указывая максимальную длину.

Глава 7

Правило 7.1 Все ВП должны захватывать ошибки и сообщать об ошибках, возвращаемых терминалами ошибок.
Правило 7.2 Отслеживайте ошибки, передавая кластер ошибок между терминалами ошибок.

Правило 7.3 Отслеживайте ошибки на всех итерациях цикла.
Правило 7.4 Отключите индексацию ошибок в продолжительных циклах.
Правило 7.5 Отслеживайте все ошибки от всех узлов, имеющих терминалы ошибок.
Правило 7.7 Используйте ВП General Error Handler, а не ВП Simple Error Handler.
Правило 7.8 Используйте запись в лог-файл при внедрении приложения.
Правило 7.9 Запрещайте диалоговые окна ошибок для удаленных или не требующих внимания приложений.
Правило 7.10 Избегайте ВПП с встроенными отчетами об ошибках.
Правило 7.11 Храните коды ошибок, определяемые пользователем в XML-файле.
Правило 7.12 Используйте отрицательные коды для ошибок устройств ввода/вывода и положительные – для предупреждений.
Правило 7.13 Пропускайте большинство блок-диаграмм ВПП при возникновении ошибок, используя Error Case Structure.
Правило 7.14 Используйте значения по умолчанию для несоединенных туннелей в случае Error.
Правило 7.15 Используйте ВПП с шаблоном обработки ошибок.
Правило 7.16 Отслеживание ошибок требуется для узлов, осуществляющих операции ввода/вывода, рекомендуется для узлов, обладающих терминалами ошибок, и является опциональным для блок-диаграмм, не содержащих узлов с терминалами ошибок.
Правило 7.17 Туннели для кластера ошибок следует размещать внизу структуры.
Правило 7.18 Оставляйте автоматическую обработку ошибок выключенной.

Глава 8

Правило 8.1 Используйте несколько критериев выхода из цикла.
Правило 8.2 Используйте Timed Loop (цикл с ограничением по времени исполнения или числу итераций) для задания точного или сложного времени выполнения задачи, в других случаях используйте цикл While.
Правило 8.3 Вставляйте задержки в непрекращающиеся циклы While.
Правило 8.4 Избегайте опрашивать объекты графического интерфейса.
Правило 8.5 Используйте событие Value Change (Изменение значения) для большинства элементов управления графического интерфейса.
Правило 8.6 Терминалы элементов управления размещайте внутри их случая Value Change.
Правило 8.7 Измените размер узла Even Data Node так, что бы спрятать не используемые терминалы.
Правило 8.8 Избегайте продолжительных событий «по истечении времени» (timeout).
Правило 8.9 Используйте шаблон конечного автомата в большинстве ВП средней или высокой сложности.

Правило 8.10 Основные состояния приложения берите из спецификации или проектной документации.
Правило 8.11 Разделяйте основные состояния на второстепенные.
Правило 8.12 Используйте перечень в качестве селектора случаев.
Правило 8.13 Минимизируйте внешний по отношению к структуре Case код.
Правило 8.14 Добавьте состояния Initialize (Инициализация), Idle (Бездействие), Shutdown (Выключение) и Blank (Пустое).
Правило 8.15 Избегайте терминала timeout функций Enqueue Element и Dequeue Element.
Правило 8.16 Используйте очереди, общие переменные или RT FIFO для обмена данными между параллельными циклами.
Правило 8.17 Расставьте приоритеты циклов, используя задержки или свойства потоков.
Правило 8.18 Ширину параллельных циклов установите одинаковой и выровняйте по вертикали.
Правило 8.19 Минимизируйте свободное место между циклами.
Правило 8.20 Маркируйте каждый цикл в левом верхнем углу.
Правило 8.21 Используйте Call By Reference Node вместо метода Run.
Правило 8.22 Используйте стандартные назначения соединительных терминалов.
Правило 8.23 Все компоненты храните в выделенной директории.
Правило 8.24 Данные между компонентами передавайте через сдвиговые регистры.
Правило 8.25 Один входной и выходной терминал назначьте как универсальный.
Правило 8.26 Отображайте лицевую панель плагина в субпанели.
Правило 8.27 Создавайте параллельный цикл для каждой связанной параллельной задачи.
Правило 8.28 Добавьте циклы обработки событий, основного конечного автомата, аппаратного ввода/вывода и обработки ошибок.
Правило 8.29 Передавайте ссылки на элементы управления в ВПП, используя кластер с определением типов.
Правило 8.30 Избегайте использования утилиты subVI from Selection с непрерывными циклами.
Правило 8.31 Цикл обработки сообщений оставляйте на верхнем уровне.
Правило 8.32 Высокоскоростное обновление экрана следует также оставить на верхнем уровне.

Глава 9

Правило 9.1 Оставляйте метки терминалов видимыми на блок-диаграмме.
Правило 9.2 Используйте комментарии в свободных метках.
Правило 9.3 Помечайте каждую вложенную диаграмму каждой структуры с вложенными диаграммами.
Правило 9.4 Помечайте алгоритмы, константы и узлы Call Library Function Nodes.

Правило 9.5 Используйте простой черный шрифт размера 13 по умолчанию для всего текста на блок-диаграмме.
Правило 9.6 Оставляйте заметки для команды разработчиков.
Правило 9.7 Используйте пронумерованные типы данных со структурами Case.
Правило 9.8 Создавайте иконки и описания во время разработки исходного кода.
Правило 9.9 Предоставляйте онлайн-документацию вместе с разрабатываемым приложением.

Глава 10

Правило 10.1 Усильте стандарты стиля программирования в вашей организации, используя экспертные оценки программы.
Правило 10.2 Используйте комбинацию самостоятельных экспертных оценок и экспертных оценок, выполненных компетентными сотрудниками, – для лучших результатов.
Правило 10.3 Осуществляйте самостоятельные экспертные оценки программы перед экспертной оценкой.
Правило 10.4 Используйте ВП Analyzer для автоматизации тщательных проверок.
Правило 10.5 Настройте критерии тестирования в ВП Analyzer.
Правило 10.6 Устанавливайте очередность каждого теста согласно приоритету каждого правила.
Правило 10.7 Используйте контрольный список, чтобы провести всесторонний анализ кода.
Правило 10.8 Проводите хотя бы одну экспертную оценку каждого проекта.
Правило 10.9 К обсуждению привлекайте следующих людей: менеджера проекта, ведущего разработчика, опытного разработчика и неопытного члена команды разработчиков.
Правило 10.10 На обсуждение принесите список требований и контрольный список правил стиля.
Правило 10.11 Назначьте нейтральную группу для записи желаемых изменений.
Правило 10.12 Не изменяйте исходный код во время экспертной оценки.

Предметный указатель

Иностранные термины

Online-документация, 346
Performance and Memory Profiler, 25
Select Alliance Partner, 49

А

Автоматическая обработка ошибок, 256
Автоматический режим соединения, 112
Анализатор ВП, 355
Архитектура приложения, 60

Б

Библиотека проекта LabVIEW, 62
Блок-диаграмма, 106

В

Вейвлеты пакетные, 131
Вкладка (Tab), 178
Вложение структур, 123
Вложенные структуры, 211
ВП верхнего уровня, 65, 242
ВП верхнего уровня (Top level VI), 67
ВП «Дотошный», 20
ВП «Многоуровневый», 22
ВП пользовательского интерфейса (GUI Vis), 68
ВП «Спагетти», 23
ВПП из блок-диаграммы, 96
Время разработки, 37
Всплывающая подсказка, 86
Выравнивание объектов, 113
Выровнять и расположить объекты, 70

Г

Глиф, 152
Группировка объектов, 70

Д

Дерево элементов (Tree), 179
Диалоговые ВП (Dialog VI), 67
Диапазон данных, 93
Динамическая объектная структура, 304
Динамический тип данных, 210
Документация, 331
Драйвер прибора, 54

Ж

Журнал проекта, 42

З

Заглавные буквы, 85
Закрашенный прямоугольник (Filled Rectangle), 154
Запись ошибок в лог-файл, 236
Злоупотребление цветами, 90
Зона разработчиков NI, 49

И

Иерархия ВП, 109
Иконка, 150
Имена по умолчанию, 64
Именование файлов LabVIEW, 64

К

Кластер, 179
Кластерs ошибок error out, 228
Коды ошибок, 239
Кольцевая структура (Ring), 178
Комбинированный элемент управления (bombo box), 179
Конструкты, 176
Контейнер .Net, 179
Контейнер activeX, 179

Л

Лицевая панель, 67
Логические переменные, 178, 193
Локализованные глифы, 158
Локальный терминал (sequence local), 121

М

Массив, 179
Матрица, 179
Меню быстрого доступа, 334
Метрика ВП, 109
Модули, 305
Модульная объектная структура приложения, 316

Н

Навигация по приложению, 92
Надежность, 30
Недопустимые данные, 93

О

Обработка ошибок, 32, 226
Объектная структура приложения, 304
Окно навигации, 108
Описание ВП, 345
Определение типа
- статус (Typedef Status), 120
- строгое (strict type definition), 120

Определение типа (type definition), 120
Опрос ресурса (polling), 128
Опции LabVIEW, 51
Отчет об ошибке, 234
Очереди (queue), 128
Очищение ошибок, 254

П

Панель виртуального подприбора, 79
Параллельные цепочки ошибок, 232
Параллельные циклы, 300
Передача номера итерации, 115
Перенос на другие платформы, 89
Перечень (Enumeration), 178
Печать документации к ВП, 349
Плотность размещения элементов, 76
Повторное использование кода, 53
Подготовка приложения, 39
Поисковик примеров NI Example Finder, 49
Показать сетку, 53
Показывать имена ВПП, 52
Показывать точки в контактах проводов, 52
Поток данных, 120
Префикс данных
- входных, 165
- выходных, 165

Префикс прибора, 65
Префикс прибора (instrument prefix), 158
Пробный ВП, 49
Проверка обособленности, 110
Проводник проекта (Project Explorer), 57
Программа взаимопомощи NI, 49
Проектная документация, 47
Производительность, 34
Промышленные ВП, 68
Простота, 34
Простота использования, 24
Простота поддержки, 29
Простые структуры данных, 177

Р

Разделяемая переменная одного процесса (single process shared variable), 124
Размещать терминалы как иконки, 52
Разрешение экрана, 71, 107
Разъем
- дополнительный (optional), 164
- обязательный (required), 164
- рекомендуемый (recommended), 164

Расположение (Layout), 68
Распространение кластера ошибок, 228
Редактор элементов, 95
Редактор элементов управления, 56

С

Свободные метки, 338
Свойства палитры, 55
Связанные в кластер данные, 116
Сдвиг требований, 40
Сдвиговые регистры (Shift registers), 131
Сертифицированные учебные центры NI, 49
Сетка, 53
Событие по истечении времени (timeout), 279
События (occurrence), 128
Соединительная панель (connector pane), 159
Сообщение об ошибке
- диалоговое окно, 234
- запись в бинарный файл, 236
- запись в текстовый файл, 236

Сообщество LAVA, 49
Справка
- CHM, 346
- HTML, 346
- PDF, 346

Ссылка (reference number), 296
Ссылка (refnum), 198
Статус определения типа, 95
Стили текста, 83
Строка (String), 179
Строки с полосой прокрутки, 339
Структура варианта со списком, 93
Структура завершения работы, 125
Структура папок проекта, 60

Т

Таблица, 179
Таблица совместимости типов данных, 183, 184
Тайпдеф (type definition, определение типов), 284
Терминал Discard? (Отменить?), 281
Техническое задание, 40
Типы данных, 176
Точка приведения типа (coercion dot), 122
Точки на узлах проводников, 122
Трехмерный стиль элементов управления, 52

У

Уведомители (notifiers), 129
Узел вызова динамической библиотеки (Call Library Function Node), 339
узел обратной связи (feedback node), 121
Узел обращения к динамической библиотеке Call Library Node, 244
Узел свойств (Property Node), 341
Универсальный тип данных (Variant), 179
Управление исходниками, 65
Уровни риска, 247
- высокий, 247
- низкий, 247
- средний, 248

Ф

Файл настроек, 53
Фокус ввода (Key focus), 94

Ц

Цикл программирования-отладки, 41

Ч

Черный периметр иконки, 151
Численное значение (Numeric), 178
Читабельность, 27

Ш

Шаблон
- автомат событий (event machine), 295
- ВП, 266
- ВПП с обработкой ошибок (SubVI with Error Handling), 270
- глобальный функционал (Functional Global), 271
- конечный автомат (State machine), 283
- конечный автомат, классический, 287
- конечный автомат, с очередью, 288
- конечный автомат, с событиями, 292
- начальный ВПП (Immediate subVI), 268
- непрерывный цикл (Continuous Loop), 272
- цикла с обработкой событий (event handling loop), 278

Шаблон ВП Диалог с событиями, 57
Шаблон ВП с обработкой ошибок, 56
Шаблон ТЗ, 44
Шаблоны ВП, 56
Шрифт
- диалог, 83
- приложения, 83
- системный, 83

Э

Экспертные оценки, 355
Эффективность, 25

Книги издательства «ДМК Пресс» можно заказать в торгово-издательском холдинге «АЛЬЯНС-КНИГА» наложенным платежом, выслав открытку или письмо по почтовому адресу: **123242, Москва, а/я 20** или по электронному адресу: **orders@alians-kniga.ru**.

При оформлении заказа следует указать адрес (полностью), по которому должны быть высланы книги; фамилию, имя и отчество получателя. Желательно также указать свой телефон и электронный адрес.

Эти книги вы можете заказать и в Internet-магазине: **www.alians-kniga.ru**.

Оптовые закупки: тел. **(495) 258-91-94, 258-91-95**; электронный адрес **books@alians-kniga.ru**.

Питер Блюм

LabVIEW:
стиль программирования

Главный редактор *Мовчан Д. А.*
dm@dmk-press.ru
Корректор *Кикава Л. В.*
Верстка *Чаннова А. А.*
Дизайн обложки *Мовчан А. Г.*

Подписано в печать 16.05.2008. Формат 70×100 $^{1}/_{16}$.
Гарнитура «Петербург». Печать офсетная.
Усл. печ. л. 37.5. Тираж 1000 экз.
№

Издательство ДМК Пресс
Web-сайт издательства: www.dmk-press.ru
Internet-магазин: www.alians-kniga.ru