Руководство разработчика CompactRIO

Рекомендуемые архитектуры LabVIEW и методы проектирования приложений управления механизмами

В этом документе дается обзор рекомендуемых архитектур и методы проектирования приложений управления механизмами с использованием контроллеров NI CompactRIO и компьютеров NI с сенсорными панелями.
Примеры и архитектуры разработаны в NI LabVIEW версий 8.6 и 2009.

Декабрь 2009

NATIONAL INSTRUMENTS

Содержание

РАЗДЕЛ 1
Обзор и исходные данные

Обзор руководства разработчика

В данном документе рассматривается архитектура и даются рекомендации по созданию управляющих приложений на базе контроллеров NI CompactRIO, работающих под управлением NI LabVIEW Real-Time Module версии 8.6 и выше. В нем объясняется, как вы можете использовать новые возможности CompactRIO – такие, как механизм сканирования (NI Scan Engine), обработчик ошибок (Fault Engine) и менеджер распределенных систем (Distributed System Manager), представленные в LabVIEW 8.6. CompactRIO содержит встроенные компоненты для упрощения разработки управляющих приложений; однако, та же базовая архитектура также работает на других платформах, в том числе Compact FieldPoint, PXI и контроллерах на базе Windows. LabVIEW Real-Time - полнофункциональный язык программирования, представляющий разработчику множество способов разработки контроллера и помогающий создать очень гибкие и сложные системы. Контроллеры LabVIEW Real-Time используются в различных приложениях, от управления стержнями на атомной электростанции, до программно-технического тестирования систем электронного управления двигателем, адаптивного управления бурением нефтяных скважин и высокоскоростного мониторинга вибраций при диагностировании отказов. Этот документ разработан, чтобы предложить концептуальную основу инженерам, разрабатывающим приложения для управления промышленными объектами, особенно тем, кто знаком с программируемыми логическими контроллерами (ПЛК), и предназначен для использования в качестве дополнительного руководства к стандартному курсу по LabVIEW Real-Time. Используйте это руководство, чтобы узнать, как создавать приложения LabVIEW Real-Time, объединяющие не только обычные для PLC возможности, но и гибкость, необходимую в таких нетрадиционных приложениях, как высокоскоростной буферизированный ввод-вывод, регистрация данных или машинное зрение.

Терминология

Вы можете использовать LabVIEW Real-Time для создания управляющего приложения различными способами, если понимаете следующие фундаментальные концепции программирования задач управления в реальном времени:

- *Быстрота реагирования* – управляющее приложение должно реагировать на события, такие, как изменение сигналов ввода-вывода, вводимых оператором данных или команд (Human Machine Interface – HMI) или изменение внутреннего состояния. Время, необходимое на выполнение действия после возникновения события, называется временем реагирования, и в различных управляющих приложениях к нему предъявляются различные требования, от микросекунд до минут. Для большинства промышленных приложений время реагирования должно быть в диапазоне от миллисекунд до секунд. Требуемое время реагирования является важным критерием управляющего приложения, потому что оно определяет быстродействие контура регулирования и влияет на ввод-вывод, процессор и программные решения.
- *Детерминизм и джиттер* – Детерминизм – это повторяемость временных характеристик контура регулирования. Джиттер, погрешность синхронизации – мера измерения детерминизма. Например, если контур регулирования настроен на запуск и обновление выходов раз в 50 мс, но иногда обновление выходов происходит через 50.5 мс, тогда джиттер составляет 0.5 мс. Повышенные характеристики детерминизма и надежности – главные преимущества управляющей системы реального времени, а высокий детерминизм критичен для стабильности работы приложения. Низкий детерминизм ведет к ухудшению характеристик аналогового управления и может сделать систему невосприимчивой.

- *Приоритет* – в большинстве контроллеров используется единственный процессор для выполнения всех задач управления, мониторинга и связи. Поскольку единственный ресурс (процессор) получает несколько параллельных запросов, необходимо найти способ управления наиболее важными запросами. Придав критическим контурам регулирования более высокий приоритет, вы можете получить полнофункциональный контроллер с высоким уровнем детерминизма и временем реагирования. Например, в приложении с контуром регулирования температуры и встроенной функцией регистрации, вы можете присвоить контуру регулирования приоритет более высокий, чем операциям регистрации, и обеспечить детерминированное управление температурой. Подобные меры гарантируют, что задачи с низким приоритетом, например, регистрация данных, Web-сервер, HMI и другие, не окажут отрицательного влияния на аналоговое управление или цифровую логику.

Обзор архитектуры систем управления механизмами

Системы управления механизмами, как правило, включают человеко-машинный интерфейс (HMI) и систему управления реального времени. Контроллеры реального времени обеспечивают надежное и предсказуемое поведение механизмов, в то время как HMI предоставляют оператору графический интерфейс пользователя (Graphical User Interface - GUI) для наблюдения за состоянием механизма и настройки параметров его функционирования. В типичной системе управления механизмами управляющая система строится на основе программируемого логического контроллера (Programmable Logic Controller - PLC) или программируемом контроллере автоматизации (Programmable Automation Controller - PAC). Базовые функции контроллеров включают:

- аналоговый и цифровой ввод-вывод
- табличную память для значений ввода-вывода и переменных (Tag)
- последовательностный автомат, который определяет поведение машины.

В дополнение к этим возможностям PLC, PAC National Instruments могут поддерживать более сложные функции, в том числе:

- высокоскоростной сбор и обработку данных
- управление движением
- функции машинного зрения и визуального контроля
- специализированную аппаратную обработку сигналов
- регистрацию данных

Вы можете программировать HMI на персональном компьютере (ПК) под управлениев Windows или на компьютере с сенсорной панелью со встроенной ОС наподобие Windows XP Embedded. HMI, как правило, реализует следующие функции:

- Операции с сенсорной панелью
- Система постраничного отображения с управлением навигацией
- Объекты ввода данных (кнопки, клавиатура и т.п.)
- Дисплеи и журналы тревог и событий

Конфигурации систем управления

Простейшая система управления механизмом состоит из единственного контроллера, работающего автономно (см. рисунок 1.1).

Эта конфигурация используется в приложениях, где не требуется HMI, кроме, как для техобслуживания и диагностических целей.

Рисунок 1.1. Автономный контроллер

На следующем уровне возможностей и сложности системы добавляется HMI и/или дополнительные узлы контроллеров (рисунок 1.2). Эта конфигурация типична для машин, управляемых оператором локально.

Рисунок 1.2. Локальная система управления машиной

Сложные приложения управления механизмами могут состоять из нескольких контроллеров и HMI (рисунок 1.3). Они часто включают также высокопроизводительный сервер, действующий как средство регистрации и пересылок данных. Подобная конфигурация системы поддерживает физически большие или сложные машины. С ее помощью вы можете организовать взаимодействие с машиной из разных мест или распределять конкретные функции мониторинга и управления среди группы операторов.

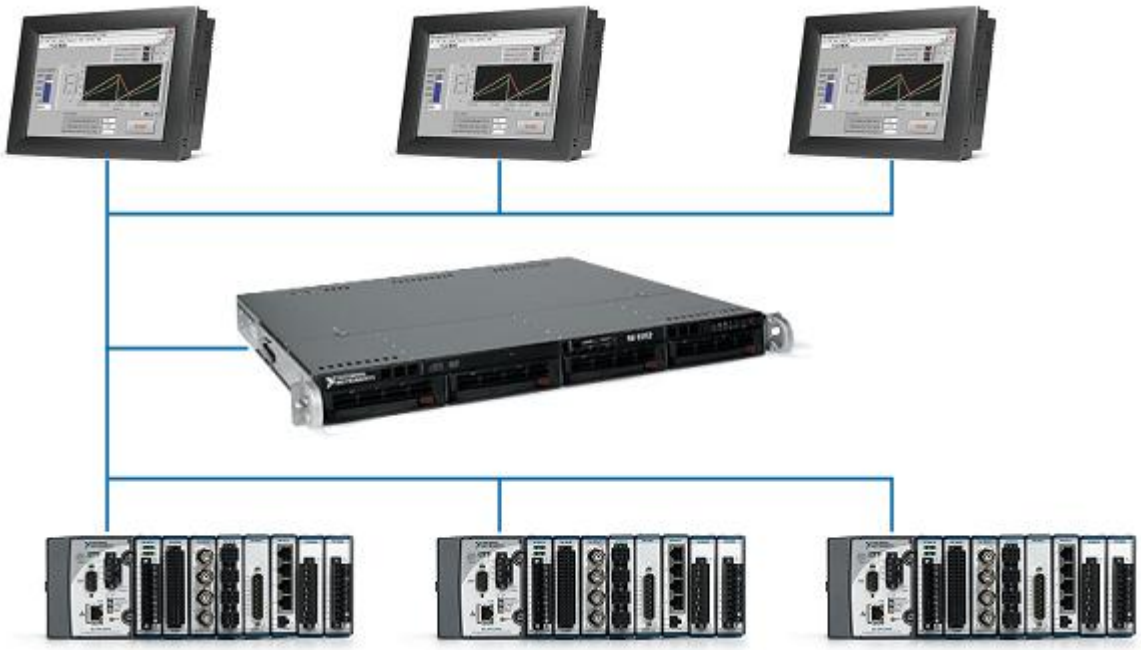


Рисунок 1.3. Распределенная система управления машиной

Блок-схема архитектуры управляющей системы

Управляющая система с PAC и HMI содержит все программные компоненты, необходимые для построения большинства приложений управления машинами. Знание того, как создаются базовые PAC и HMI, поможет вам создать систему управления механизмами любого масштаба.

1. Контроллер содержит компоненты для коммуникации и интерфейса с внешними устройствами, например, датчиками и актюаторами, HMI и сетевыми устройствами
2. Хранение текущих данных в табличной памяти (иногда называемой tag engine – движок переменных)
3. Выполнение логических алгоритмов управления механизмами или процессами
4. Выполнение служебных операций, например, при начальном запуске
5. Наблюдение и сообщение об ошибках системы

HMI содержит аналогичные компоненты, только вместо операций контроля реализует интерфейс пользователя. Вы можете осуществлять дополнительные задания, например, обнаружение и регистрацию тревог и событий, как на уровне контроллера, так и на уровне HMI.

Рисунок 1.4. Высокоуровневый вид архитектуры локального машинного управления
Housekeeping – служебные операции, Control/Meas Tasks – задачи управления и измерения, Memory Table – табличная память, IO and Comm Drivers – драйвера ввода-вывода и коммуникаций, Fault Engine – обработчик ошибок, Alarm Engine – обработчик тревог

Анализируя операции контроллера, вы можете разбить систему на мелкие компоненты, каждый из которых отвечает за конкретную задачу во всем приложении. На рисунке 1.5 показана архитектура контроллера и индивидуальные компоненты приложения управления машиной. Некоторые из этих компонентов готовы к работе как часть эталонной архитектуры управления машиной, в то время как другие должны быть созданы в процессе разработки и реализации конкретного управляющего приложения.

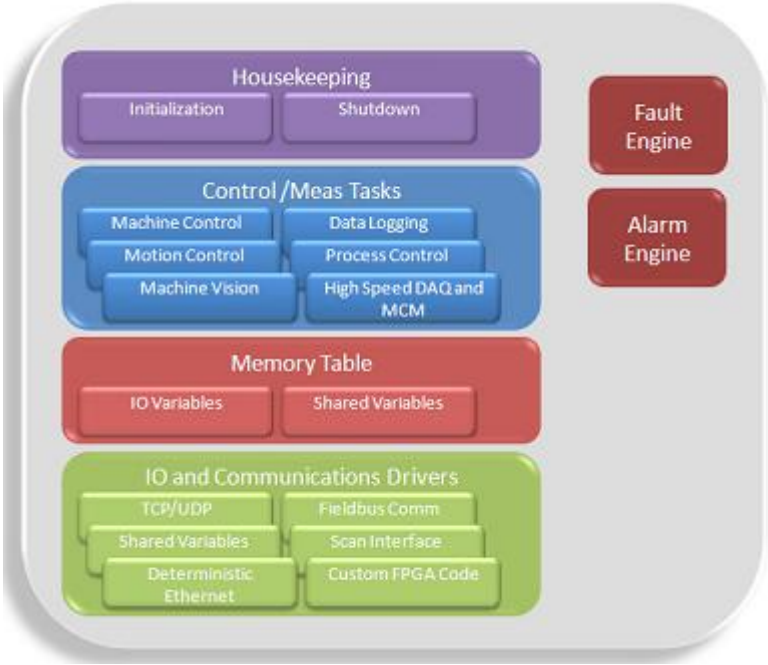


Рисунок 1.5. Архитектура и компоненты контроллеров
Housekeeping – служебные операции, Initialization - инициализация, Shut Down – выключение,

Control/Meas Tasks – задачи управления и измерения, Machine Control - управление машиной, Motion Control - управление движением, Machine Vision – машинное зрение, Data Logging – регистрация данных, Process Control – управление процессом, High Speed DAQ and MCM – высокоскоростные модули сбора данных и системы мониторинга состояния машин,

Memory Table – табличная память, IO Variables - переменные ввода-вывода, Shared Variables – переменные общего доступа,

IO and Comm Drivers – драйвера ввода-вывода и коммуникаций, Fault Engine – обработчик ошибок, Alarm Engine – обработчик тревог. TCP/UDP – протоколы TCP/UDP, Shared Variables – переменные общего доступа, Scan Interface - интерфейс сканирования, Custom FPGA Code – пользовательский код FPGA.

В данном документе рассмотрены рекомендуемые реализации различных компонентов архитектуры контроллера и HMI. Здесь также представлены примеры кодов, в некоторых случаях, альтернативные и компромиссные реализации.

Введение в CompactRIO

CompactRIO - промышленная реконфигурируемая встраиваемая система, содержащая три компонента - контроллер реального времени, реконфигурируемый программируемый массив вентилей (FPGA) и промышленные модули ввода-вывода.

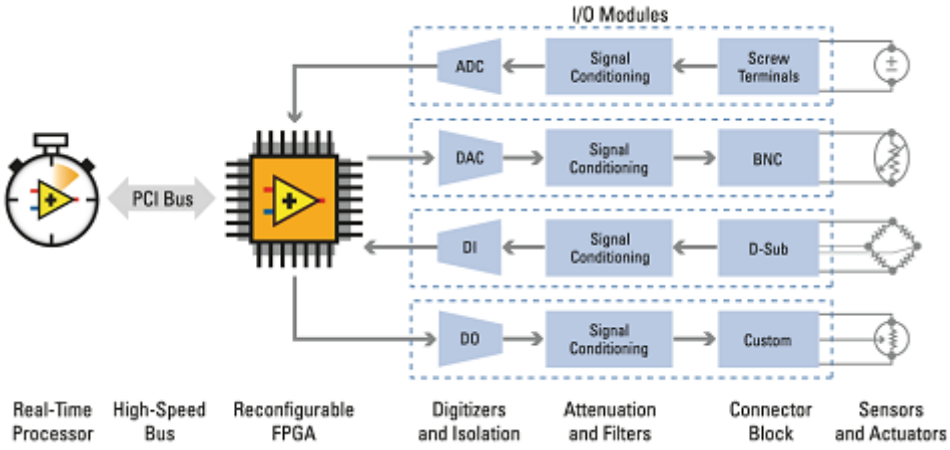


Рисунок 1.6. Архитектура реконфигурируемой встраиваемой системы

Real-Time Processor – процессор реального времени, High-Speed Bus – высокоскоростная шина, Reconfigurable FPGA – реконфигурируемая FPGA, Digitizers and Isolation – дискретизаторы и устройства изоляции, Attenuation and Filters – аттенюаторы и фильтры, Connector block – коннекторный блок, Sensors and Actuators – датчики и актюаторы , PCI Bus – шина PCI , ADC – АЦП , DAC - ЦАП, DI – цифровой ввод , DO – цифровой вывод, Signal Conditioning – преобразование (согласование) сигналов, Screw Terminals – терминалы с винтовыми клеммами, BNC, D-Sub и Custom – разъемы типа BNC, D-Sub и специальные

Контроллер реального времени

Контроллер реального времени содержит промышленный процессор, который с высокой степенью надежности и детерминизма выполняет приложения LabVIEW Real-

Time и обеспечивает управление с разными скоростями, трассировку выполнения, встроенную регистрацию данных и связь с периферийными устройствами.

Дополнительные возможности включают резервные входы питания от 9 до 30 В постоянного тока, таймер реального времени, аппаратные сторожевые таймеры, сдвоенной порт Ethernet, накопитель данных до 2 Гб и встроенные порты USB и RS232.

Рисунок 1.7. Контроллер реального времени NI cRIO-9014

Рисунок 1.8. Реконфигурируемое шасси на основе FPGA

Реконфигурируемое шасси FPGA

Реконфигурируемое шасси с FPGA – сердце архитектуры встраиваемых систем. Реконфигурируемые каналы ввода-вывода (RIO) на основе FPGA напрямую подключены к модулям

ввода-вывода, что обеспечивает высокопроизводительный доступ к цепям ввода-вывода каждого модуля и неограниченную гибкость синхронизации и запуска. Поскольку каждый модуль подключен к FPGA непосредственно, а не через шину, вы практически не испытываете задержек управления, связанных с реакцией системы, в отличие от прочих промышленных контроллеров. По умолчанию эти FPGA автоматически связываются с модулями ввода-вывода и предоставляют детерминированный ввод-вывод для процессора реального времени. Новые FPGA обеспечивает программе или контроллеру реального времени доступ к вводу-выводу с джиттером между циклами менее 500 нс. Вы можете также перепрограммировать FPGA для выполнения пользовательского кода. Благодаря быстродействию FPGA это шасси часто используется для создания управляющих систем, включающих высокоскоростной буферизированный ввод-вывод, очень быстрые контуры регулирования или пользовательскую фильтрацию сигналов. Например, при использовании FPGA в одном шасси можно одновременно реализовать более 20 аналоговых пропорционально-интегрально-дифференциальных (ПИД) контуров регулирования с частотой 100 кГц. Кроме того, поскольку в FPGA весь код выполняется на аппаратном уровне, это обеспечивает высокую надежность и детерминизм, которые идеально подходят для аппаратных блокировок, пользовательской синхронизации и запуска, или для исключения специализированных схем, обычно необходимых вместе со специальными сенсорами.

Промышленные модули ввода-вывода

Модули ввода вывода содержат схемы гальванической развязки и преобразований, согласования сигналов и встроенную средства для прямого подключения к промышленным сенсорам и актюаторам. Благодаря широкому набору вариантов подключения и интеграции коннекторного блока в модули, система CompactRIO значительно снижает требования к размерам и уменьшает стоимость подключения. Вы можете выбирать более чем из 50 модулей ввода-вывода C-серии для CompactRIO с возможностью подключения практически любого сенсора или актюатора.

Различные типы модулей позволяют подключать термопары; выполнять одновременную дискретизацию сигналов в диапазоне ±10 V, 24-битный аналоговый ввод-вывод; цифровой ввод-вывод сигналов промышленных уровней 24 В с током нагрузки до 1 А; принимать цифровые дифференциальныеl/ТТЛ сигналы; проводить измерения сигналов от IEPE акселерометров с разрешением 24 разряда, измерения деформаций; измерения температуры с помощью резистивных датчиков; измерение мощности, формировать сигналы на аналоговых выходах; подключаться к сети контроллеров (CAN); использовать SD-карты цифровой защиты при регистрации. Кроме того, это открытая платформа, то есть вы можете создавать ваши собственные модули или покупать модули других производителей. С помощью комплекта разработчика модулей NI cRIO-9951 CompactRIO вы можете создавать собственные модули под нужды конкретного приложения. Комплект разработчика предоставляет доступ к архитектуре нижнего, электрического уровня встраиваемых систем CompactRIO для разработки специализированного ввода-вывода, коммуникаций и модулей управления. Он включает также библиотеки LabVIEW FPGA для интерфейса со схемами вашего специализированного модуля.

Рисунок 1.9. Вы можете выбирать более чем из 50 модулей ввода-вывода для CompactRIO с возможностью подключения практически любого сенсора или актюатора

Спецификации CompactRIO

Многие потребители CompactRIO создают системы, которые поставляются и используются по всему миру. Упрощает процесс разработки систем для глобального использования наличие у CompactRIO множества сертификатов и результатов тестирования независимых агентств.

Description	Standard
Electromagnetic Compatibility (EMC)	89/336/EEC EN 55011 Class A at 10 m FCC Part 15A above 1 GHz Industrial levels per EN 61326-1:1997 + A2:2001, Table A.1 CE, C-Tick, and FCC Part 15 (Class A) Compliant
Product Safety	73/23/EEC EN 61010-1, IEC 61010-1 UL 61010-1 CAN/CSA C22.2 No. 61010-1
Hazardous Locations, Class I, Division 2	Class I, Division 2, Groups A, B, C, D, T4; Class I, Zone 2, AEx nC IIC T4, EEx nC IIC T4
Shock and Vibration	IEC 60068-2-64, IEC 60068-2-27,IEC 60068-2-6
Mean Time Before Failure (MTBF)	Bellcore Issue 6, Method 1, Case 3 MIL-HDBK-217F
Marine	Lloyds Register (LR Type Approval System Test Spec No. 1)
Quality/Environmental Management System (QMS/EMS)	ISO 9001/14001

Typical Certifications – Actual specifications vary from product to product. Visit ni.com/certification for details.

Рисунок 1.10. Спецификация CompactRIO

РАЗДЕЛ 2
Базовая архитектура для управления

Сведения о базовой архитектуре контроллера

Для создания сложных систем нужна архитектура, которая позволяет повторно использовать код, масштабировать систему и управлять выполнением программы.

В следующих двух разделах описывается, как строится базовая архитектура управляющих приложений и как создается простой контур ПИД-регулирования с использованием этой архитектуры.

Базовая архитектура контроллера имеет три главных состояния:

1. Инициализация (служебная операция)
2. Управление (драйверы ввода-вывода и коммуникации, таблица памяти, задачи измерения и управления)
3. Выключение (служебная операция)

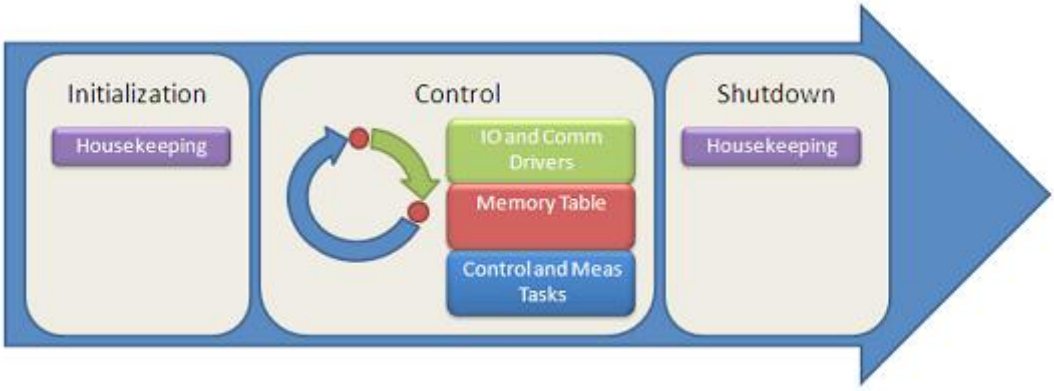


Рисунок 2.1. Три главных состояния базовой архитектуры контроллера

Initialization - инициализация, Housekeeping – служебные операции,

Control – управление, IO and Comm Drivers – драйвера ввода-вывода и коммуникаций, Memory Table – таблица памяти, Control/Meas Tasks – задачи управления и измерения,

Shut Down – выключение.

Подпрограмма инициализации

До выполнения основного контура регулирования программа должна выполнить инициализацию. Подпрограмма инициализации подготавливает контроллер к выполнению программы, и здесь не место логическим операциям, относящимся к самой машине, например, операциям запуска или инициализации машины. Эти логические операции должны выполняться в основном контуре регулирования. При выполнении подпрограммы инициализации:

1. Всем внутренним переменным присваиваются значения по умолчанию.
2. Создаются программные структуры, необходимые для работы. Это могут быть очереди, буферная память реального времени (FIFOs), ссылки на VI и загрузка битовых файлов в FPGA.
3. Выполняется дополнительная заданная пользователем логика для подготовки контроллера к работе, например, подготовка файлов журнала.

Подпрограмма управления

Ввод-вывод, связь, таблица памяти

Многие программисты знакомы с прямым доступом к вводу-выводу, при котором подпрограммы непосредственного посылают/получают значения ввода-вывода в/из аппаратных средств. Этот метод идеален при сборе данных, обработке сигналов и для небольших одноканальных приложений. Однако управляющие приложения обычно используют одноточечные чтение и запись, и могут стать очень большими, с большим количеством состояний - каждому из которых необходим доступ к вводу-выводу.

При доступе к вводу-выводу возникает дополнительная нагрузка на систему, которая может замедлить ее функционирование. Кроме того, управление сразу несколькими операциями доступа к вводу-выводу через все уровни программ делает очень сложным изменение режимов ввода-вывода и реализацию таких операций, как симуляция или форсирование. Чтобы избежать этих проблем, управляющая программа использует архитектуру сканирования ввода-вывода. При этом доступ к физическим аппаратным средствам осуществляется один раз за итерацию цикла с использованием драйверов ввода-вывода и коммуникаций (обозначенных как IO and Comm Drivers на рисунке 2.1). Значения входа и выхода сохраняются в таблице памяти, а задачи управления и измерения обращаются в область памяти вместо непосредственного доступа к аппаратным средствам. Эта архитектура имеет следующие преимущества:

- Абстрагирование ввода-вывода, так что вы можете повторно использовать subVI (никакого жесткого программирования ввода-вывода)
- Низкая дополнительная загрузка
- Детерминированные операции
- Поддержка симуляций
- Поддержка "форсирования"
- Исключения риска изменения ввода-вывода во время выполнения логических операций

Задачи управления и измерения

Задачи управления и измерения представляют собой машинно-ориентированную логику, определяющую алгоритм управляющего приложения. Это может быть процесс простого управления или более сложное управление механизмом. Во многих случаях эта логика основана на конечном автомате (state machine) для отработки сложной логики с множеством состояний. Далее описывается, как использовать конечный автомат для реализации логики управления. Главная управляющая задача должна:

- Выполняться за меньшее время, чем сканирование ввода-вывода.
- Осуществлять доступ к вводу-выводу через таблицу памяти ввода-вывода вместо непосредственного чтения и записи каналов ввода-вывода.
- Не использовать циклы while, кроме как для сохранения информации о состоянии в сдвиговых регистрах
- Не использовать циклы for, кроме как в алгоритмах
- Не использовать задержки wait, а использовать функции таймера или «Tick Count» для логики синхронизации
- Не выполнять операции с сигналами, регистрацию данных или недетерминированные операции (использовать параллельные циклы с низким приоритетом для этих операций).

Логика пользователя может

- Включать одноточечные операции, например, ПИД-регулирование или обработку "по точкам".
- Использовать конечный автомат для структуризации кода

Блок-схему алгоритма программы управления можно представить в виде одного цикла, где выполняется ввод и вывод, и задачу управления, связывающиеся через таблицу памяти; но реально это несколько синхронизированных циклов, в которых может быть более одной задачи управления и измерения.

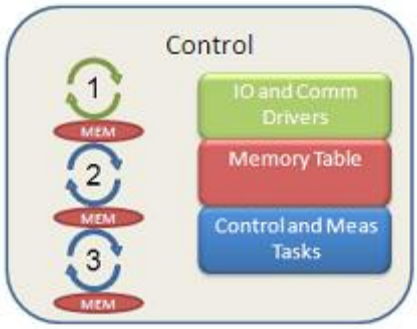


Рисунок 2.2. Три главных состояния контроллера базовой архитектуры
Control – управление, IO and Comm Drivers – драйвера ввода-вывода и коммуникаций, Memory Table – таблица памяти, Control/Meas Tasks – задачи управления и измерения, MEM – память

Подпрограмма выключения

Когда контроллер должен остановиться по команде или из-за ошибки, он останавливает основной контур регулирования и запускает подпрограмму выключения. Подпрограмма выключения останавливает контроллер и переводит его в безопасное состояние. Используют ее только для остановки контроллера – здесь не место для подпрограмм выключения машины, которые должны выполняться в основном контуре регулирования.

Подпрограмма выключения:

1. Устанавливает все выходы в безопасное состояние
2. Останавливает все работающие параллельные циклы
3. Выполняет некоторые дополнительные логические операции, например, уведомление оператора о сбоях контроллера или регистрация информации о состоянии.

Пример базовой архитектуры контроллеров в LabVIEW

В данном разделе приведен пример кода LabVIEW

Чтобы продемонстрировать эту управляющую архитектуру, создадим приложение простого ПИД-регулирования. Это простое приложение должно поддерживать температуру в температурной камере равной 350 °F. Приложение использует ПИД-алгоритм для управления с одним аналоговым входом для термопары и одним цифровым выходом ШИМ (широтно-импульсной модуляции)

Излишне упрощенное приложение здесь используется, чтобы объяснить компоненты архитектуры без добавления запутанного примера управления. Более сложные примеры с использованием этой архитектуры будут рассмотрены позднее.

Чтобы создать это приложение в LabVIEW, используем пять компонентов архитектуры контроллера:

1. Подпрограмму инициализации
2. Подпрограмму отключения

3. Простую задачу управления процессом
4. Переменные ввода-вывода в таблице памяти
5. Интерфейс RIO Scan для доступа к вводу-выводу.

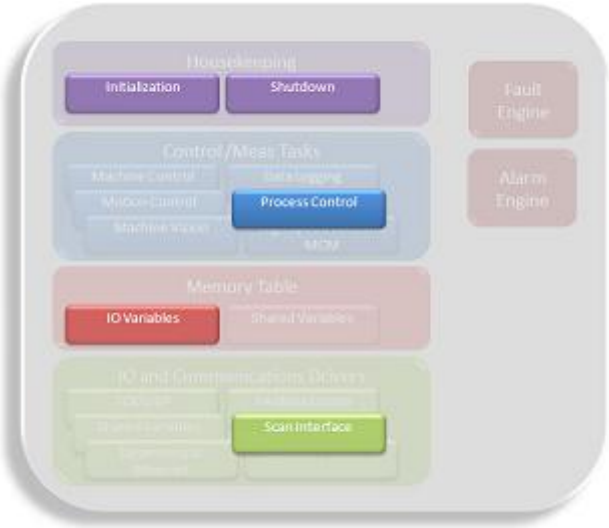


Рисунок 2.3. Пример архитектуры ПИД-контроллера

Housekeeping – служебные операции, Initialization - инициализация, Shut Down – выключение,

Control/Meas Tasks – задачи управления и измерения, Machine Control - управление машиной, Motion Control - управление движением, Machine Vision – машинное зрение, Data Logging – регистрация данных, Process Control – управление процессом, High Speed DAQ and MCM – высокоскоростные модули сбора данных и системы мониторинга состояния машин, Memory Table – таблица памяти, IO Variables - переменные ввода-вывода, Shared Variables – переменные общего доступа,

IO and Comm Drivers – драйвера ввода-вывода и коммуникаций,

Fault Engine – обработчик ошибок, Alarm Engine – обработчик тревог, TCP/UDP – протоколы TCP/UDP, Shared Variables – переменные общего доступа, Scan Interface - интерфейс сканирования, Custom FPGA Code – пользовательский код FPGA

Инициализация и выключение

1. Сначала добавьте подпрограммы инициализации и выключения. Подпрограмма инициализации должна сконфигурировать контроллер, чтобы он готов был к выполнению любых логических операций, а подпрограмма выключения нужна для выполнения любых действий, требуемых при выключении.
2. Чтобы выполнить эту управляющую последовательность, создайте структуру последовательности (sequence structure) с тремя кадрами (frames): один для программы инициализации, один – для задач контроля и измерения, и один – для программы выключения.

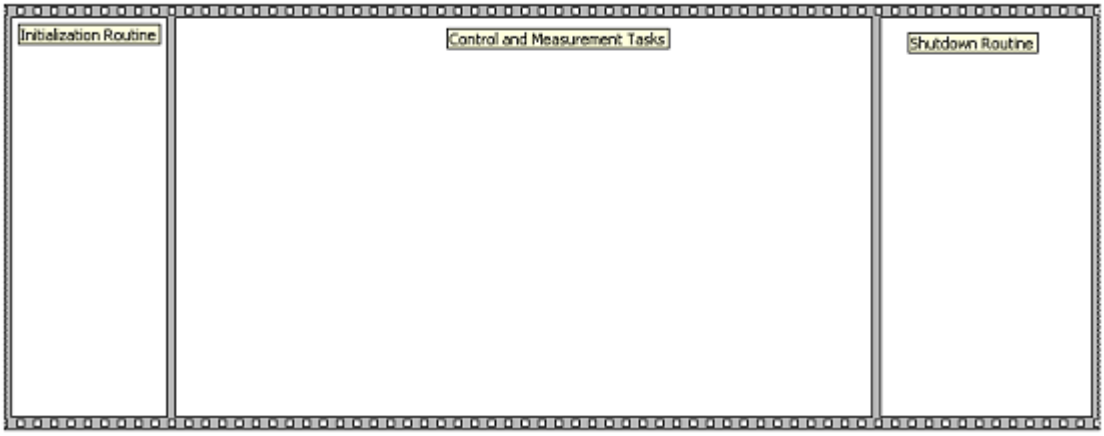


Рисунок 2.4. Управление этой последовательностью действий контроллера тремя кадрами: подпрограмма инициализации, задачи управления и измерения и процедура выключения

3. Добавьте логику инициализации или выключения. Для данного приложения инициализация контроллера не требуется. По умолчанию контроллер оставляет выходы в их последнем состоянии. В этом приложении при выключении нужно установить выходы в состояние "выключено". Вы также можете добавить другую логику, например, регистрацию ошибок.

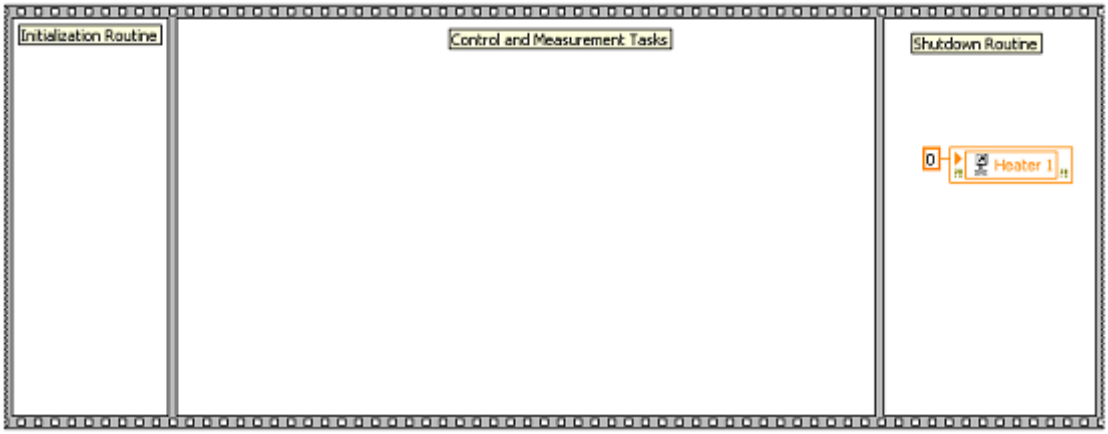


Рисунок 2.5. Добавьте любую логику инициализации или выключения

Теперь у вас есть готовые подпрограммы инициализации и выключения, и вам необходимо добавить задачи управления и измерения.

Сканирование ввода-вывода и таблицы памяти

Начиная с LabVIEW версии 8.6, у вас есть режим программирования CompactRIO, называемый интерфейс сканирования RIO (RIO Scan Interface). Когда вы открываете свой контроллер CompactRIO в проекте LabVIEW, у вас есть возможность запрограммировать контроллер через интерфейс сканирования или интерфейс LabVIEW FPGA. (Если у вас не установлен LabVIEW FPGA, по умолчанию LabVIEW выбирает интерфейс сканирования).

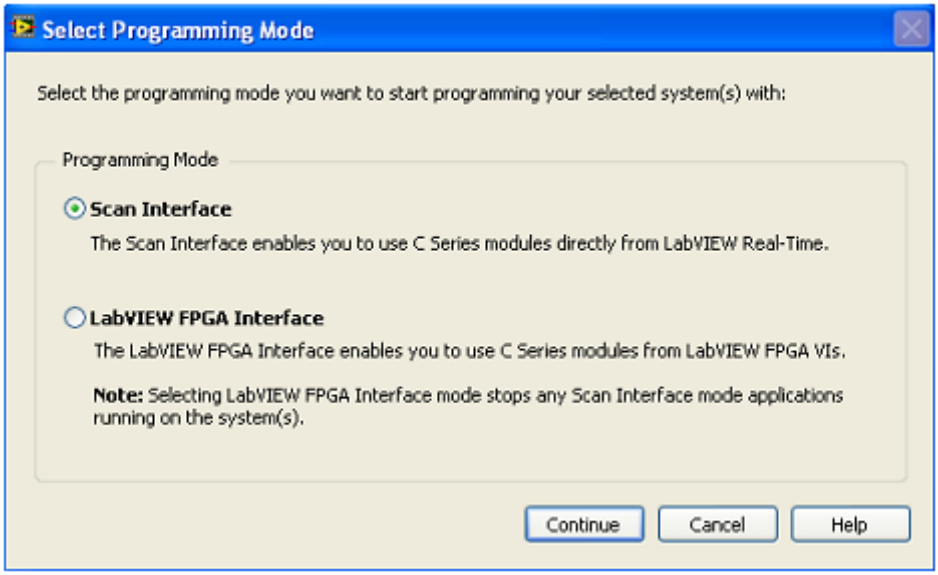


Рисунок 2.6. Начиная с LabVIEW 8.6 вы можете программировать CompactRIO с использованием интерфейса сканирования

Когда контроллер связывается с подсистемой ввода-вывода через интерфейс сканирования, модуль ввода-вывода автоматически считывает данные из модулей и помещает их в таблицу памяти в контроллере CompactRIO. По умолчанию время сканирования ввода-вывода составляет

10 мс и может быть настроена в свойствах контроллера. Вы можете обратиться к подсистеме ввода-вывода с помощью псевдонимов переменных ввода-вывода (I/O variables aliases).

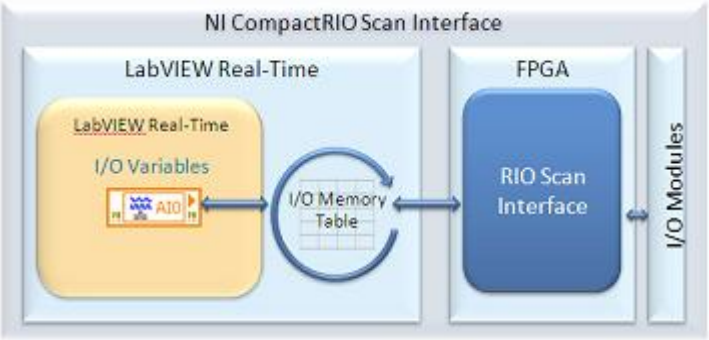


Рисунок 2.7. Блок-схема программных компонентов интерфейса сканирования CompactRIO
NI CompactRIO Scan Interface – интерфейс сканирования NI CompactRIO, I/O Variables – переменные ввода-вывода, I/O Memory Table – таблица памяти ввода-вывода, RIO Scan Interface – интерфейс сканирования RIO, I/O Modules – модули ввода-вывода.

В вашей системе есть модуль измерения сигналов термопары и выходной модуль ШИМ. Вы можете конфигурировать их и обращаться к ним через интерфейс сканирования. Чтобы читать и записывать значения из LabVIEW, создайте элементам псевдонимы ввода-вывода. Псевдоним ввода-вывода ссылается на физический ввод-вывод, который вы можете использовать для дополнительного масштабирования и поддержания компактности кода.

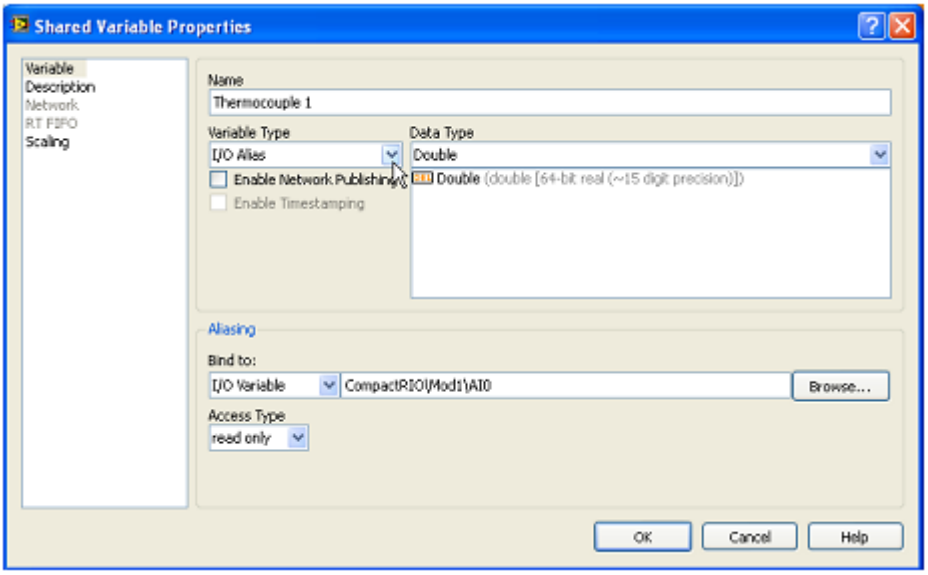


Рисунок 2.8. Создание псевдонима ввода-вывода

1. Чтобы создать псевдоним ввода-вывода, щелкните правой кнопкой мыши по контроллеру и выберите новую переменную (new variable). Выберите тип переменной I/O Alias (псевдоним ввода-вывода) и привяжите ее к физическому вводу-выводу.
2. Для этого примера создайте два псевдонима ввода-вывода Thermocouple 1 (привяжите к модулю термопары) и Heater 1 (привяжите к модулю цифрового вывода, сконфигурированного на формирование ШИМ-сигнала), и поместите их в библиотеку IO Library.

Задачи управления и измерения

Process Control

Составьте расписание каждой задачи управления и измерения при помощи синхронизируемого цикла (timed loop). Вы должны синхронизировать цикл timed loop со сканированием ввода-вывода (NI Scan Engine) для обеспечения должной синхронизации между управляющей задачей и вводом-выводом.

1. Создайте синхронизируемый цикл и настройте его на синхронизацию с механизмом сканирования. Оставьте период равным 1, чтобы цикл запускался каждый раз при запуске сканирования ввода-вывода.

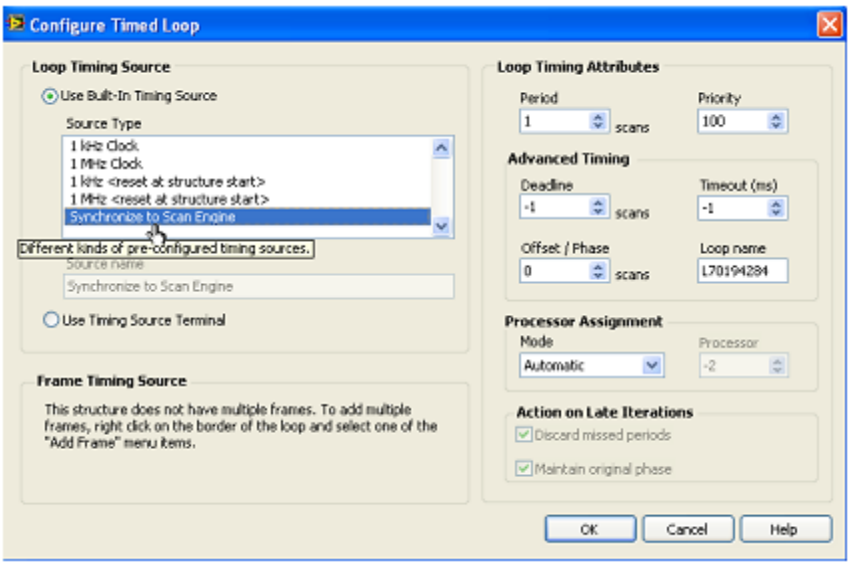


Рисунок 2.9. Синхронизация NI Scan Engine

2. Создайте управляющую логику для чтения входов с псевдонимов ввода-вывода, выполнения логики и записи в псевдонимы ввода-вывода. Обычно для инкапсуляции кода и возможности его повторного использования создаются subVI. Однако, поскольку цель этого примера – показать общую архитектуру, код тривиален и его дальнейшая инкапсуляция излишня. В последующих примерах вы узнаете о правильной инкапсуляции кода. В этом простом примере поместите PID VI на блок-диаграмму и подключите константы, чтобы выходной диапазон (output range) составлял [100, 0], коэффициенты ПИД-регулятора (PID gains) - [10, 0.1, 0], а уставка (setpoint) составляла 350. Вместо констант вы можете использовать переменные, конфигурируемые во время выполнения программы. Подключите псевдоним ввода-вывода «Temperature 1» к терминалу «Process Variable», а псевдоним «Heater 1» I/O к терминалу «Output». Добавьте подходящие обработчики ошибок.

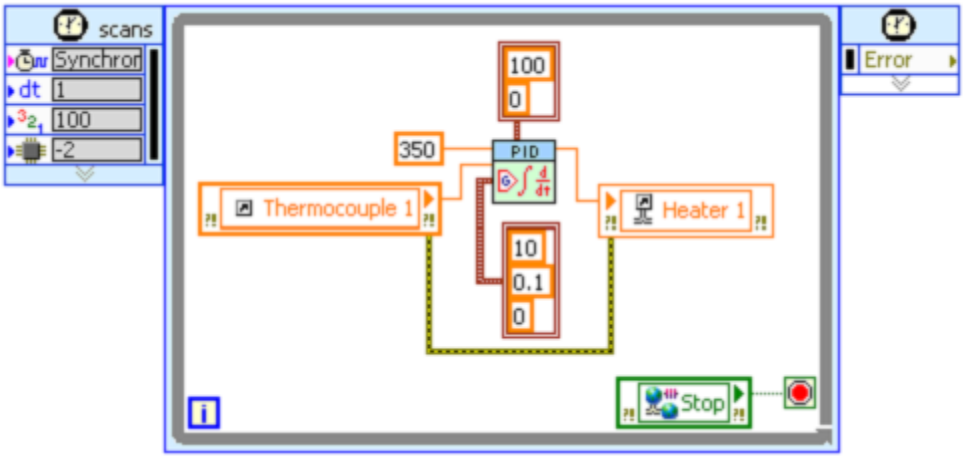


Рисунок 2.10. В этом примере используется публикуемая в сети переменная общего доступа для останова цикла

Теперь вы можете запустить программу управления температурой. Она содержит программы инициализации и выключения, сканирующую архитектуру, повторно используемые subVI, которые не привязаны жестко к вводу-выводу, и обработку ошибок. Это базовая архитектура для управления машиной, используемая в этом документе.

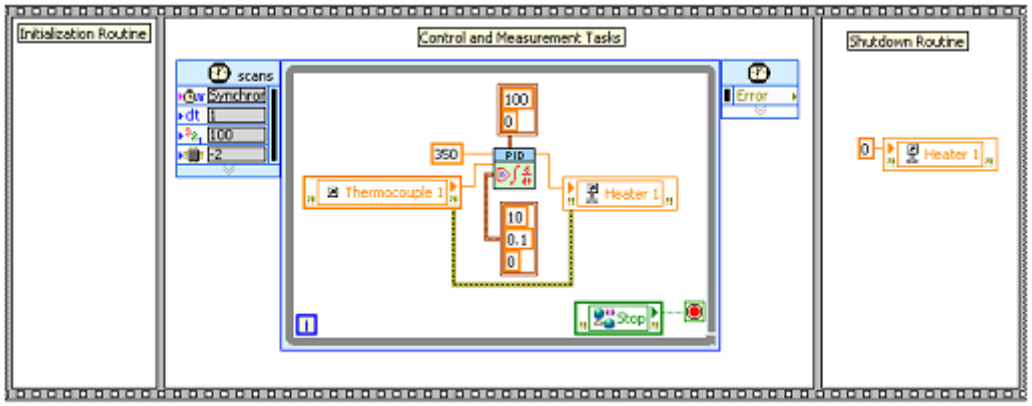


Рисунок 2.11. Базовая архитектура управления механизмами

Проектирование на основе состояний

С помощью этой базовой архитектуры вы можете создавать сложные приложения управления механизмами. Однако, по мере того, как логика становится более сложной, важно использовать нужную архитектуру для организации вашей разработки.

Применяя программную архитектуру, вы можете создавать расширяемые и легко поддерживаемые приложения. Создание архитектуры, в которой система представлена последовательностью состояний, является распространенным методом разработки расширяемого и легко управляемого кода.

Общее представление о конечном автомате

Конечный автомат (машина состояний) – широко распространенная и полезная программная архитектура. Вы можете использовать шаблон проекта конечного автомата для реализации любого алгоритма, который может быть детально описан диаграммой состояний или блок-схемой алгоритма. Конечный автомат обычно демонстрирует алгоритм принятия решений средней сложности, например, в программе диагностики или при контроле за ходом процесса.

Конечный автомат состоит из набора состояний и функции перехода из текущего состояния в следующее. Каждый конечный автомат должна разрабатываться так, чтобы выполнялись действия по входу в состояние и выходу из него. Поскольку конечные автоматы используются как часть большей архитектуры системы управления, в них не могут использоваться состояния ожидания и

циклы, за исключением сохранения состояний или выполнения алгоритмов, таких, как с циклами for для манипуляций с массивами.

Используйте машины состояний в приложениях, где существуют различимые состояния. Если вы можете разбить приложение на несколько зон операций, конечный автомат – хороший выбор архитектуры. Каждое состояние может приводить к одному или нескольким состояниям или же завершать поток процессов. В конечном автомате используются ввод данных пользователем или внутренние вычисления для определения того, в какое состояние переходить в следующий момент. Многие приложения требуют состояния инициализации, следующее за состоянием по умолчанию, где вы можете производить различные действия. Эти действия зависят как от предыдущих и текущих значений входа, так и от состояний. Состояние выключения обычно используется для выполнения действий по "очистке".

Пример сценария разработки с использованием конечного автомата

Чтобы увидеть, какие преимущества получает приложения при использовании архитектуры конечного автомата, разработаем систему, управляющую резервуаром для химической реакции. В этом приложении контроллер должен:

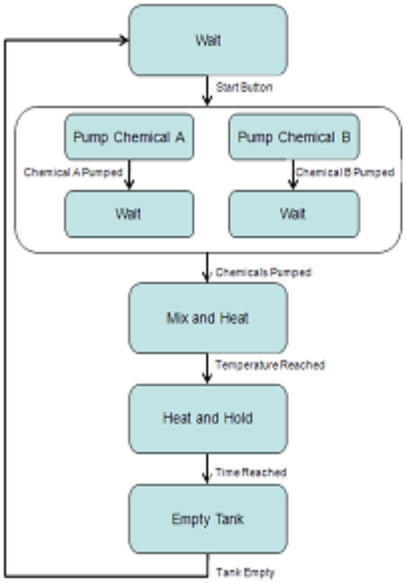
1. Подождать, пока оператор даст команду, нажав на кнопку.
2. Измерять два потока реагентов в резервуаре на основе выходного значения счетчика расхода (два параллельных процесса, по одному для каждого потока).
3. После заполнения резервуара включить миксер и повысить температуру в резервуаре. Как только температура достигнет 200 °F, выключить миксер и удерживать температуру постоянной 10 секунд.
4. Вылить содержимое в бак сборки.
5. Вернуться в состояние ожидания.

Отметим, что для простоты приложения, скорости потоков химических реагентов были жестко закодированы на значение 859, температура – на 200 °F, а время – на 10 секунд. В реальном приложении вы можете загружать эти значения из рецепта или их может ввести оператор.

Пример машины состояний в LabVIEW

В данном разделе приведен пример кода LabVIEW

Первым шагом при создании этого приложения является планирование логики и точек ввода-вывода. Поскольку это приложение содержит последовательность шагов, для его разработки хорошо подходит блок-схема. Ниже представлена блок-схема для этого приложения и список сигналов ввода-вывода.



I/O Signals	I/O Name
Operator push button	Input_Operator_PB
Pump A	Output_PumpA
Pump B	Output_PumpB
Chemical A Flow	Input_ChemA_Flow
Chemical B Flow	Input_ChemB_Flow
Stirrer	Output_Stirrer
Heater	Output_Heater
Thermocouple	Input_TC
Drain Pump	Ouput_PumpDrain
Tank Empty Level Sensor	Input_TankEmpty_LS

Рисунок 2.12. Блок-схема приложения и список сигналов ввода-вывода

Wait – ожидание, Pump Chemical A (B) – насос реагента A (B), Chemical A (B) pumped– закачивание реагента A (B), Chemical Pumped – реагенты закачаны, Mix and Heat – перемешивание и нагрев, Temperature Reached – температура достигнута, Heat and Hold – нагрев и удержание, Time Reached – время выдержано, Empty Tank – выкачивание из резервуара, Tank Empty – резервуар пуст

Operator push button – оператор нажал кнопку, Input_Operator_PB – ввод кнопки оператора; Pump A (b) – насос A (B), Output_PumpA (B) – вывод на насос A (B); Chemical A (B) Flow – расход реагента A (B), Input_ChemA (B)_Flow – ввод расхода реагента A (B); Stirrer – миксер, Output_Stirrer – вывод на миксер; Heater – нагреватель, Output_Heater вывод на нагреватель; Termocouple – термопара, Input_TC – ввод с термопары; Drain Pump – выходной насос, Output_PumpDrain – вывод на выходной насос; Tank Empty Level Sensor – датчик сигнала "резервуар пуст", Input_TankEmpty_LS – ввод с датчика "резервуар пуст"

Каждое состояние автомата состояний выполняет уникальное действие и вызывает другие состояния. Переходы между состояниями зависят от некоторых условий или последовательностей их выполнения. Перевод диаграммы переходов-состояний в блок-диаграмму LabVIEW требует следующих компонентов:

- Структура Case – содержит выбор (фрейм) и код для выполнения в каждом состоянии
- Сдвиговый регистр (Shift register) – содержит информацию о переходах между состояниями
- Код выполняемой в состоянии функции – реализует функцию состояния
- Код перехода (Transition code) – определяет следующее состояние в последовательности

Как только вы определите состояния системы, используйте структуру case в LabVIEW для представления и помещения логики для каждого состояния.

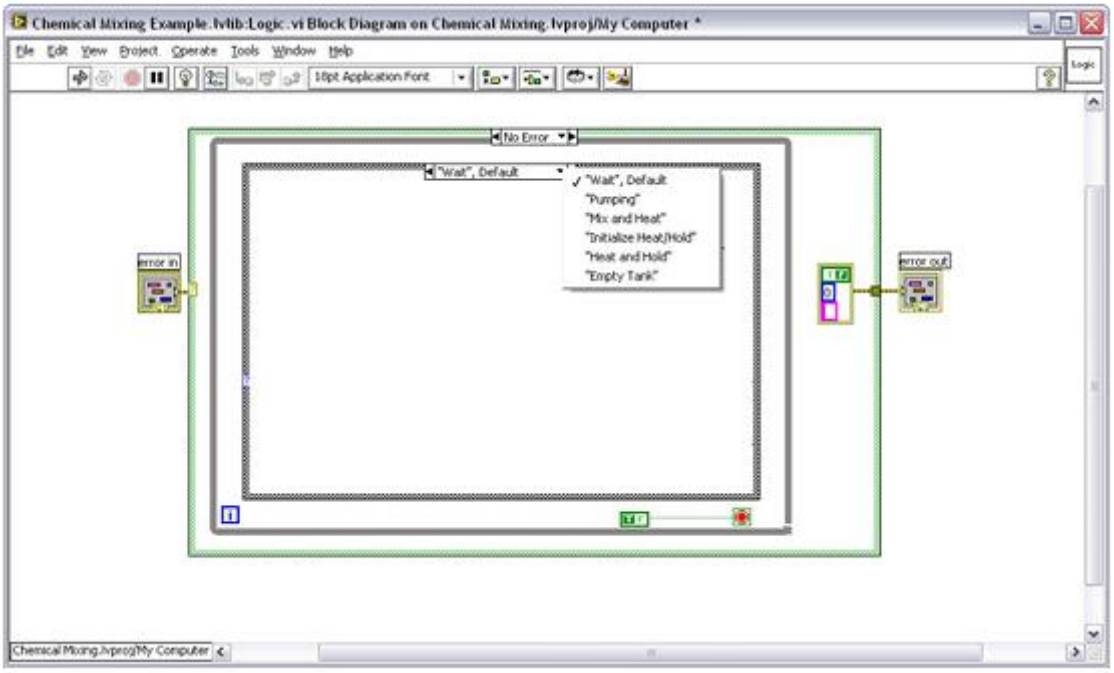


Рисунок 2.13. Создание case для каждого состояния автомата

Добавляя в приложение сдвиговый регистр, вы можете отслеживать и передавать информацию о текущем состоянии каждый раз при выполнении автомата состояний. Входной терминал Case-структуры подключен к сдвиговому регистру.

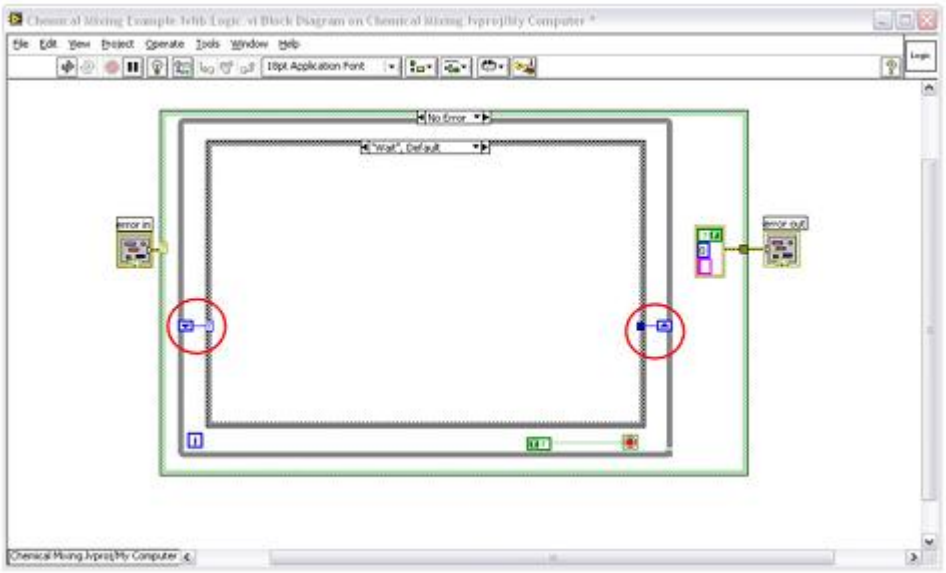


Рисунок 2.14. Добавьте сдвиговый регистр для передачи данных о состоянии

Каждое состояние в пределах case-структуры теперь содержит некий код LabVIEW , исполняемый каждый раз, как состояние становится активным. Код на рисунке 2.15 выполняет ПИД-регулирование для смешивания и нагревания вашего резервуара с химическими реагентами.

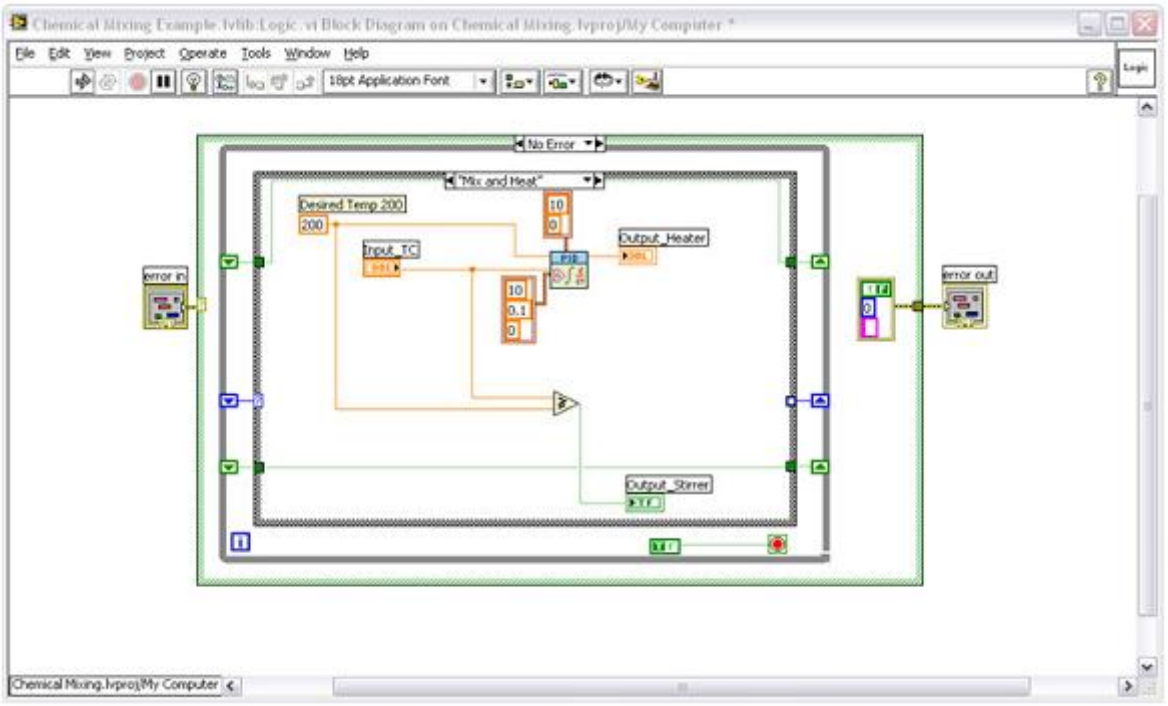


Рисунок 2.15. Добавьте вашу логику в каждое состояние

Каждое состояние должно определять, в какое состояние переходить дальше в зависимости от внутренних условий системы. Добавьте логику перехода для определения того, какое состояние должно выполняться следующим.

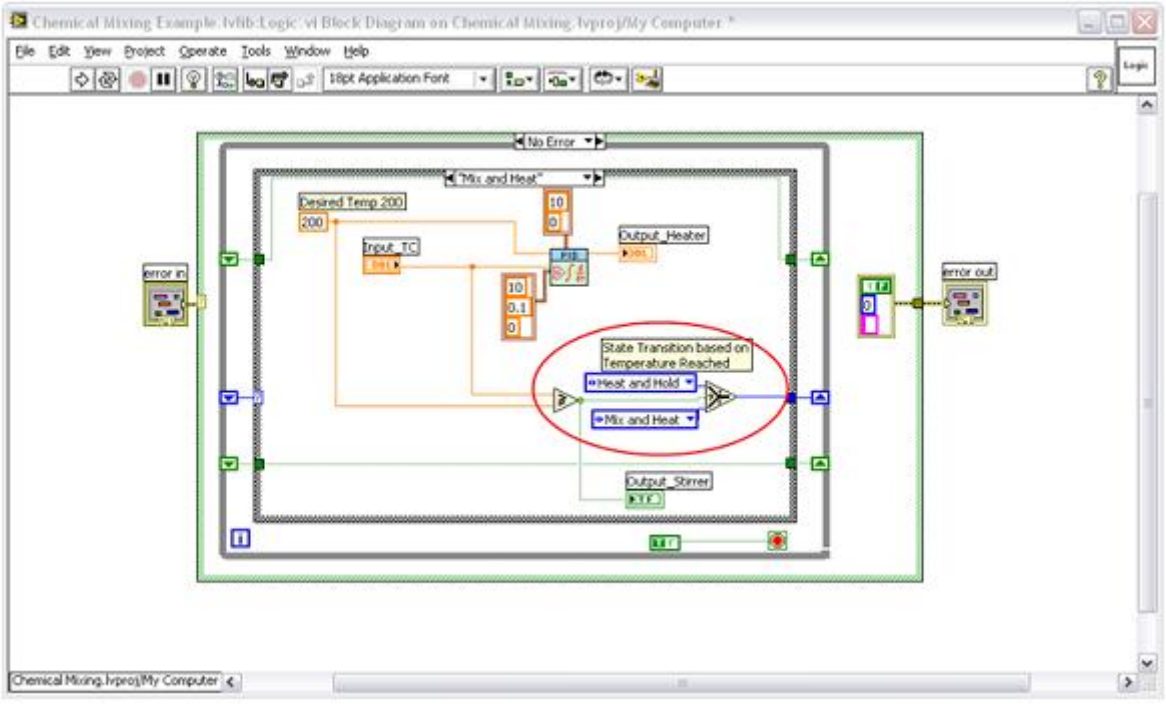


Рисунок 2.16. Используйте селектор для определения следующего фрейма case

В этом примере вместо физического ввода-вывода используйте симуляцию для тестирования вашей логики. Чтобы это сделать, используйте глобальную переменную вместо VI, читающих и записывающих на аппаратном уровне. Это удобный способ проверить вашу логику в интерактивном режиме с помощью управляющих элементов и индикаторов, прежде чем перейти к настоящим аппаратным средствам. Возможность легкого симулирования ввода-вывода – одно из преимуществ подобной архитектуры.

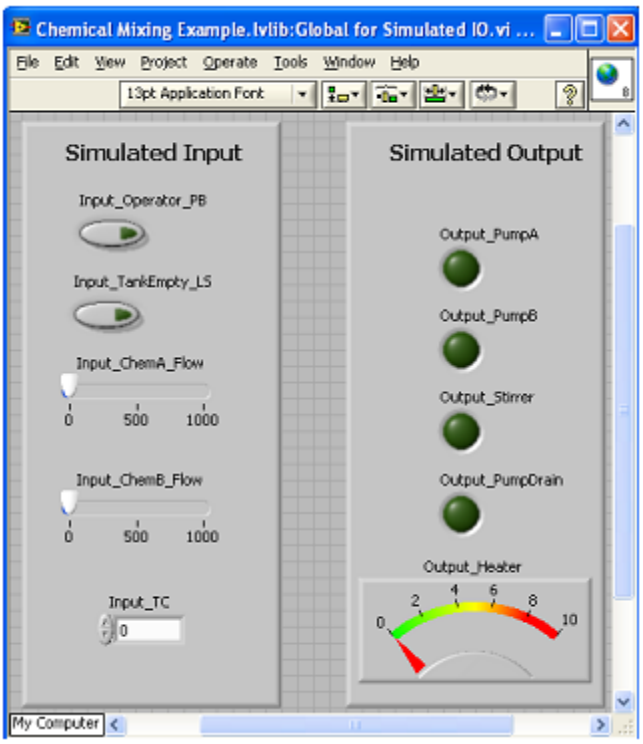


Рисунок 2.17. Глобальные переменные – удобный способ тестирования логики без применения аппаратных средств

Поскольку одно состояние вашего автомата содержит параллельные процессы, вам нужно создать второй конечный автомат для реализации параллельной логики и вызова параллельных процессов в этом состоянии. Выходите из состояния только после завершения обоих параллельных процессов.

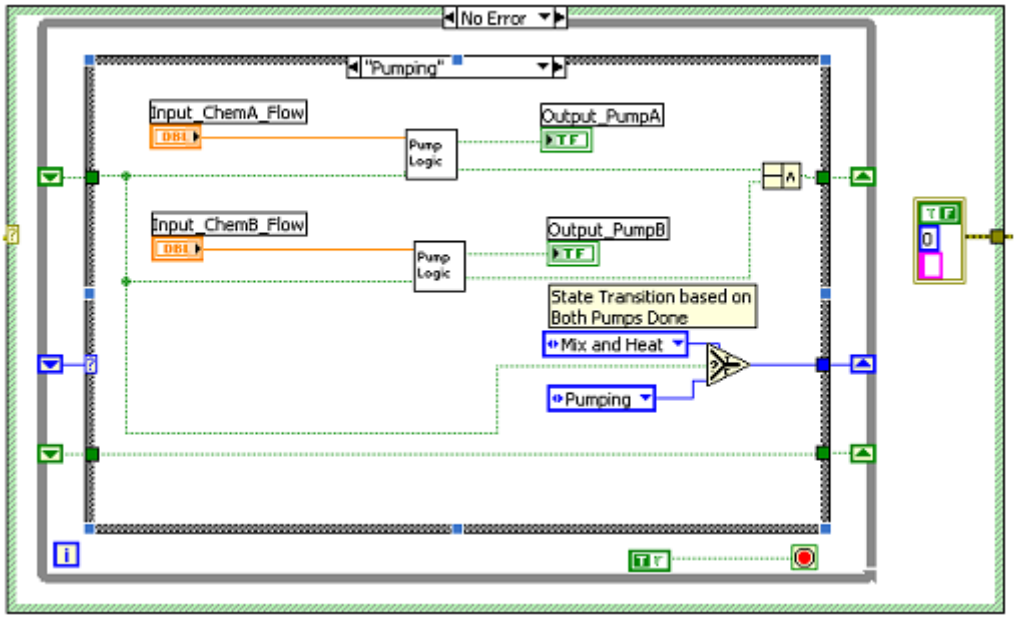


Рисунок 2.18. Параллельная логика может выполняться в одном состоянии

Введение в диаграммы состояний

Конечный автомат – лишь один метод для отображения в программном обеспечении диаграмм, основанных на состояниях. По мере того, как система становится более сложной, необходимо переходить к более высоким уровням абстракции, чтобы создать удобный в сопровождении программный проект.

Диаграммы состояний (Statecharts) предлагают расширенную функциональность конечного автомата и свойства, улучшающие масштабируемость приложения. Диаграммы состояний предлагают набор инструментов графического программирования высокого уровня и обеспечивают системный уровень представления задачи, полностью описывая функционирование системы или приложения. Использование диаграмм состояний помогает организовать программные приложения так, чтобы свести к минимуму непредсказуемое поведение путем гарантирования учета всех возможных состояний. Модель программирования, основанная на диаграммах состояний, особенно полезна для реактивных систем, которые характеризуются их реакцией на входы. Диаграммы состояний напоминают графические программы тем, что они управляются потоком данных (dataflow), обладают свойством самодокументирования и обеспечивают простую передачу знаний между разработчиками. Новый член команды разработчиков может посмотреть на диаграмму состояний и быстро понять все элементы системы.

Для понимания диаграмм состояний лучше начать с классической диаграммы, состоящей из двух основных конструкций: состояний и переходов. На рисунке 2.19 для иллюстрации приведена диаграмма состояний, описывающая работу простого автомата по продаже напитков с пятью состояниями и семью переходами. Автомат начинает работу с состояния «ожидание» и переходит в состояние "подсчет монет" при вводе монет. На диаграмме состояний показаны дополнительные состояния и переходы, когда машина ждет выбора напитка, затем выдает газировку и, наконец, выдает сдачу.

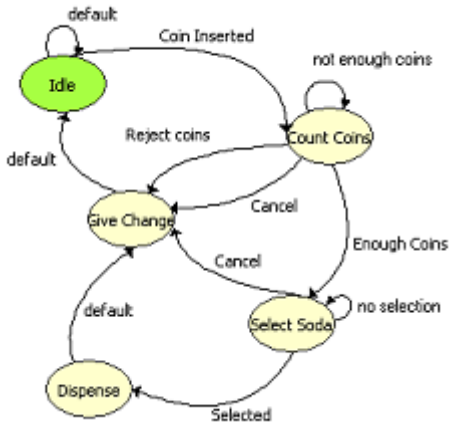


Рисунок 2.19. Диаграмма состояний простого автомата по продаже газировки
Idle – ожидание, Count Coins – подсчет монет, Give Change – выдать сдачу, Select Soda – выбрать напиток, Dispense – выдача напитка, Coin Inserted – ввод монет, Not Enough Coins – не хватает монет, Enough Coins – монет достаточно, no selection – выбора нет, Selected – выбрано, Default – по умолчанию, Cancel – отмена.

На рисунке 2.20 показана диаграмма состояний, описывающая поведение такой машины. На ней вы можете поместить "подсчет монет" и "разлив" в одно суперсостояние, используя свойство иерархии, что позволяет состояниям быть включенным одно в другое. При помощи иерархии вы можете упростить вашу разработку. Теперь вам необходимо определить только один переход (T3) от каждого из этих состояний в ответ на три события: напиток выдан, требуется сдача, или выброс монет. Обратите внимание, что состояние "выбор напитка" в классической диаграмме здесь было удалено. Это достигается путем использования защитного условия (guard condition) в переходе T2. Со сторожевыми условиями вы можете встраивать логику в переход. Защитное условие должно оказаться "истинным", чтобы произошел переход. Если защитное условие "ложно", событие игнорируется, и перехода не происходит.

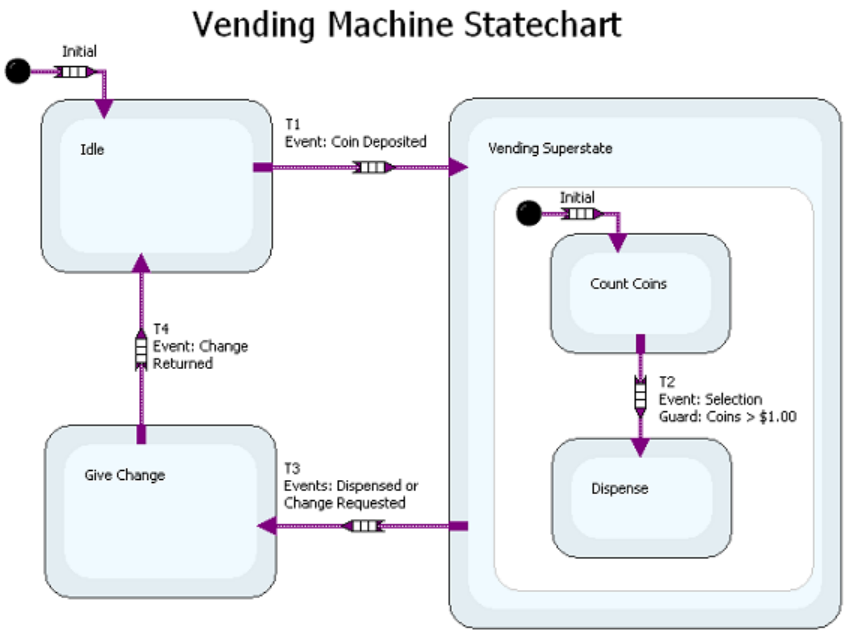


Рисунок 2.20. Диаграмма состояний простого автомата по продаже напитков

В предыдущем примере показан самый простой торговый автомат. Вы можете усложнить вашу систему, добавив требования наблюдения за температурой автомата, в то же время продолжая подсчет монет и выдачу газировки. С помощью диаграмм состояний вы легко можете расширить функциональность проектируемых программ. Они обладают свойством параллелизма, что позволяет диаграмме состояний находиться в нескольких состояниях в одно и то же время. Эти состояния называются ортогональными (orthogonal) или и-состояниями (and-states). Применяя параллелизм в вашей диаграмме состояний, вы можете инкапсулировать логику выдачи и температурный контроль в и-состоянии. И-состояния описывают систему, находящуюся одновременно в двух независимых друг от друга состояниях. Переход T7 показывает, как в диаграмме состояний можно определить выход, применяемый к обоим субдиаграммам.

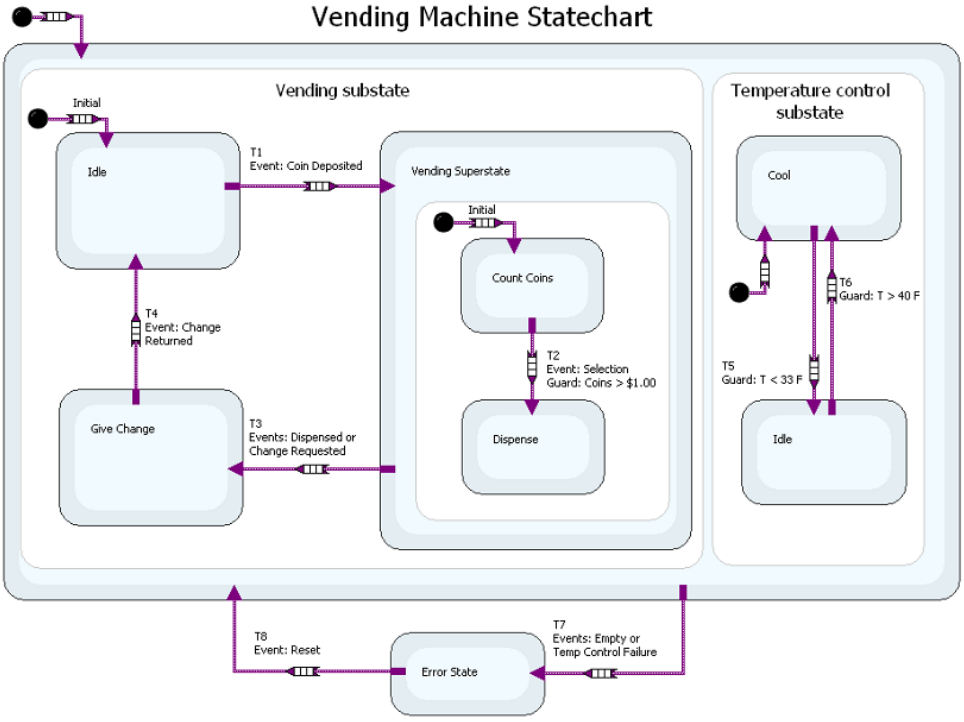


Рисунок 2.21. Переход T7 показывает, как в диаграмме состояний можно определить выход, применяемый к обоим субдиаграммам

Помимо иерархии и параллелизма, диаграммы состояний обладают свойствами, делающих их ценными для сложных систем. Им свойственна концепция истории, позволяющая суперсостоянию "помнить", какое егог внутренне субсостояние было ранее активно. Например, рассмотрим суперсостояние, описывающую механизм, который разливает некую жидкость и затем подогревает ее. Событие паузы может остановить работу машины в процессе наливания. Когда происходит событие возобновления, машина "вспоминает", что нужно продолжить разлив.

Руководство по модулю LabVIEW Statechart

LabVIEW Statechart Module – редактор LabVIEW, в котором вы можете быстро построить полноценный автомат состояний. Он поддерживает иерархию, так что вы можете использовать сразу несколько диаграмм состояний и VI LabVIEW в сложном приложении. Диаграммы состояний работают в Windows, на платформах реального времени и FPGA. Они содержат **регионы, состояния, псевдосостояния, переходы и коннекторы**.

Регионы диаграмм состояний

Регион – это область, содержащая **состояния**. Диаграмма состояний верхнего уровня представляет собой регион, в котором помещаются состояния. Кроме того, вы можете создавать регионы внутри состояний, чтобы воспользоваться преимуществом иерархии и создать состояние внутри другого состояния. Эта возможность иллюстрируется рисунком 2.22, где субсостояние создается внутри состояния с использованием региона. Каждый регион должен содержать **инициирующее** псевдосостояние.

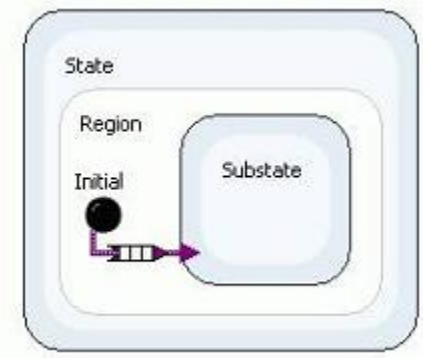


Рисунок 2.22. Создание субсостояния внутри состояния с использование региона

Состояния диаграмм состояний

Состояние – это основной объект диаграммы состояний. Состояния должны находиться внутри регионов и иметь хотя бы один входящий переход.

Рисунок 2.23 Состояние на диаграмме состояний

Каждое состояние имеет связанные с ним **действия при входе и при выходе из состояния. Действие при входе в состояние** – код LabVIEW, выполняемый при входе в состояние. **Действие при выходе из состояния** – код LabVIEW, выполняемый при выходе из состояния (перед тем, как произойдет переход в следующее состояние). Каждое состояние имеет только одно действие при входе и при выходе, при этом оба не являются обязательными. **Действие при входе и/или при выходе** выполняется каждый раз, как происходит вход в состояние или выход из него.

Вы можете получить этот код через диалоговое окно **Configure State** (Конфигурирование состояния).

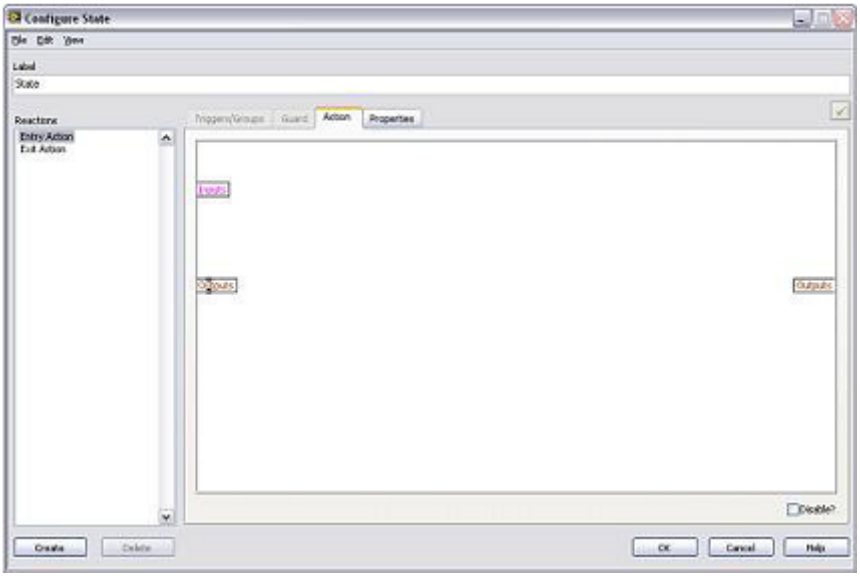


Рисунок 2.24. Вы можете получить код для входа или код для выхода через диалоговое окно Configure State

Далее вы можете настроить состояния на **статические реакции**, при которых состояние осуществляет действие, когда нет входящих или исходящих переходов. Индивидуальное состояние может иметь несколько статических реакций, которые могут выполняться на каждой итерации диаграммы состояний.

Каждая статическая реакция состоит из трех компонентов – триггера, сторожа и действия.

Триггер – это событие или сигнал, вызывающее реакцию диаграммы состояний. **В синхронных диаграммах состояний** триггеры автоматически передаются диаграммам состояний через периодический интервал. По умолчанию значение триггера установлено в NULL.

Сторож – часть кода, которая выполняется перед выполнением действия состояния. Если сторож принимает значение "истина", то код действия выполняется. Если сторож принимает значение "ложь", то действие не выполняется.

Если диаграмма состояний получает триггер, который должен быть обработан конкретной статической реакций, а сторож принимает истинное значение, то реакция выполняет код действия. **Действие** – это код LabVIEW, реализующий желаемую логику состояния. Это может быть чтение входов или внутренняя информация состояния и соответствующее модифицирование выходов.

Вы можете создать статические реакции в диалоге **Configure State**, создав новую реакцию. Как только вы это сделаете, вы можете связать ее с триггером и добавить сторож и код действия. Только статические реакции могут иметь триггер и сторож.

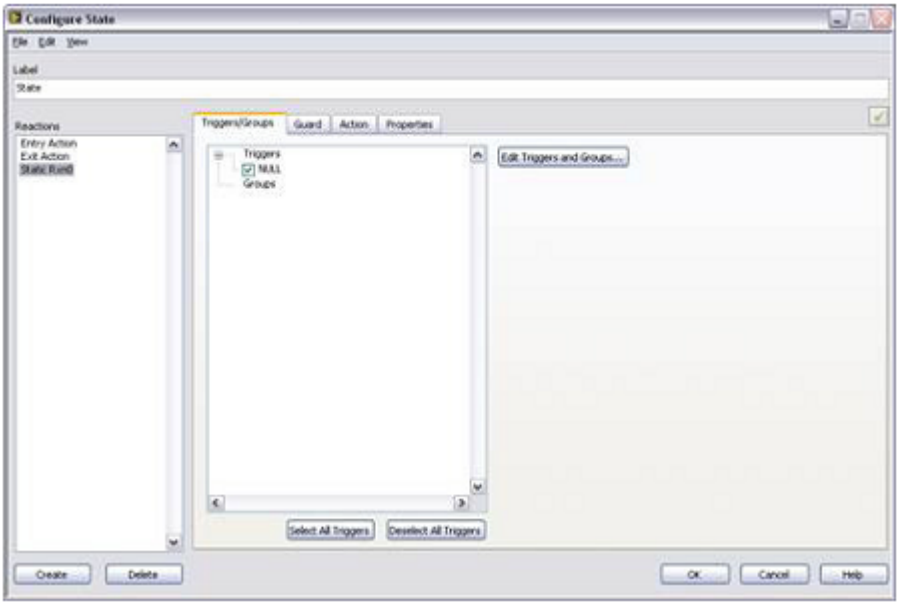


Рисунок 2.25. Вы можете создать статические реакции, создавая новую реакцию в диалоге Configure State

Ортогональные регионы и параллелизм

Когда состояние содержит два или более регионов, регионы должны быть ортогональными, как регионы 1 и 2 на рисунке 2.26.

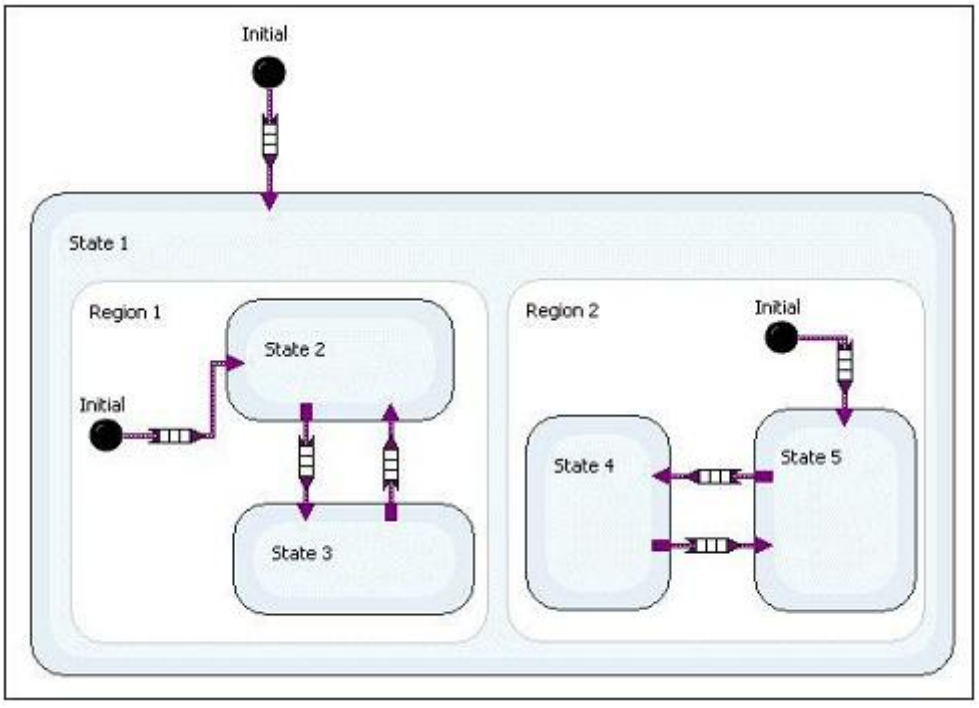


Рисунок 2.26. Регионы 1 и 2 - ортогональные

Субсостояния в ортогональных регионах параллельны, что значит, что, как только активируется суперсостояние, диаграмма состояний может находиться только в одном субсостоянии из каждого ортогонального региона во время каждой итерации диаграммы состояний.

Переходы

Переходы определяют условия, при которых происходит переход в новое состояние.

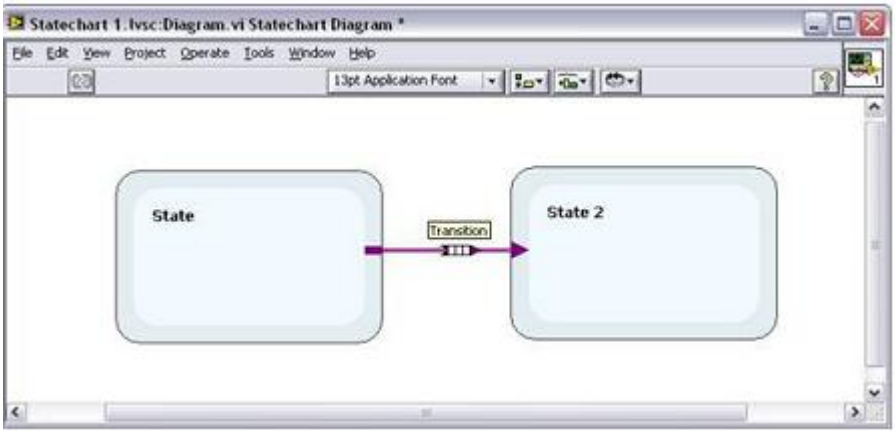


Рисунок 2.27. Переходы определяют условия, при которых происходит переход в новое состояние

Переходы состоят из **портов** и **узлов перехода**. Порты – это соединения между состояниями, а узлы переходы определяют поведение перехода с помощью триггеров, сторожей и действий. Узлы переходы настраиваются в диалоге **Configure Transition** (Настройка переходов).

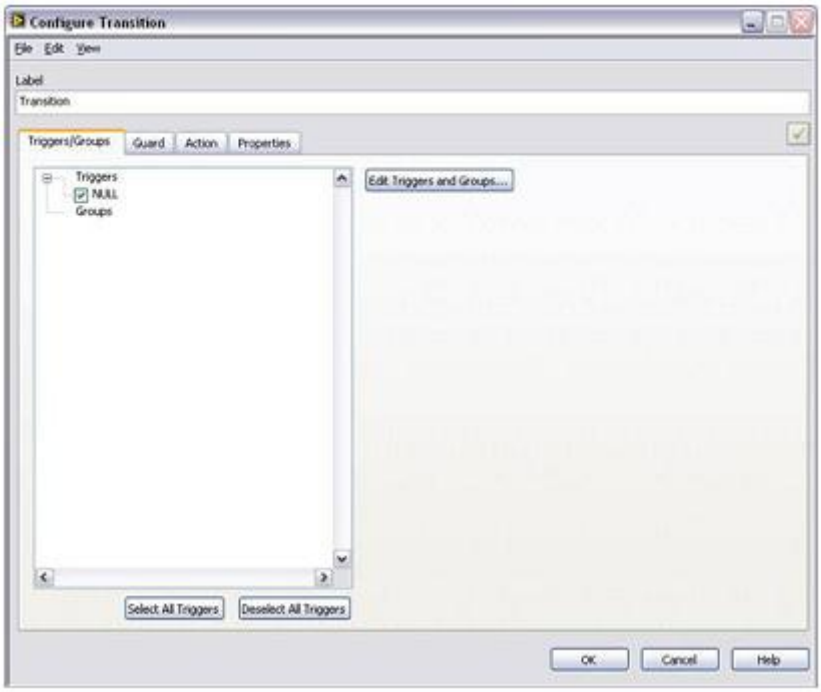


Рисунок 2.28. Узлы переходов настраиваются в диалоге Configure Transition

Триггеры, **сторожа** и **действия** в переходах ведут себя так же, как в состояниях. Переход отвечает на триггер, и, если выполнение кода сторожа дает истину, действие выполняется, и диаграмма состояний переходит в следующее состояние. Если выполнение кода сторожа дает ложь, код действия не выполняется и диаграмма состояний не переходит в новое состояние.

Псевдосостояния

Псевдосостояние – объект диаграммы состояний, представляющий состояние. Модуль LabVIEW Statechart содержит следующие псевдосостояния:

- **Начальное состояние** (Initial state) – отображает первое состояние, которое возникает при входе в регион. Начальное состояние должно присутствовать в каждом регионе.
- **Конечное состояние** (Terminal state) – отображает финальное состояние региона и завершает исполнение всех состояний в этом регионе.
- **Недавняя история** (Shallow history) – определяет, что при выходе из региона и обратном возвращении в него диаграмма состояний входит в субсостояния самого верхнего уровня, которые были активны, когда диаграмма состояний покинула регион.

- **Давняя история** (Deep history) – задает, что при выходе из региона и обратном возвращении в него диаграмма состояний входит в субсостояния самого нижнего уровня, которые были активны, когда диаграмма состояний покинула регион.

Коннекторы

Коннектор – объект диаграммы состояний, содержащий множество сегментов переходов.

LabVIEW Statechart Module содержит следующие коннекторы:

- Расщепление (Fork) – разделяет один сегмент перехода на несколько сегментов.
- Слияние (Join) – объединяет несколько сегментов перехода в один сегмент.
- Объединение (Junction) – соединяет несколько сегментов перехода.

Пример диаграммы состояний в LabVIEW

В данном разделе приведен пример кода LabVIEW

Чтобы продемонстрировать преимущества LabVIEW Statechart Module, используем предыдущий пример, но с конечным автоматом. Контроллер должен:

1. Подождать, пока оператор инициирует команду, нажав на кнопку.
2. Измерять два потока реагентов в резервуар на основе выходного значения счетчика расхода (два параллельных процесса, по одному для каждого потока).
3. После заполнения резервуара включить миксер и повышать температуру в резервуаре. Как только температура достигнет 200 °F, выключить миксер и поддержать температуру постоянной 10 секунд.
4. Вылить содержимое в бак сбора.
5. Вернуться в состояние ожидания.

Отметим, что для простоты приложения, скорости химических потоков были жестко задано равными 859, температура – 200 °F, а время – 10 секунд. В реальном приложении вы можете загрузить эти значения из рецепта или их может ввести оператор.

Чтобы создать это приложение, сначала создайте библиотеку с псевдонимами ввода-вывода для каждого сигнала ввода-вывода.

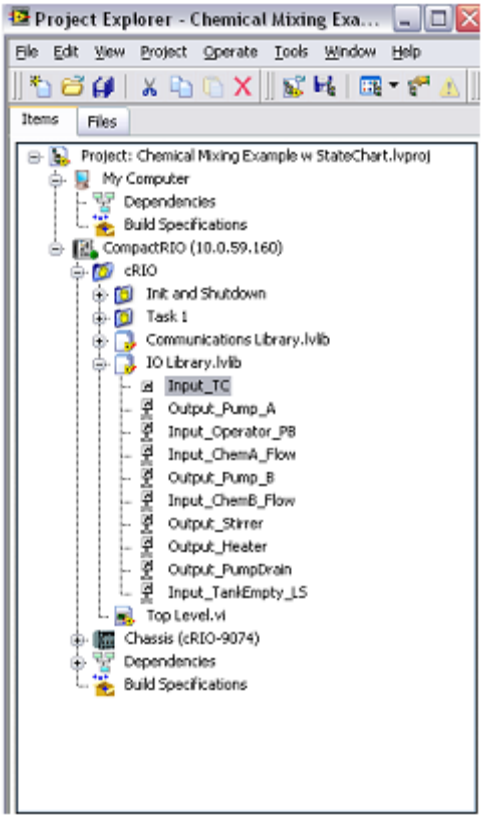


Рисунок 2.29. Создание библиотеки с псевдонимами ввода-вывода для каждого сигнала ввода-вывода

Затем создайте задачу выключения, чтобы установить выходные значения по умолчанию для выходных псевдонимов ввода-вывода.

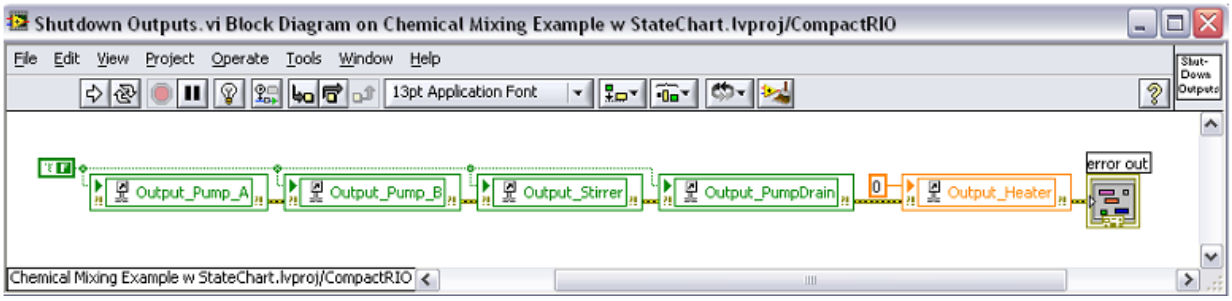


Рисунок 2.30. Создание задачи выключения для установки выходных значений по умолчанию для выходных псевдонимов ввода-вывода

Процедура разработки приложения, основанного на диаграмме состояний, включает следующие шаги:

1. Разработка VI Caller
2. Определение входов, выходов, триггеров
3. Разработка диаграммы состояний
4. Помещение диаграммы состояний в VI Caller

Шаг 1. Разработка VI Caller

VI верхнего уровня в этом приложении называется Caller VI. Он содержит синхронизируемый цикл (timed loop), который непрерывно вызывает вашу диаграмму состояний. Кроме того, VI верхнего уровня содержит секции кода для запуска и выключения, инкапсулированные в структуру последовательности.

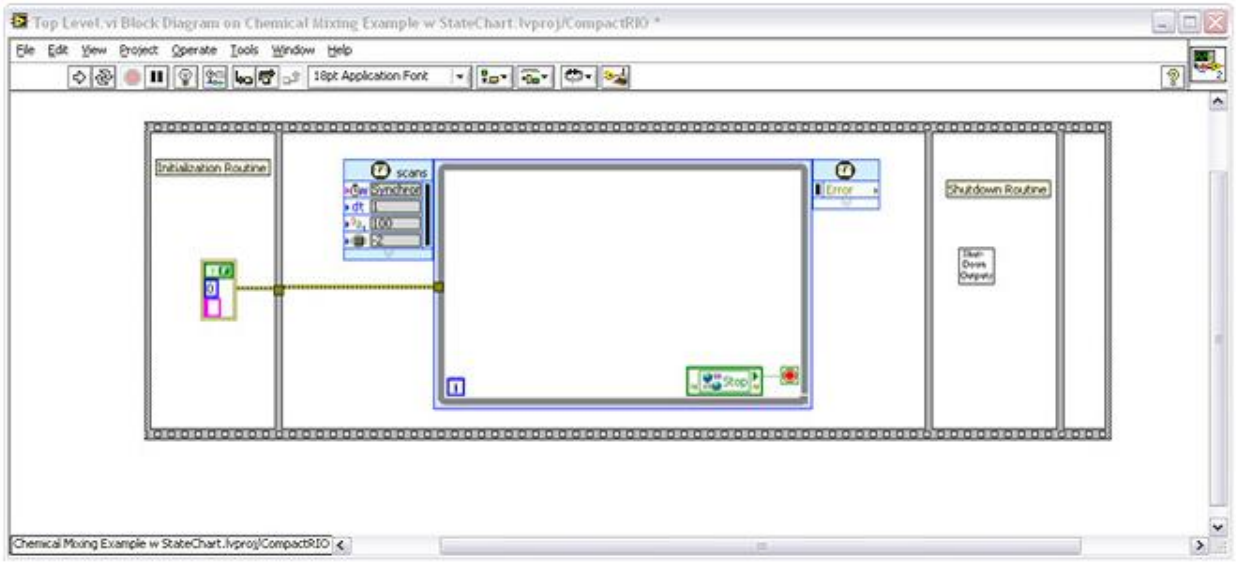


Рисунок 2.31. VI верхнего уровня содержит синхронизируемый цикл, который непрерывно вызывает диаграмму состояний. Он также включает фрагменты кода для запуска и выключения, которые инкапсулированы в структуру последовательности.

Теперь добавьте в проект LabVIEW новую диаграмму состояний. Каждая диаграмма состояний LabVIEW содержит несколько компонентов, которые вы можете использовать для конфигурирования контекста вашей разработки.

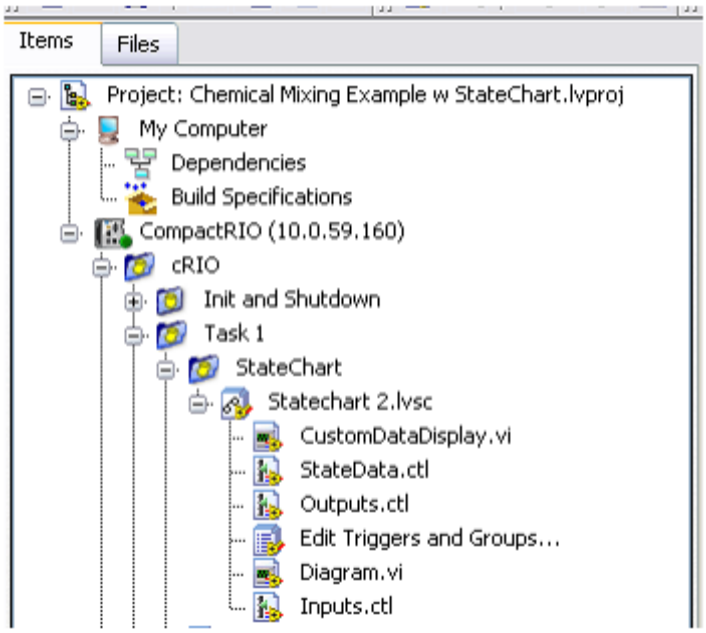


Рисунок 2.32. Добавление в проект LabVIEW новой диаграммы состояний

Файл diagram.vi содержит саму диаграмму состояний, inputs.ctl и outputs.ctl - кластеры, определяющие входы и выходы диаграммы, statedata.ctl содержит информацию о внутреннем состоянии, используемую только в диаграмме состояний. В этом примере не используются триггеры, statedata.ctl или customdatadisplay.vi.

Шаг 2. Определение входов, выходов и триггеров

Откройте, модифицируйте и сохраните inputs.ctl и outputs.ctl для создания входов и выходов, ассоциируемых с каждой точкой ввода-вывода. Кластер outputs.ctl содержит кластер ошибок в событии, которое ваша диаграмма состояний определяет как условие ошибки.

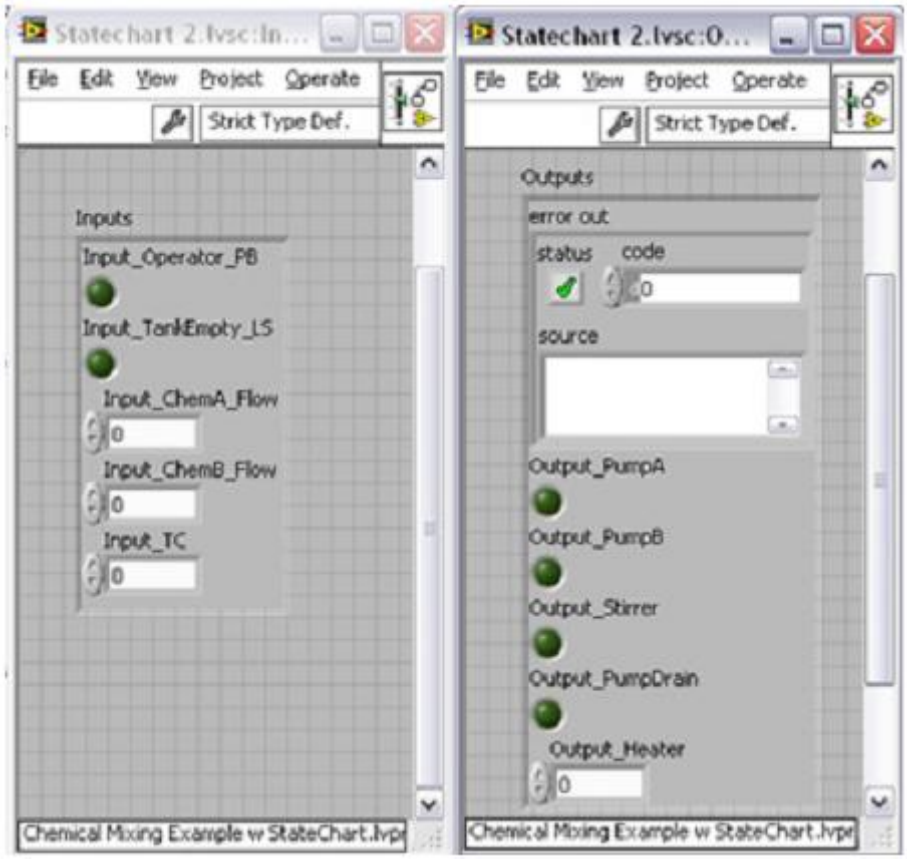


Рисунок 2.33. Открытие, модификация и сохранение inputs.ctl и outputs.ctl для создания входов и выходов, ассоциируемых с каждой точкой ввода-вывода

Шаг 3. Разработка диаграммы состояний

Теперь откройте файл diagram.vi. В этой диаграмме вы можете создавать состояния и переходы между ними. Создайте требуемые состояния, регионы и переходы для выполнения вашей логики. Каждое состояние и переход содержат код LabVIEW, исполняемый в активном состоянии. Диаграмма состояний – асинхронная диаграмма состояний, использующая сторожей для определения того, когда совершать переход между состояниями. Одним из главных преимуществ диаграмм состояний является визуальное представление поведения системы, и, следовательно, самодокументирование программного обеспечения.

Шаг 4. Помещение диаграммы состояний в Caller VI

Как только вы завершите создание диаграммы, щелкните по иконке вверху слева, чтобы LabVIEW сгенерировал код для диаграммы состояний.

Рисунок 2.34. Щелкните по иконке в верхнем левом углу, чтобы LabVIEW сгенерировал код для диаграммы состояний

В вашем главном приложении перетащите диаграмму состояний в логический фрагмент кода. Поскольку диаграмма состояний требует передачу входов и выходов через кластеры, вам придется создать subVI для чтения псевдонимов ввода-вывода и передачи их в и из кластеров диаграммы состояний. Ваш subVI для выходов должен проверять условия ошибок, прежде чем записывать в

переменные. Если произойдет ошибка, ошибка распространяется через subVI без записи в переменную.

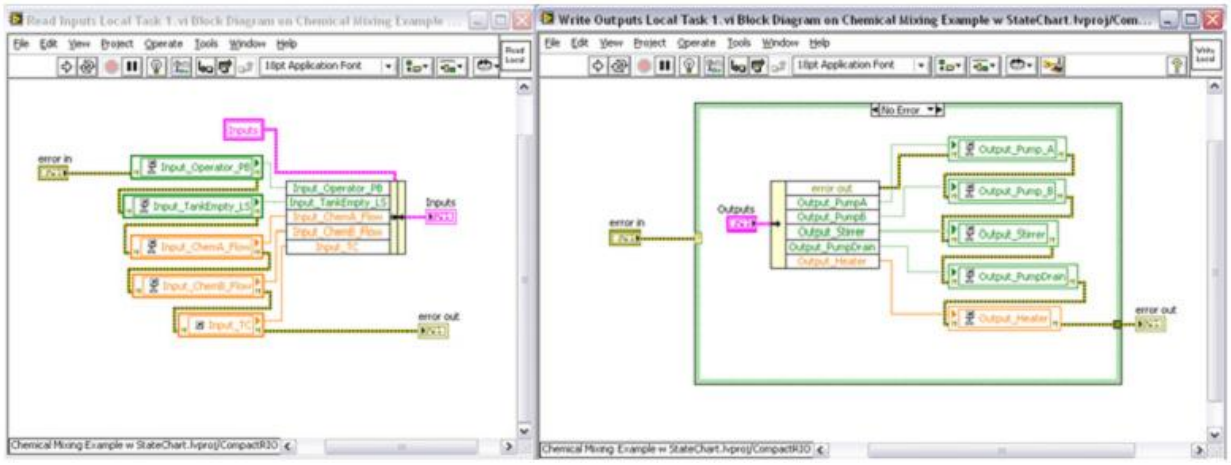


Рисунок 2.35. Создание subVI для чтения псевдонимов ввода-вывода и передачи их в кластеры и из кластеров диаграммы состояний

Наконец, перетащите и подключите всё в главном VI. Диаграмма состояний помещена в структуру case, проверяющую, не произошло ли ошибки. Если ошибка произошла, диаграмма состояний не выполняется.

Это обеспечивает надежные результаты проверок на ошибки, надлежащее поведение и выполнение вашей системы управления. Отладка диаграммы состояний запускается щелчком правой кнопки мыши из окна Свойств (properties). Запустив отладку, вы можете визуально отлаживать диаграмму состояний при помощи подсветки выполнения в LabVIEW и стандартных инструментов отладки наподобие точек останова, пробников (variable watch windows) и пошагового выполнения. Убедитесь в том, что не забыли отключить режим отладки, прежде, чем развертывать систему, для улучшения ее производительности.

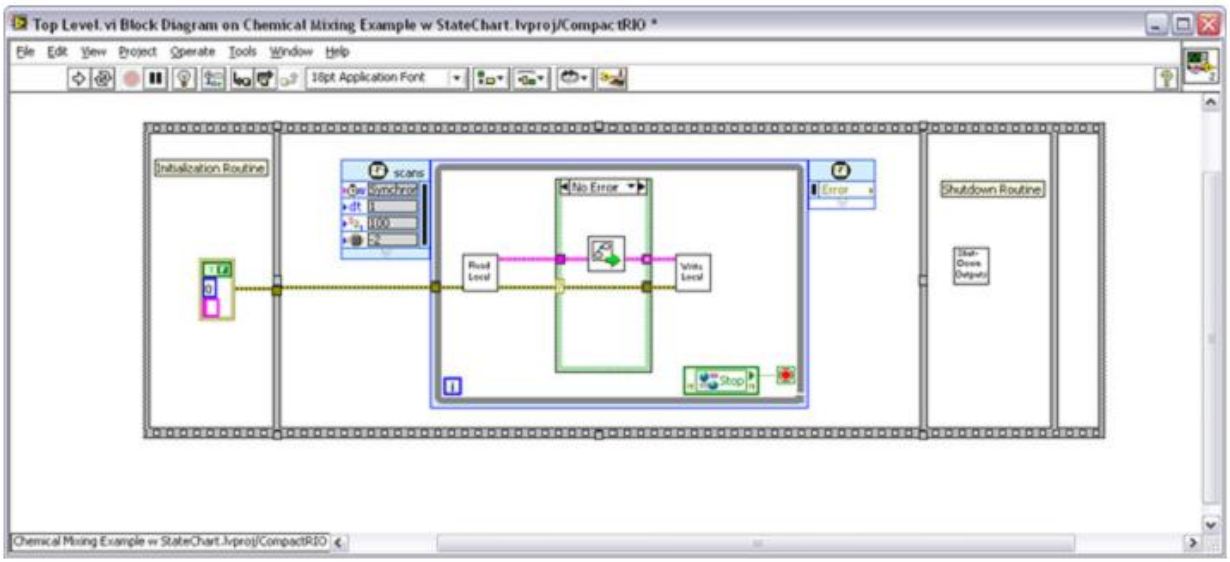


Рисунок 2.36. Если ошибок нет, диаграмма состояний выполняется

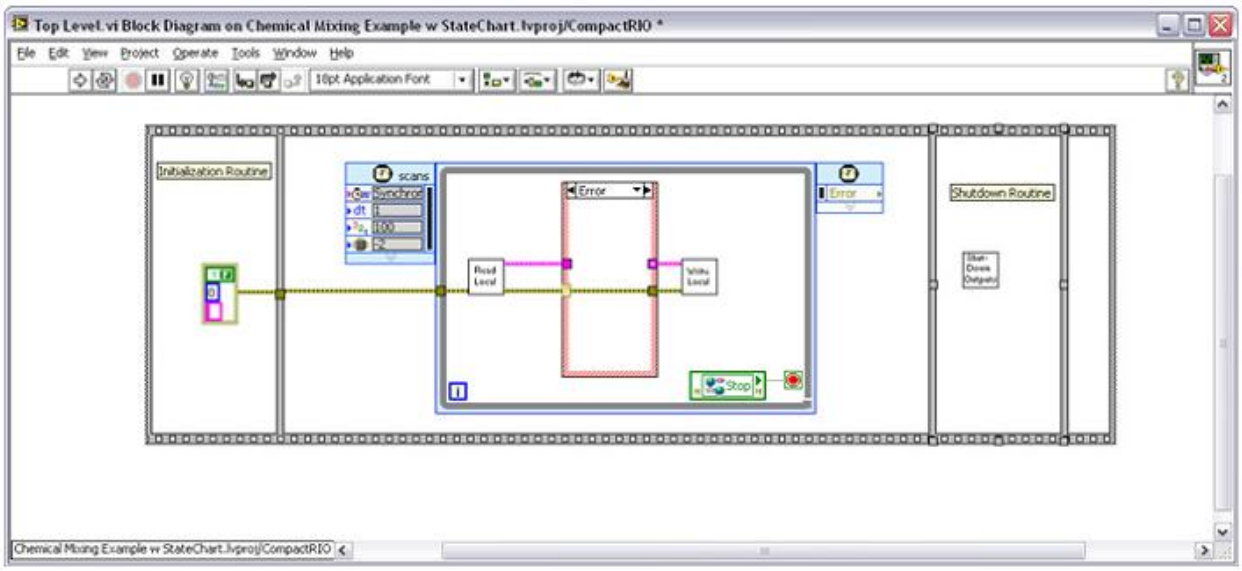


Рисунок 2.37. Если обнаружена ошибка, диаграмма состояний не выполняется

Начало работы – модифицирование примера

В данном разделе приведен пример кода LabVIEW

Самый простой способ начать работать с этим проектом – модицифировать существующий пример. В этом разделе вы модифицируете предыдущий пример с системы смешивания химических реагентов, где использовалась диаграмма состояний для построения своего собственного приложения. Процесс разработки состоит из четырех основных шагов:

1. Корректировка библиотеки ввода-вывода (IO Library) для создания псевдонимов физического ввода-вывода вашего приложения.
2. Корректировка процедуры выключения для записи в физические выходы значений, необходимых при отключении.
3. Корректировка Задачи 1 (Task 1) для чтения и записи ввода-вывода из диаграммы состояний.
4. Корректировка/перезапись диаграммы состояний под ваше приложение.

Шаг 1. Корректировка библиотеки ввода-вывода

Откройте файл Chemical Mixing.lvproj. Поскольку в вашем приложении используются другие входы и выходы, вам нужно модифицировать библиотеку ввода-вывода для создания псевдонимов физического ввода-вывода. Если вы завершили монтаж, можете подключать псевдонимы к физическому вводу-выводу прямо сейчас, если монтаж еще не закончен, вы сможете переподключить псевдонимы ввода-вывода позже. Раскройте в проекте библиотеку ввода-вывода. Вы можете редактировать переменные по одной, дважды щелкнув по псевдониму. Быстрее можно модифицировать библиотеку, используя Редактор множества переменных (Multiple Variable Editor). Вы можете открыть его, щелкнув по библиотеке правой кнопкой мыши и выбрав «Multiple Variable Editor…»

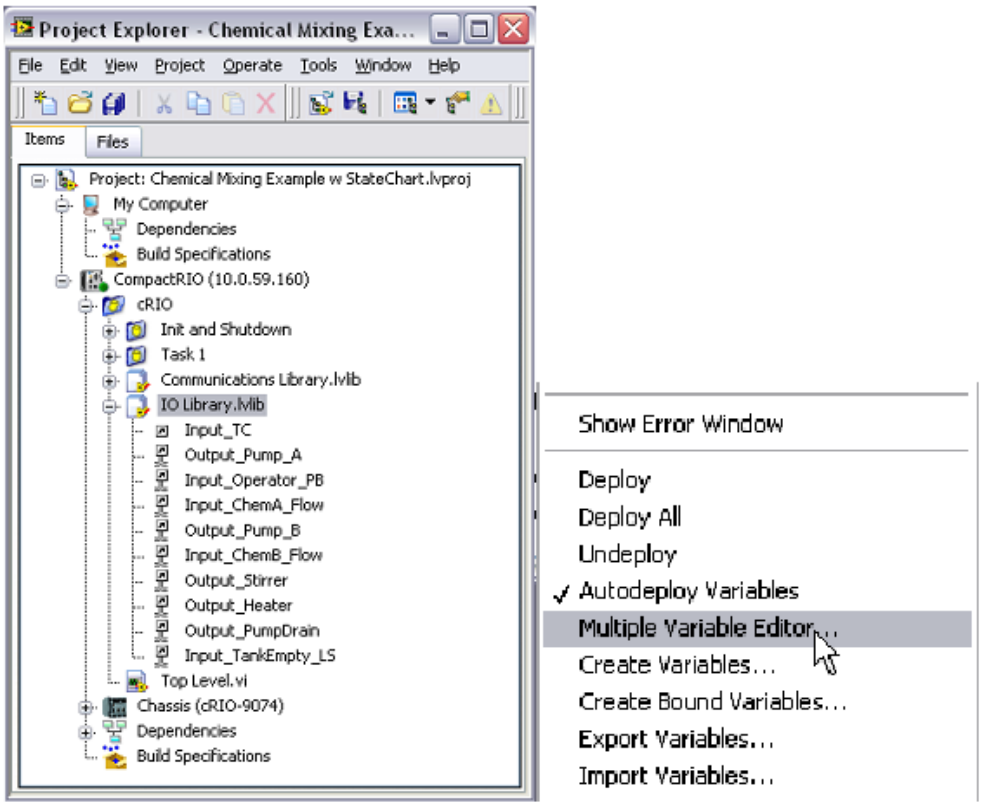


Рисунок 2.38. Вы можете быстрее редактировать переменные в редакторе Multiple Variable Editor

1. В редакторе Multiple Variable Editor измените имена, типы данных и физическую привязку (путь псевдонимов) переменных. Вы также можете быстро создавать новые переменные, копируя и вставляя существующие.

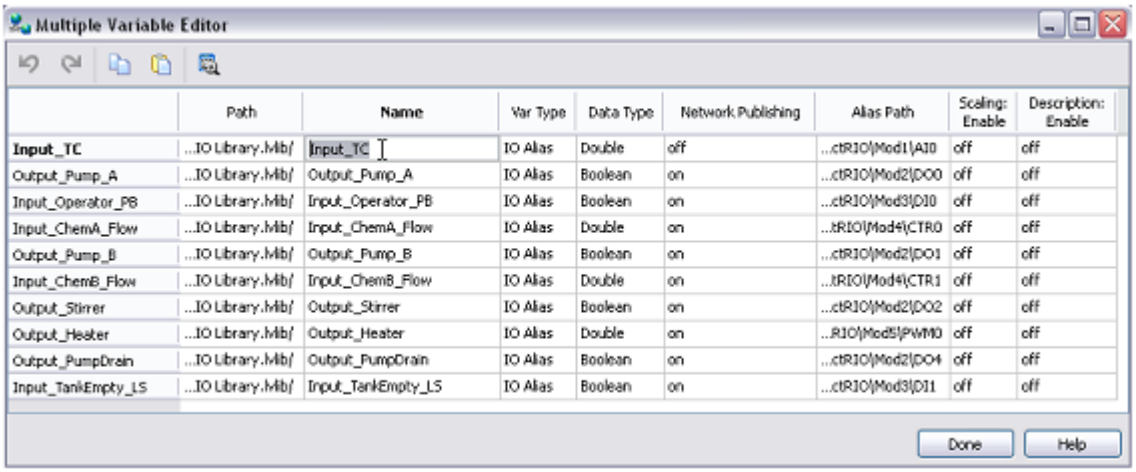


Рисунок 2.39. Параметры редактора Multiple Variable Editor

Шаг 2. Корректировка процедуры выключения

Поскольку в вашем приложении используются другие входы и выходы, вам необходимо изменить процедуру выключения, чтобы задать для ваших выходов требуемые значения по выключению.

- Откройте файл Shutdown Outputs.vi и измените его, установив выходные значения по умолчанию для созданных вами псевдонимов.

Шаг 3. Корректировка Задачи 1 для распределения ввода-вывода

Теперь вам необходимо модифицировать вашу логику. Поскольку каждая логическая задача создает локальную копию ввода-вывода для своего выполнения, вам нужно перераспределить ввод-вывод. В папке диаграммы состояний откройте файлы outputs.ctl и inputs.ctl. Измените их так, чтобы они соответствовали вводу-выводу в вашем приложении.

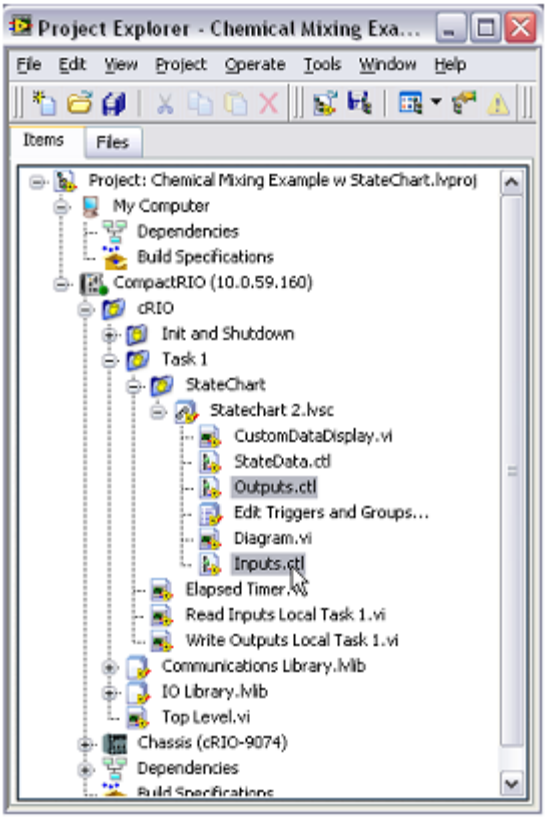


Рисунок 2.40. Откройте файлы outputs.ctl и inputs.ctl и измените их так, чтобы они соответствовали вводу-выводу в вашем приложении.

- Обновите файлы Write Outputs Local Task 1.vi и Read Outputs Local Task 1.vi для чтения и записи вашего ввода-вывода.

Шаг 4. Корректировка/перезапись диаграммы состояний

Теперь необходимо добавить вашу логику. Вы можете сделать это, изменив/переписав диаграмму состояний для выполнения вашего приложения.

РАЗДЕЛ 3
Техника программирования масштабируемых систем

Повторно используемые функции

Разработка кода управления механизмами может считаться идеальной, если фрагменты кода пригодны к повторному использованию. Это экономит время, поскольку вы можете в пределах проекта использовать модульное программирование и создавать библиотеку кода для последующих проектов. В других средах разработки эти повторно используемые фрагменты кода называются функциями или функциональными блоками.

Для возможности многократного использования код должен удовлетворять трем главным требованиям:

1. Должен существовать метод для вызова кода и предоставления входных и выходных данных.
2. Код должен поддерживать собственную область памяти для сохранения собственного состояния (это может быть необязательно для некоторых функций).
3. Код должен обладать возможностью вызываться неоднократно в одной программе.

Создание многократно используемого кода в LabVIEW

В LabVIEW повторно используемые фрагменты кода называются subVI. LabVIEW – иерархический язык программирования, разработанный, чтобы облегчить многократное использование кода при помощи реентерабельных subVI. Для создания трех компонентов повторно используемого кода:

1. Должен существовать метод для вызова кода и предоставления входных и выходных данных.

- В LabVIEW вы можете этого добиться путем создания входов и выходов на лицевой панели и подключения этих элементов управления и индикаторов к коннектору.

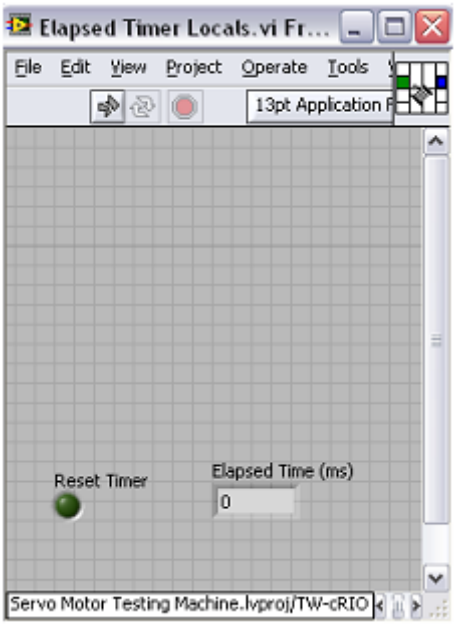


Рисунок 3.1. Входы и выходы панели подключения соедините с элементами управления и индикации на лицевой панели

2. Код должен поддерживать собственную область памяти для сохранения собственного состояния (это может быть необязательно для некоторых функций).

- В LabVIEW вы можете добиться этого двумя способами. Вы можете использовать цикл while с неинициализированными сдвиговыми регистрами для реализации памяти, в которой будут храниться состояния, или же можете создать локальные переменные. Локальные переменные - несколько более затратный, но и более гибкий и простой для понимания способ. Вы можете создавать локальные переменные для элементов управления и индикаторов лицевой панели. Щелкните правой кнопкой мыши по элементу лицевой панели и создайте локальную переменную. На одной блок-диаграмме к этой переменной можно обращаться неоднократно.

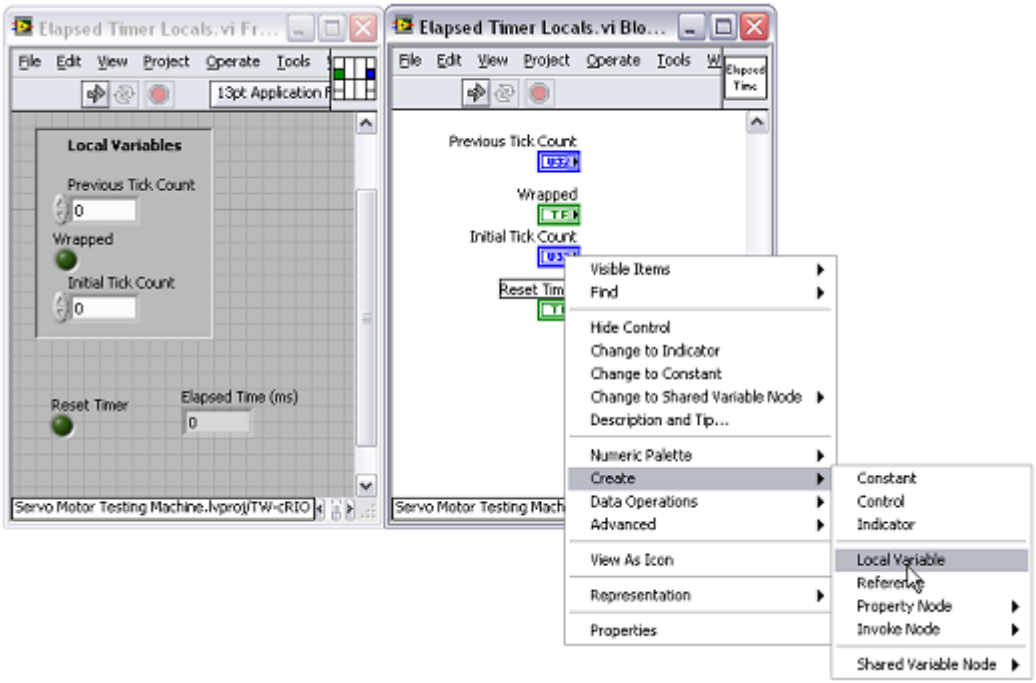


Рисунок 3.2. Щелкните правой кнопкой мыши по элементу управления или индикатору, чтобы создать локальные переменные

3. Код должен обладать возможностью вызываться неоднократно в одной программе

- В LabVIEW это осуществляется путем создания реентерабельного VI. Реентерабельный VI имеет отдельную область памяти для каждого случая его вызова в программе. Чтобы сделать subVI реентерабельным, зайдите в окно свойств (Properties) в меню Файл, выберите Execution (Выполнение) из выпадающего меню Category (Категория) и поставьте флажок Reentrant execution (Реентерабельное выполнение).

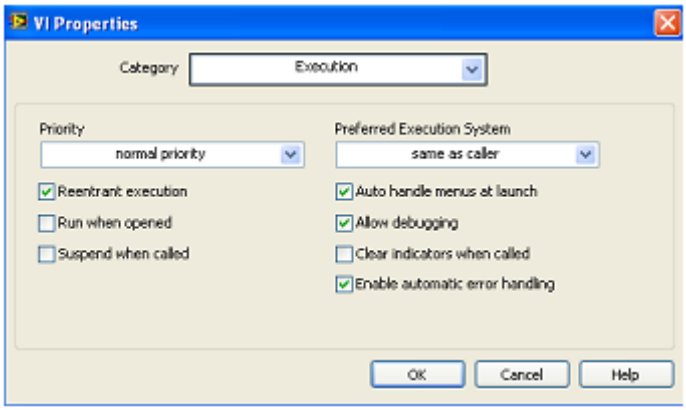


Рисунок 3.3. Создание реентерабельного ВП

Пример разработки повторно используемого кода в LabVIEW

В данном разделе приведен пример кода LabVIEW

Если вы посмотрите на предыдущий пример системы смешивания химических реагентов, одним из требований было удерживать заданную температуру смеси в течение заданного времени. Поскольку управляющее приложение должно по-прежнему реагировать на изменения входов, вы не можете использовать команду «wait» для управления временем выполнения приложения. Если вы это сделаете, остальная часть управляющего алгоритма, находящаяся внутри цикла, не выполнится, пока не завершится ожидание, и ваше приложение перестанет реагировать на команды или изменения входов. Поскольку вы не можете поместить задержку в цикл, вам необходим метод, позволяющий на каждой итерации цикла проверять время нахождения в этом состоянии. Это общее требование к повторно используемому коду идеального приложения.

Чтобы научиться создавать повторно используемый код, создайте subVI для оценки прошедшего времени. Эта функция должна выдавать на выходе истекшее время и иметь вход для сброса таймера. В LabVIEW есть функция Tick Count, считывающая счетчик микросекунд. Функция Tick Count выдает на выход количество микросекунд в формате U32 . Ниже приведена логика работы subVI, рассчитывающего прошедшее время:

- Проверьте, в первый ли раз запущена эта копия VI, или находится вход сброса счетчика в состоянии "истина". Если это так, прочитайте значение Tick Count и сохраните его как начальное значение счетчика, установите выход прошедшего времени равным 0 и запишите "ложь" в регистр переполнения.
- Проверьте, произошло ли переполнение на выходе Tick Count (если значение на выходе превысит допустимый для формата U32 размер, счет начнется заново с 0), сравнив с последним значением на выходе Tick Count. Если переполнение было, установите регистр переполнения (wrapped register) в состояние "истина".
- Вычтите текущее значение Tick Count из исходного значения Tick Count. Если произошло переполнение, преобразуйте значение в формат U64, прибавьте 2^32 -1, вычтите исходное значение Tick count и преобразуйте обратно в формат U32.

4. Должен существовать метод для вызова кода и предоставления входных и выходных данных.

- Создайте новый VI. На передней панели создайте управляющий элемент для сброса («reset») и индикатор для прошедшего времени («elapsed time»). Подключите их к коннектору VI.

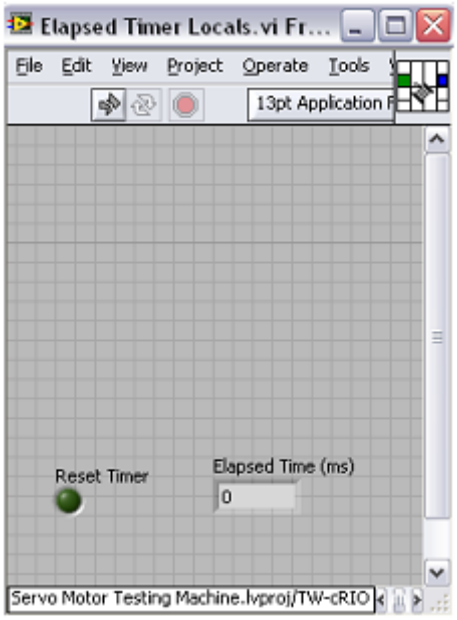


Рисунок 3.4. Создайте новый VI для вызова кода и предоставления входных и выходных данных

2. Код должен поддерживать собственную область памяти

- На лицевой панели создайте управляющие элементы и индикаторы для трех локальных переменных: Previous Tick Count (Предыдущее значение счетчика), Wrapped (Переполнение), и Initial Tick Count (Начальное значение счетчика).

3. Код должен обладать возможностью вызываться неоднократно в одной программе

- Обратитесь в окно свойств (Properties) и сделайте VI реентерабельным.

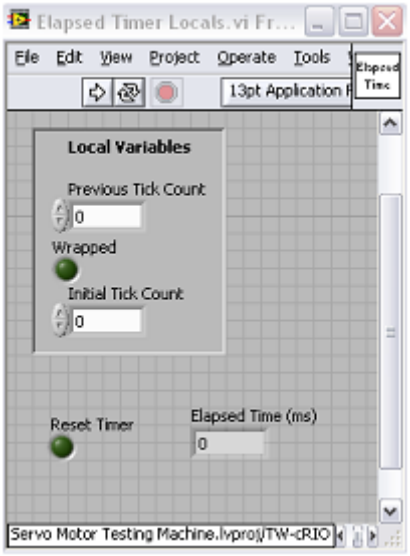


Рисунок 3.5. Создайте управляющие элементы и индикаторы для трех локальных переменных

Теперь, просто добавьте на блок-диаграмму логику для таймера истекшего времени. Когда вам потребуется обратиться к локальным данным, щелкните правой кнопкой мыши по элементу

управления или индикатору и создайте локальную переменную. Далее вы можете отладить этот код и повторно использовать его в различных управляющих программах.

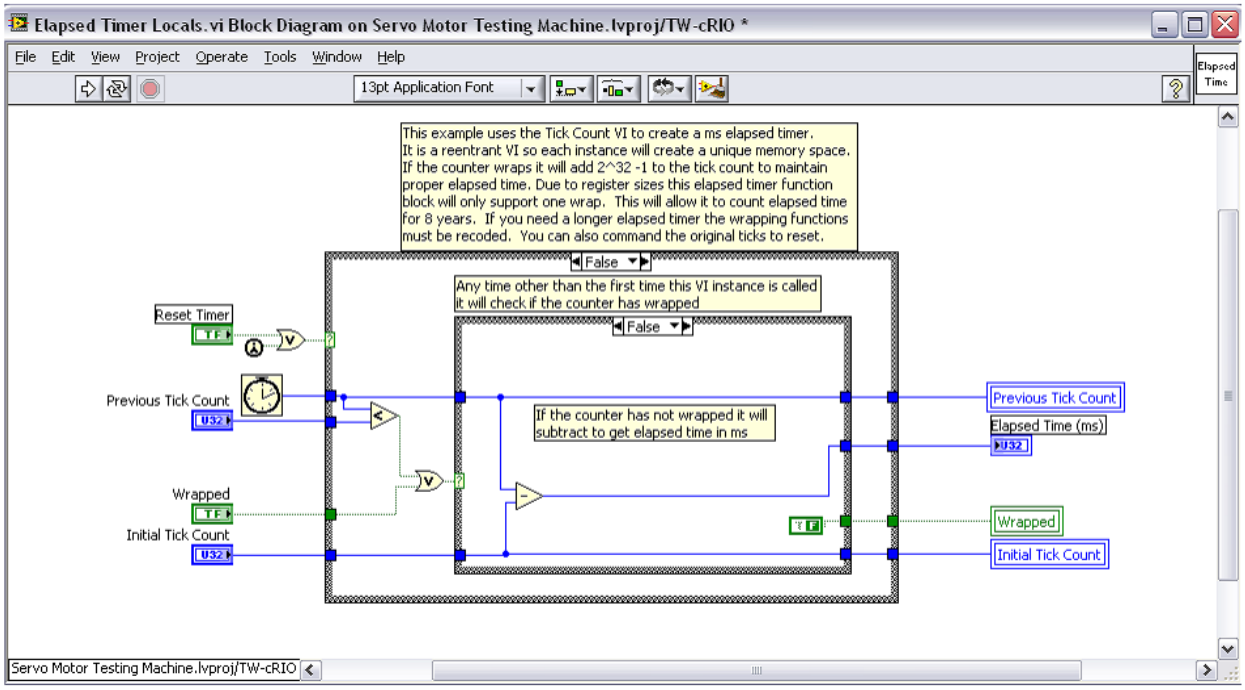


Рисунок 3.6. Код готов

Другие повторно используемые коды в LabVIEW

NI поставляет LabVIEW с обширной библиотекой кода многократного использования, к которой вы можете обратиться из палитры функций. Этот код предоставляет сотни встраиваемых функций управления, анализа, коммуникаций, файлового ввода-вывода и т.д.

Функциональные блоки IEC 61131

В LabVIEW 8.6 появился новый тип кода многократного использования, называемый функциональными блоками. Эти функциональные блоки основаны на международном стандарте IEC 61131-3 программирования промышленных систем управления. Они написаны в LabVIEW, разработаны для использования в приложениях реального времени и обладают способностью публиковать свои параметры в табличной памяти (как переменные общего доступа). Вы можете использовать эти функциональные блоки наряду с прочими кодами LabVIEW.

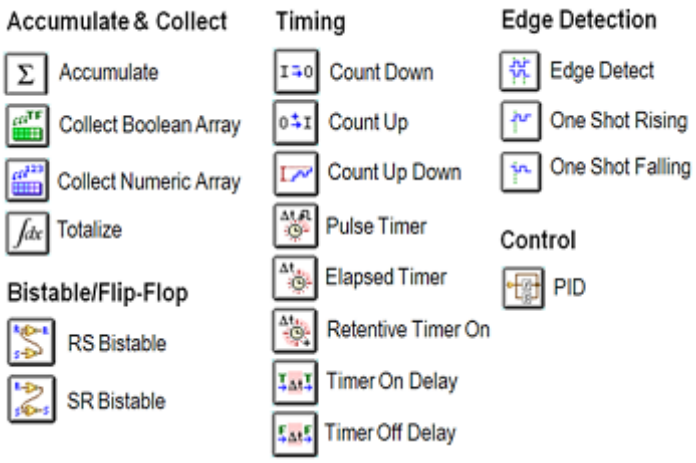


Рисунок 3.7. Новые функциональные блоки LabVIEW, основанные на международном стандарте IEC 61131-3 программирования промышленных систем управления.

Конфигурируемые переменные терминалов

Функциональные блоки отличаются от стандартных subVI тем, что имеют страницу конфигурирования и предоставляют возможность напрямую соединять входы и выходы с записями в глобальной таблице памяти, видимой в проекте LabVIEW. Вы можете также получить доступ к этим записям по сети. Вы можете настраивать терминалы и переменные из окна свойств функционального блока.

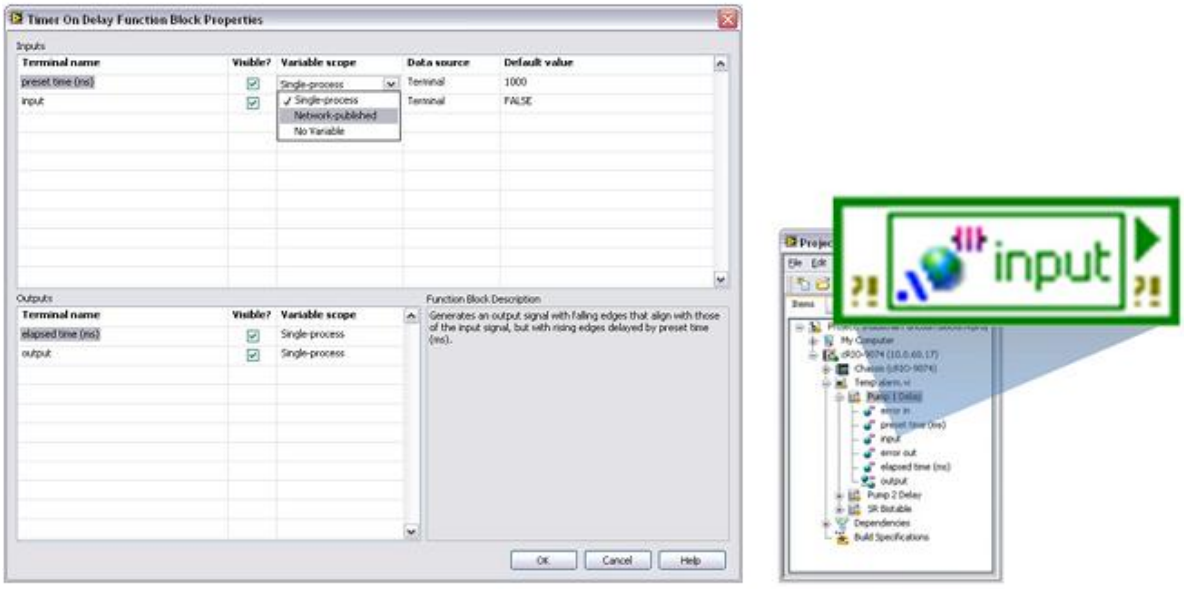


Рисунок 3.8. Конфигурируйте функциональные блоки в окне свойств и обращайтесь к входам и выходам из таблицы памяти

Многозадачность (Множественные циклы)

Во многих приложениях контроллер выполняет более чем одну задачу управления и измерения. Например, приложение управления механизмом может содержать задачу управления работой механизма с использованием диаграммы состояний и вторую задачу, осуществляющую наблюдение за состоянием механизма или задачу регистрации данных.

Управляющая программа может содержать несколько задач, которые выполняются параллельно и передают данные в таблицу памяти.

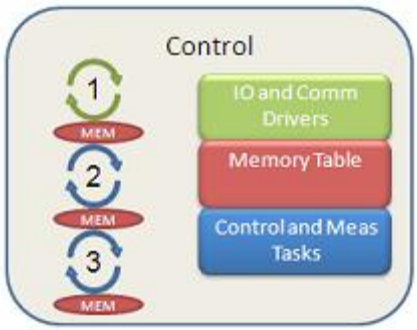


Рисунок 3.9. Управляющая программа может содержать несколько задач, которые выполняются параллельно и обмениваются данными с табличной памятью

Чтобы управлять выполнением вашего приложения, вы должны обладать возможностью:

- Устанавливать приоритет задач

- Синхронизировать задачи
- Передавать данные между задачами
- Запускать задачи

Установление приоритета и синхронизация задач

При выполнении нескольких задач вы должны убедиться, что ваша задача управления имеет наивысший приоритет. Поскольку выполнение LabVIEW происходит во времени, цикл с высоким приоритетом, такой, как цикл сканирования ввода-вывода, всегда работает строго по графику. Это обеспечивает низкий джиттер и стабильное управление. Однако, это также означает, что если контроллер не завершит все требуемые операции вовремя, эти операции прервутся. Это может быть нормально (или даже желательно) для заданий с низким уровнем приоритета, вроде регистрации данных или сетевого взаимодействия. Но если прервется управляющая задача, это может привести к нестабильной работе. Поэтому вы должны разработать свое приложение так, чтобы можно было определять приоритет заданий. Вы должны также использовать инструменты, подобные NI Real-Time Execution Trace Toolkit, для оценки эффективности вашего приложения, чтобы гарантировать адекватность времени, отводимого на фоновые задания, такие, как, например, коммуникации.

Для установления приоритета заданий вы можете использовать синхронизируемый цикл (timed loop). Синхронизируемый цикл имеет настраиваемый приоритет относительно других синхронизируемых циклов. Чем большее число вы введете, тем выше приоритет. Значение приоритета должно быть положительным целым числом от 1 до 65535. Исполняющая система LabVIEW - с приоритетным прерыванием, так что готовая к выполнению синхронизируемая структура с высоким приоритетом задерживает выполнение низкоприоритетных структур, также готовых к выполнению, и прочего кода LabVIEW, выполняемого не в режиме жестких временных ограничений.

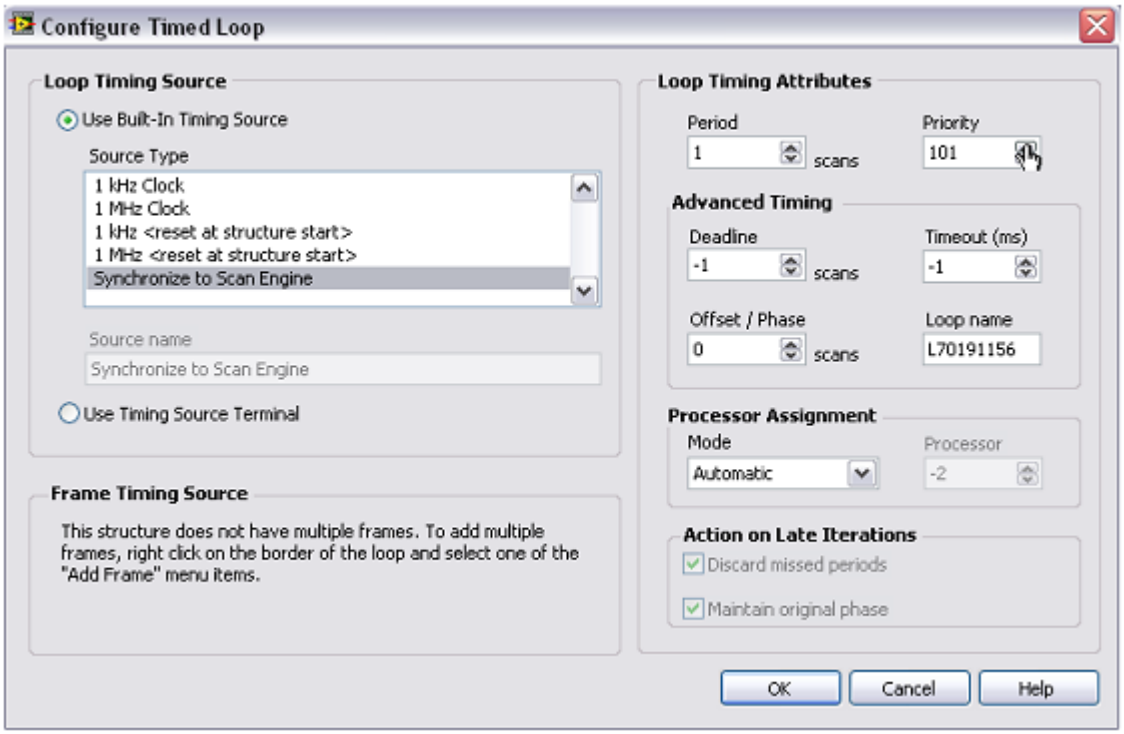


Рисунок 3.10. Установка приоритета синхронизируемого цикла

Чтобы синхронизировать несколько задач, настройте их на синхронизацию NI Scan Engine. Все циклы, синхронизированные с NI Scan Engine, запускаются с частотой сканирования ввода-вывода и выполняются по мере уменьшения приоритета. Если у вас есть фоновые или недетерминированные задачи, не требующие синхронизации или определенного приоритета, вы

можете выполнять эти задачи в стандартном цикле while с функцией задержки (wait) для управления временем выполнения цикла.

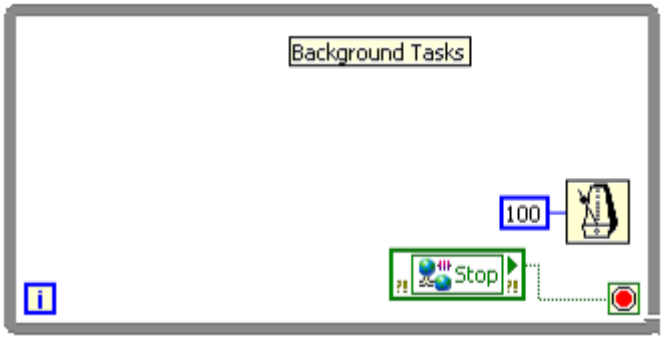


Рисунок 3.11. Циклы while работают с нормальным приоритетом

Обмен данными между задачами

Все задачи могут читать и записывать данные ввода-вывода из табличной памяти. Для передачи данных между задачами, вы должны добавить новый набор данных в табличную память, что требует использования компонента под названием переменные общего доступа LabVIEW (Shared Variable). При помощи переменных общего доступа LabVIEW вы можете передавать данные в пределах контроллера или внутри сети. Эти переменные конфигурируются, и вы можете их использовать для обеспечения требуемой функциональности в пределах контроллера или внутри сети. Пока что сосредоточимся на использовании переменных общего доступа для занесения данных в табличную память контроллера.

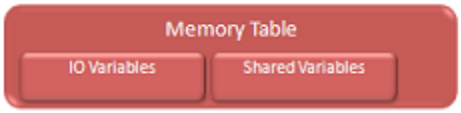


Рисунок 3.12. Переменные общего доступа и переменные ввода-вывода – элементы табличной памяти

Создание новой переменной общего доступа напоминает создание псевдонима ввода-вывода. Вы можете организовать эти переменные с использованием библиотек. Вначале создайте новую библиотеку в проекте, а затем создайте в ней новую переменную. Выберете тип данных и задайте тип переменной Single Process (переменная общего доступа для одного процесса – это глобальная переменная).

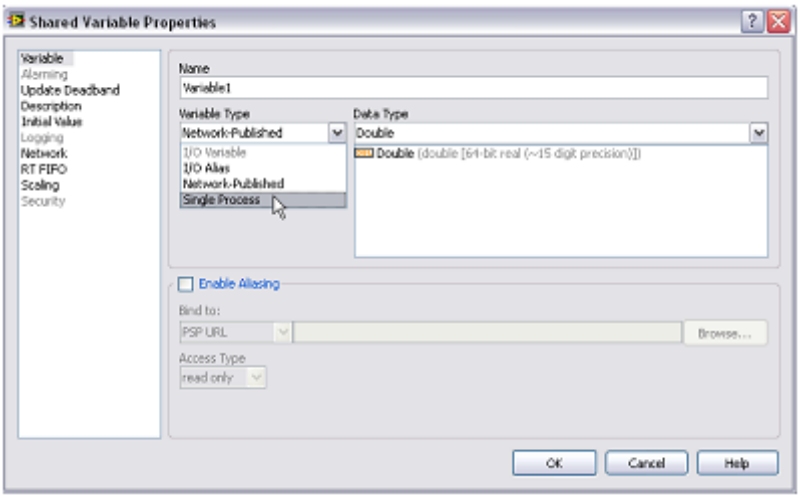


Рисунок 3.13. Создание переменной общего доступа

Далее перейдите на закладку RT FIFO. Некоторые переменные, например, массивы, не могут быть прочитаны или записаны в операции одного процесса. Когда циклы прерываются циклами с более

высоким приоритетом, эти незаконченные операции могут увеличивать нагрузку на процессор и создавать джиттер. Чтобы этого избежать, включите буфер FIFO реального времени (real-time FIFO) и установите его свойство Single Element (Один элемент).

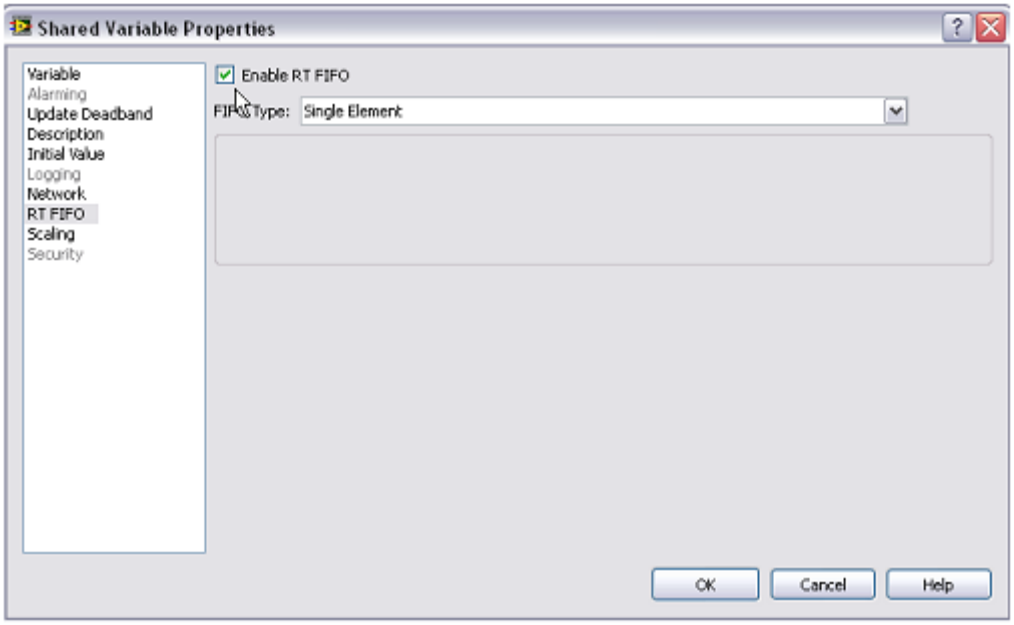


Рисунок 3.14. При включении real-time FIFO вы можете читать и писать в переменные общего доступа в нескольких параллельных циклах, джиттер при этом не возникает

Теперь вы можете читать и записывать в этот элемент табличной памяти из любого места вашего кода, так же, как и в псевдонимы ввода-вывода.

Запуск задач

Иногда необходимо запускать задачу из другой задачи. Например, код верхнего уровня, управляющий механизмом, в одном из состояний должен запустить сбор сигналов и операцию обработки данных. Эти операции с сигналами должны выполняться в параллельной задаче с низким приоритетом. Чтобы запустить параллельные задачи, используйте архитектуру, основанную на командах.

Архитектура, основанная на командах, в системах с параллельными циклами

На самом простом уровне архитектура, основанная на командах, может быть описана как модель с двумя сущностями, Начальник и Исполнитель, которые связаны сообщением - командой. Начальник командой передает Исполнителю, какое действие необходимо выполнить, и Исполнитель его выполняет.

Рисунок 3.15. Архитектура, основанная на командах, предоставляет механизм передачи команд между параллельными задачами
Commander - Начальник, Worker - Исполнитель, Command - команда

Для подобной модели используются и другие термины, в том числе главный-подчиненный, сервер-клиент и поставщик-потребитель, с незначительными вариациями смысла и реализации.

Существует множество методов для передачи команды между параллельными циклами в

LabVIEW, включая буферы реального времени FIFO, очереди или переменные общего доступа.

Рисунок 3.16. Вы можете передать команду в LabVIEW различными способами, включая буферы реального времени, очереди или переменные общего доступа
Commander - Начальник, Worker - Исполнитель, Command – команда, Queue -очередь , RT-FIFO – буфер FIFO реального времени, Variable – переменная общего доступа.

Переменные общего доступа для команд

Переменные общего доступа предлагают масштабируемый механизм передачи команд между параллельными циклами для операций, подобных запуску. Переменная общего доступа должна обладать свойством надежности в реальном времени (real-time safe) и предоставлять механизм для буферизации команд в буфере FIFO.

Чтобы это сделать, создайте переменную общего доступа типа Single-Process

и на закладке RT FIFO включите RT FIFO и установите его тип multi-element (Многоэлементный).

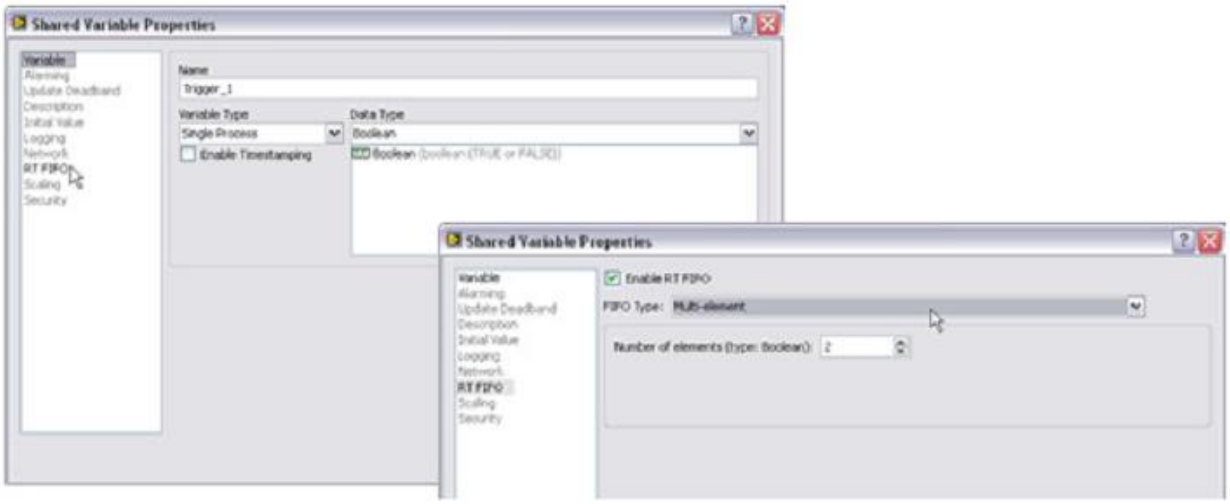


Рисунок 3.17. Переменная общего доступа Single-Process с буфером RT FIFO, установленным в режим буферизации нескольких элементов - эффективный способ передачи команд между циклами

Многоэлементный буфер FIFO представляет собой буфер, в который при каждой записи добавляется один элемент, а при каждом чтении один элемент удаляется. Если цикл Исполнителя регулярно проверяет FIFO, по умолчанию вы можете оставить размер буфера в 2 элемента и не пропустить любой триггер или команду.

Чтобы проверить состояние буфера, FIFO, задача- Исполнитель должна считать значение переменной общего доступа и проверить статус ошибки. Если буфер FIFO пуст, переменная общего доступа возвращает предупреждение -2220. Если это предупреждение не появляется, тогда буфер FIFO был не пуст, и возвращаемое значение действительно является командой.

Каждый раз, когда считывается переменная общего доступа, из буфера FIFO удаляется один элемент (при условии, что буфер не пуст). Из-за этого нельзя, чтобы несколько Исполнителей получали команду из одного и того же буфера FIFO, но нескольким Начальникам можно помещать команды в буфер FIFO.

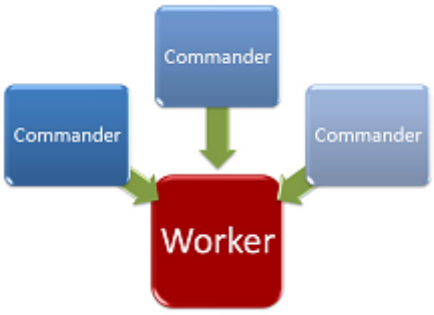


Рисунок 3.18. У вас может быть несколько записывающих команду, но только один читающий
Commander - Начальник, Worker – Исполнитель

Числовые команды

Команды могут быть более сложными, чем простой триггер (запуск). Переменные общего доступа поддерживают различные типы данных и, при выборе численного типа, могут быть использованы для передачи множества различных команд. Определение перечислительного типа (enum – стандартный тип данных LabVIEW) - отличный способ определения команд, которые могут быть напрямую переданы в переменную общего доступа.

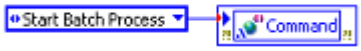


Рисунок 3.19. Подключение команды типа Enum к числовой переменной общего доступа

Определение типа enum предоставляет упорядоченный список доступных команд. Если вы сделаете определение типа, изменения автоматически распространятся в коде.

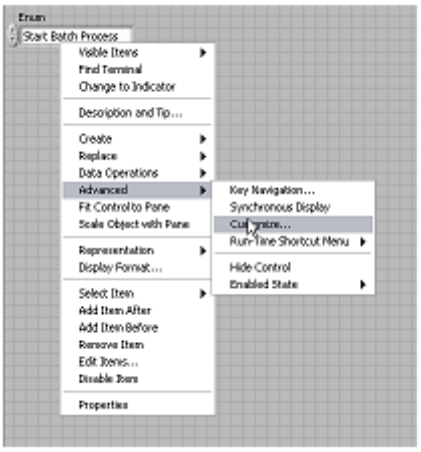


Рисунок 3.20. Если вы сделаете определение типа, изменения в списке команд автоматически распространятся на весь код

Пример использования переменных общего доступа для запуска параллельного цикла

В данном разделе приведен
пример кода LabVIEW

Теперь модифицируем код тестирования серводвигателя так, , чтобы, когда на диаграмме состояний псевдоним Output_Rotating IOV станет истинным, это вызывало запуск параллельного цикла.

В первую очередь создадим переменную общего доступа. В этом примере – это простой триггер, так что создайте переменную логического типа (Boolean) и сохраним ее в новой библиотеке Execution Control Library.

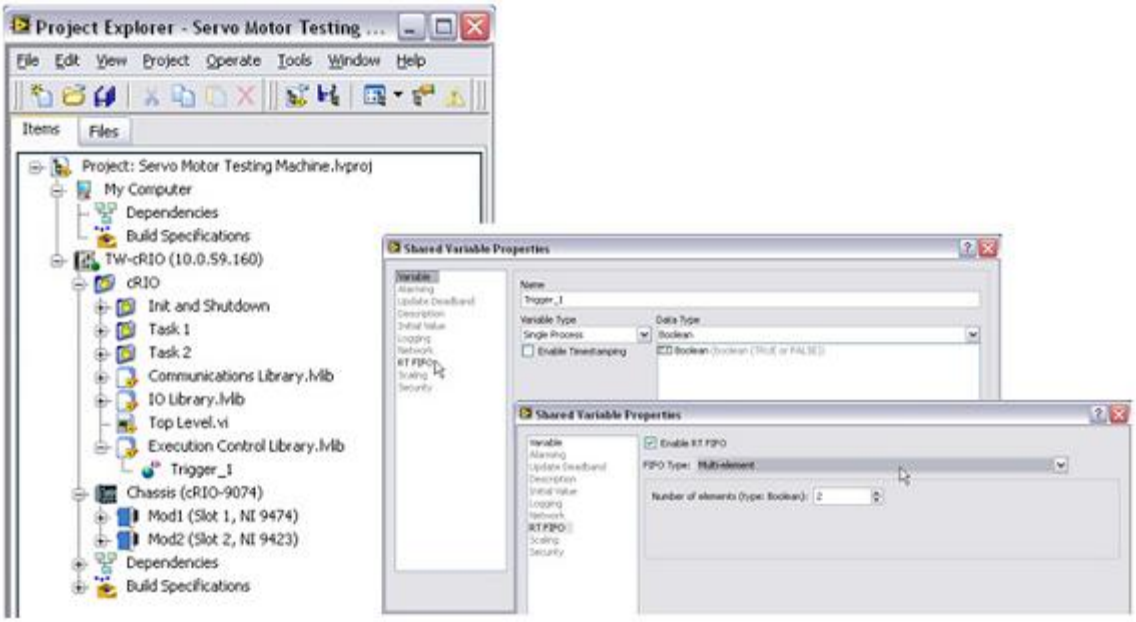


Рисунок 3.21. Вы можете использовать новую библиотеку как контейнер всех команд для организации кода

Далее изменим файл Write Outputs Local Task 1.vi, чтобы записать истинное значение в переменную Trigger_1, когда значение Output_Rotating изменится с ложного на истинное.

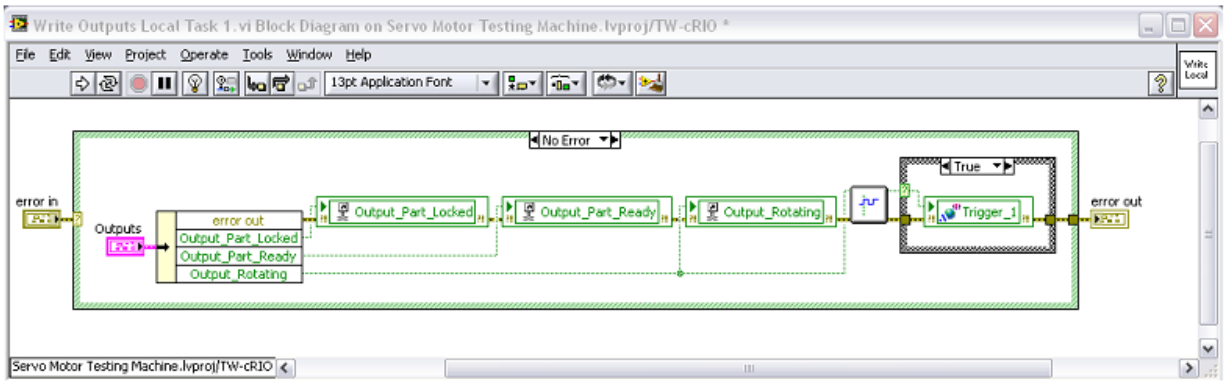


Рисунок 3.22. В отличие от нормального ввода-вывода, запись значений в переменные общего доступа используется для команд, только если вы хотите запустить параллельный цикл

Наконец, создадим параллельный цикл Исполнителя с нормальным приоритетом, в котором будет считываться переменная Trigger_1 и будем фильтровать выход, проверяя кластер ошибок на предупреждение -2220. Поскольку вполне вероятно, что вам захочется повторно использовать этот код, создадим реентерабельный subVI, который проверяет наличие этого предупреждения и сохраним его как Empty FIFO Filter.vi.

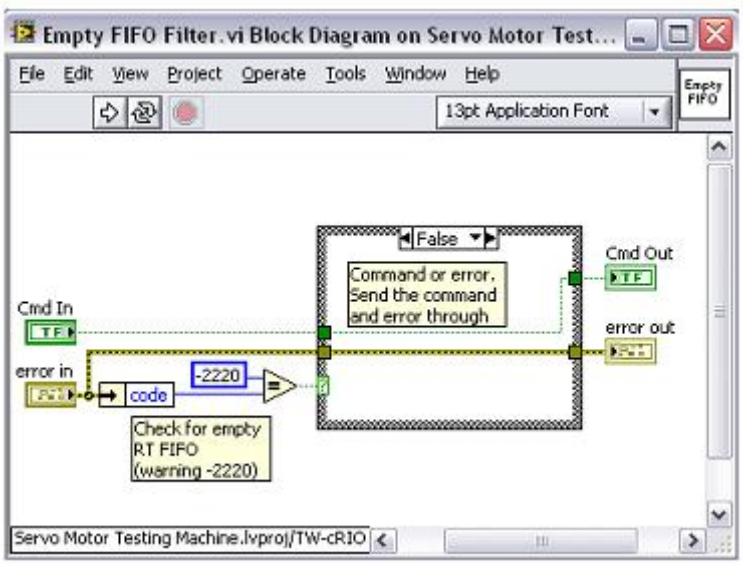


Рисунок 3.23. Повторно используемый фрагмент кода фильтрует выход, если память реального времени FIFO пуста

Параллельный цикл запускает логику, если получит истинное значение из буфера FIFO. Если он не получит истинное значение, то попробует снова считать значение переменной общего доступа после некоторого интервала ожидания.

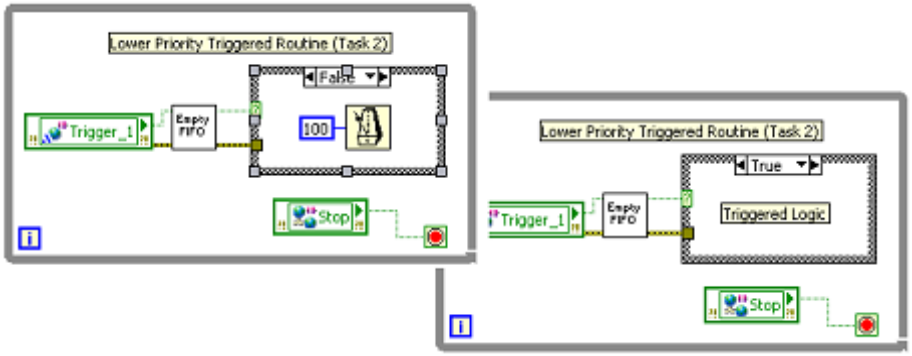


Рисунок 3.24. Ожидание во фрейме false определяет частоту опроса этого VI

Это последняя версия блок-диаграммы с основным контуром регулирования и вызываемой им задачей более низкого приоритета.

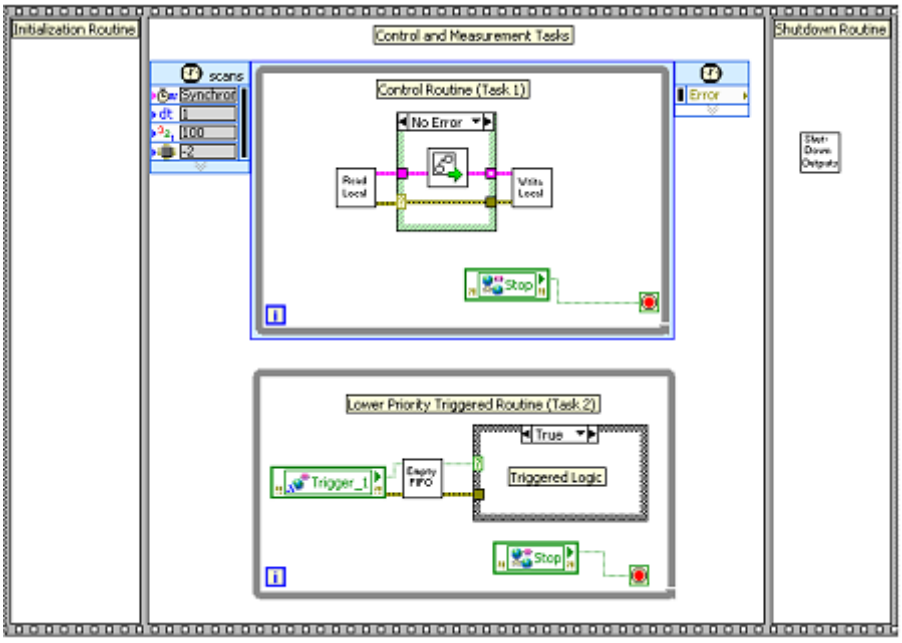


Рисунок 3.25 Приложение с циклом Начальника (задача 1) и циклом Исполнителя (задача 2)

Добавление данных к сканированию ввода-вывода

В данном разделе приведен пример кода LabVIEW

В предыдущих разделах рассматривались операции чтения и записи при вводе-выводе с помощью NI Scan Engine с псевдонимами ввода-вывода и передача данных между задачами при помощи переменных общего доступа. Во многих приложениях необходимо читать и записывать данные ввода-вывода, не обрабатываемые NI Scan Engine. Например, вам может потребоваться обратиться к данным расходомера с интерфейсом RS232 через последовательный порт контроллера CompactRIO. Еще один пример – доступ к FPGA CompactRIO при повышенных требованиях к вводу-выводу.

В этом разделе рассматривается архитектура, позволяющая добавить пользовательскую задачу сканирования ввода-вывода в ваше приложение.

Помимо ввода-вывода с помощью NI Scan Engine, вам может понадобиться получить доступ к данным из других первичных портов ввода-вывода, включая:

- FPGA
- Ethernet
- Последовательный порт
- USB

Добавление пользовательской задачи ввода-вывода (Driver Loop – цикл драйвера)

Как было сказано выше, вам нужно использовать синхронизируемый цикл для добавления задачи в приложение. Поместите дополнительный синхронизируемый цикл в кадр управления и измерения.

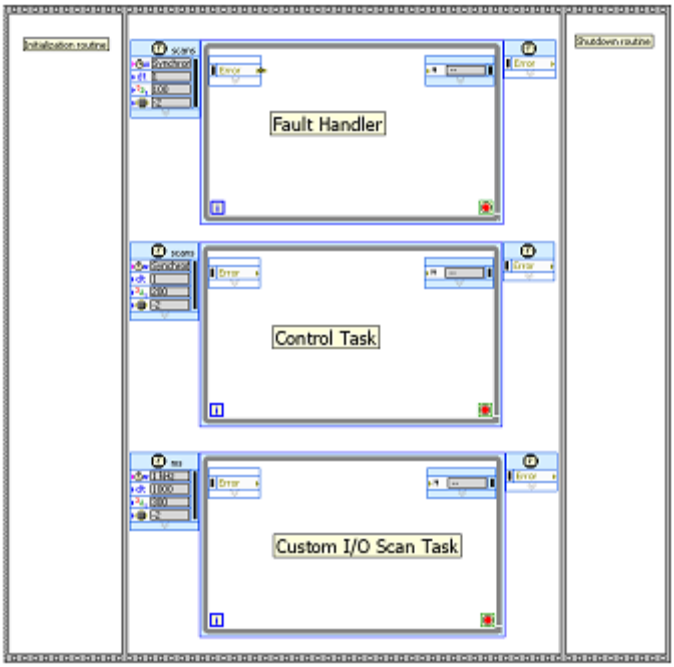


Рисунок 3.26. Добавьте еще один синхронизируемый цикл для управления пользовательской задачей ввода-вывода

Приоритет и синхронизация

Приоритет и синхронизация пользовательской задачи ввода-вывода зависит от того, к какому устройству ввода-вывода вы обращаетесь, и как вы используете данные в своем приложении. Ниже приведены два основных сценария.

Синхронный детерминированный ввод-вывод

Если драйвер ввода-вывода детерминированный, а вы используете данные в каждой итерации задачи управления, вам необходимо синхронизировать пользовательскую задачу сканирования ввода-вывода с NI Scan Engine, обладающую более высоким приоритетом, чем задача управления. Благодаря этому пользовательская задача сканирования ввода-вывода выполняется раньше каждой итерации задачи управления. Это также позволяет задаче управления выполняться с самыми последними данными, полученными от пользовательского ввода-вывода. Подобный сценарий главным образом применяется при доступе к вводу-выводу через модуль LabVIEW FPGA.

Асинхронный недетерминированный ввод-вывод.

Второй сценарий – доступ к недетерминированному устройству ввода-вывода, например последовательному порту или порту Ethernet. В этом случае вы должны назначить пользовательской задаче сканирования ввода-вывода более низкий приоритет, чем у задачи управления. Это позволяет задаче управления выполняться с высоким детерминизмом и надежностью, поскольку на нее не влияет возможно высокий джиттер устройства ввода-вывода в пользовательской задаче сканирования ввода-вывода. Установите время выполнения задачи, исходя из требуемой частоты обновлений. Подобный сценарий чаще всего применяется при чтении значений из измерительного прибора с последовательным портом или подключенного через сеть.

В следующем примере вы узнаете больше об асинхронном недетерминированном вводе-выводе, используемом для запуска и чтения данных из расходомера с интерфейсом RS232. Далее будет рассказано о доступе к вводу-выводу через LabVIEW FPGA.

Добавление записей в табличную память для пользовательского ввода-вывода

Для осуществления чтения и записи в пользовательской задаче сканирования ввода-вывода в этом приложении, создайте Single-Process переменные общего доступа с FIFO реального времени, как было показано в разделе "Обмен данными между задачами".

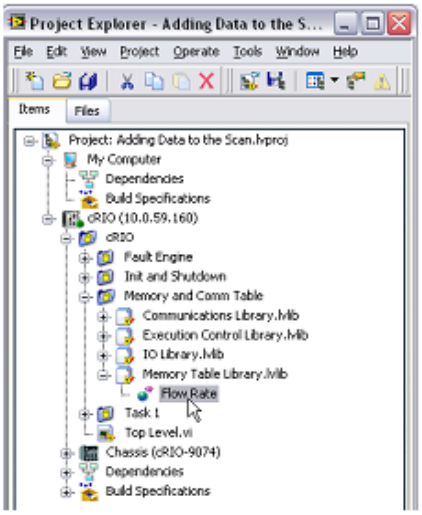


Рисунок 3.27. Используйте переменные общего доступа с буфером FIFO реального времени, чтобы обмениваться пользовательскими данными ввода-вывода с другими задачами

Добавление логики пользовательского сканирования ввода-вывода

После того, как вы добавили еще один синхронизируемый цикл, установили приоритет, задали время выполнения и создали переменные общего доступа с FIFO реального времени для хранения данных, вы готовы добавлять логику доступа к устройству ввода-вывода. Это реализуется следующими действиями:

1. Открытие сессии с устройством ввода-вывода
2. Чтение и запись ввода-вывода
3. Закрытие сессии

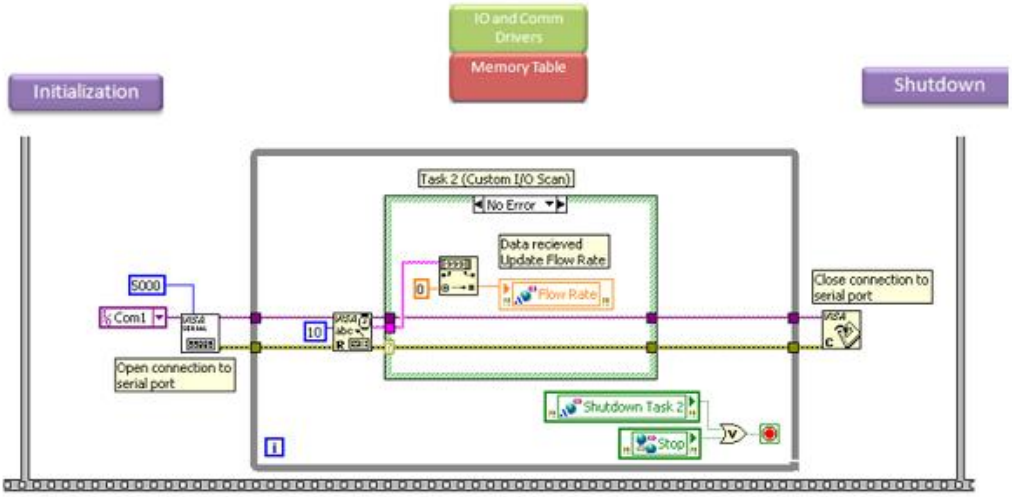


Рисунок 3.28. В пользовательских задачах ввода-вывода, вы открываете сессию, периодически записываете данные в переменную с буфером FIFO реального времени и закрываете сессию

Initialization – инициализация, IO and Comm Drivers – драйверы ввода-вывода и коммуникаций, Shutdown – выключение

Программный доступ к вводу-выводу

В данном разделе приведен пример кода LabVIEW

В усовершенствованных приложениях вам часто требуется программный доступ, конфигурирование и даже обнаружение ввода-вывода во время выполнения программы.

Например, вы можете считывать все значения ввода-вывода с 32-канального цифрового модуля либо с 30 переменными ввода-вывода, либо путем индексации массива ссылок на каналы ввода-вывода, используя программное чтение. Кроме того, вам может потребоваться сконфигурировать этот модуль ввода-вывода для вывода ШИМ-сигналов во время работы приложения, или когда вы добавляете модуль ввода-вывода для развертывания системы. LabVIEW предоставляет необходимые для этого инструменты, которые можно использовать с модулями ввода-вывода C-серии, поддерживающими сканирующий режим CompactRIO. Существуют три основных варианта использования доступа к программируемому вводу-выводу:

1. Динамический выбор каналов для чтения и записи во время выполнения программы.
2. Настройка свойств модуля ввода-вывода, например, масштаба по напряжению и типов термопар.
3. Обнаружение модулей ввода-вывода в развернутой системе.

Чтение и запись ввода-вывода

LabVIEW 2009 включает новый набор VI программного чтения и записи переменных ввода-вывода и переменных, публикуемых в сети. Просто задайте URL переменной, создав константу из ссылки на переменную общего доступа на входе Open Variable Connection VI (Открыть соединение с переменной) и выберите Просмотр (browse).

Рисунок 3.29. Программное чтение и запись в сетевые переменные или переменные ввода-вывода

У вас есть возможность просматривать переменные, развернутые на разных движках, или переменные в вашем проекте.

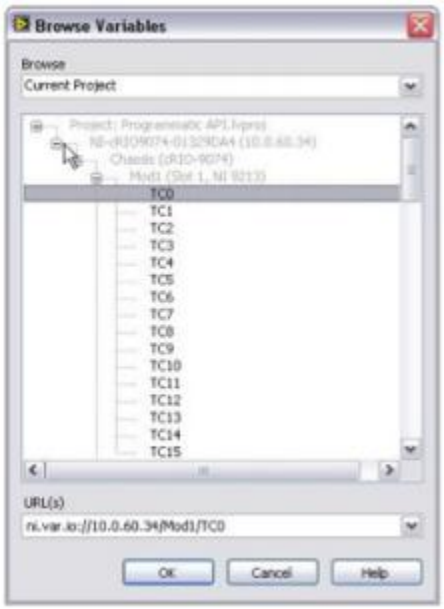


Рисунок 3.30. С помощью просмотра вы можете выбрать любую переменную, как развернутую, так и в проекте LabVIEW

Создавая массив ссылок на канал, вы можете выполнять итерации с несколькими каналами, осуществляя одну и ту же операцию над всеми, как в многоканальном ПИД-приложении, показанном на рисунке 3.31

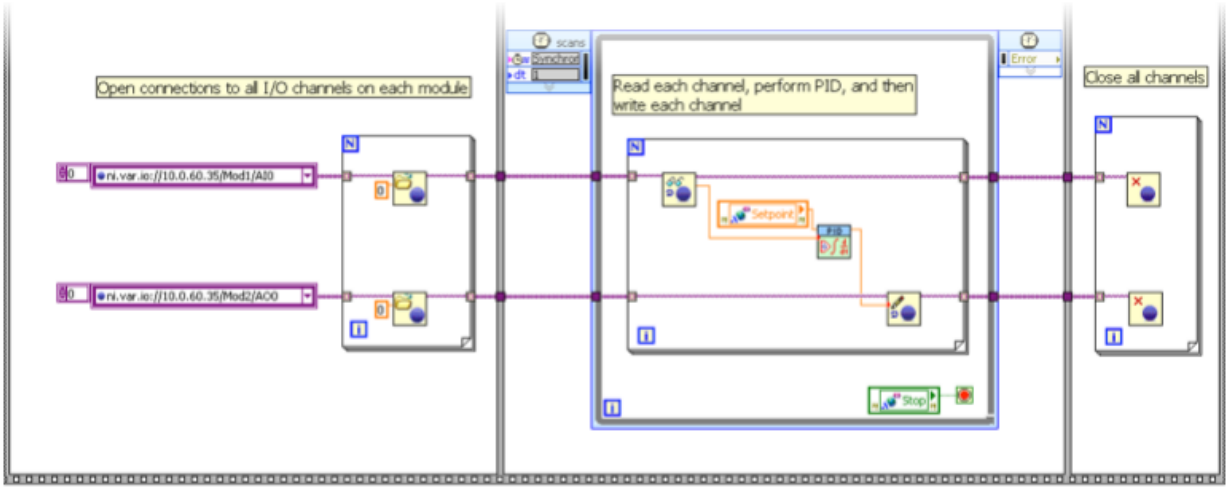


Рисунок 3.31. Программное чтение и запись позволяет использовать более сложные программные архитектуры, например, динамическое многоканальное ПИД-приложение

Open Variable Connection VI также принимает строки для входа каналов, так что вы можете наращивать имена каналов в вашем приложении и принимать решения, в какие из них читать или в какие писать.

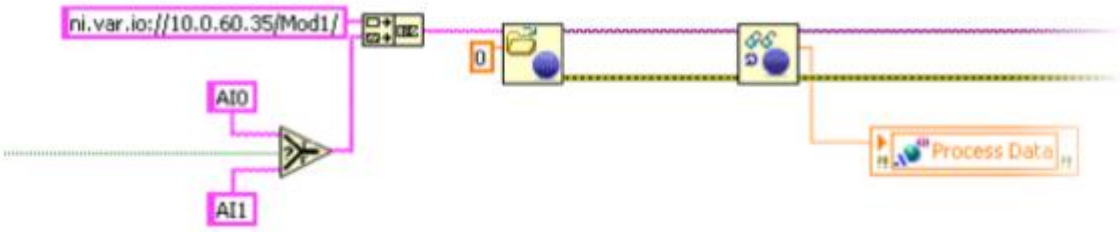


Рисунок 3.32. Вы можете также использовать строки для программного создания списков каналов для чтения или записи

Конфигурирование ввода-вывода

В данном разделе приведен пример кода LabVIEW

Используя программный интерфейс ввода-вывода (programmatic I/O API), вы можете настраивать свойства, доступные в проекте LabVIEW, во время выполнения программы. Например, вы можете устанавливать диапазоны изменения напряжения, типы термопар, свойства ШИМ и т.д. в зависимости от конкретного модуля ввода-вывода.

Это поможет вам настраивать ввод-вывод в развернутой система, когда оператор задает настройки через HMI.

Для конфигурирования создайте ссылки на каждый модуль ввода-вывода, перетащив модуль ввода-вывода из проекта LabVIEW на блок-диаграмму.

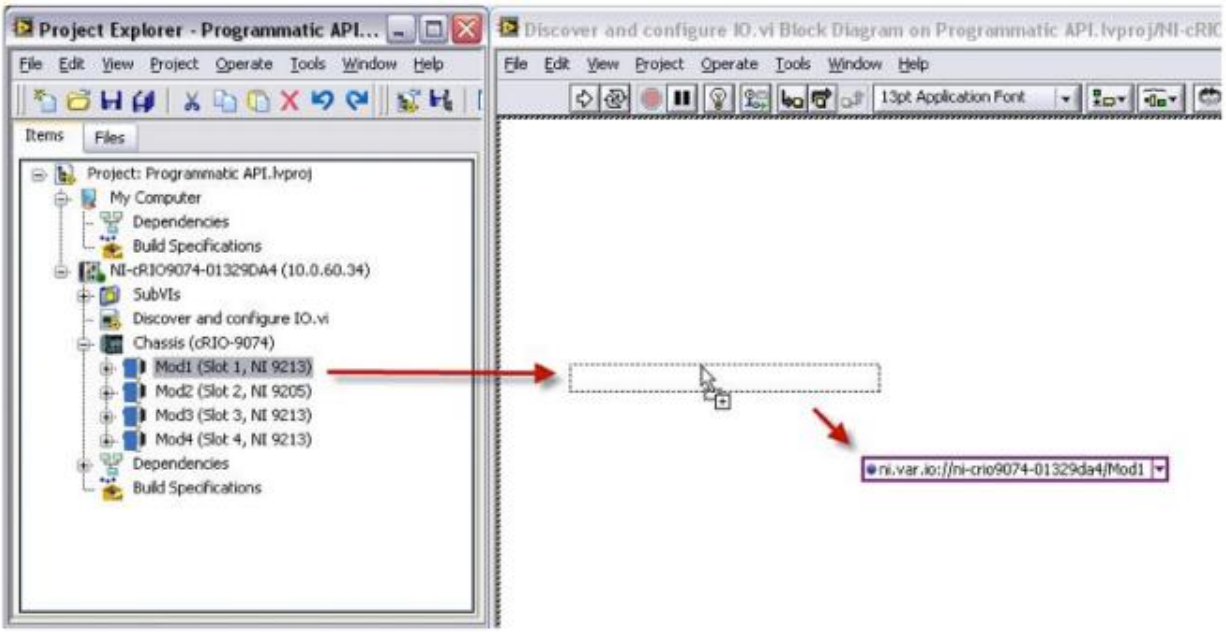


Рисунок 3.33. Вы можете использовать ссылку на модуль для программного изменения настроек конфигурации

Щелкните правой кнопкой мыши по URL модуля и выберите Create»Property for…, чтобы выбрать настаиваемое свойство. В этом примере установите тип термопары J для канала 0 модуля ввода сигналов с термопары NI 9213.

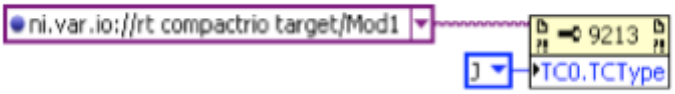


Рисунок 3.34. Узлы свойств доступны для каждого модуля для настройки из кода LabVIEW, а не из проекта

Обнаружение ввода-вывода в развернутой системе

Для максимальной гибкости развернутой системы вы можете запрашивать систему, какие модули в ней присутствуют, во время выполнения программы. Это позволяет развернутой системе управлять и использовать модули ввода-вывода, замененные конечным пользователем непосредственно во время работы.

Для идентификации модулей C-серии в системе CompactRIO, выберите Refresh local I/O VI (Обновить локальный ввод-вывод) из палитры NI Scan Engine (не вызывайте этот VI во время выполнения управляющей программы, поскольку он приводит к джиттеру системы). Как только вы обновите ввод-вывод, сможете обращаться к ссылкам на каждый модуль ввода-вывода через узлы свойств LabVIEW и иерархию объектов. Для обнаружения используйте класс Variable»Variable Object»Variable Container и выберите свойство Children[]. Это свойство возвратит массив ссылок на модули ввода-вывода, присутствующие в системе. Для идентификации модулей проследите массив Children[], используя функцию To More Specific Class (К более конкретному классу) для преобразования ссылок в модуль RSI, и выберите свойство Model and Slot (Модель и слот), как показано на рисунке 3.35.

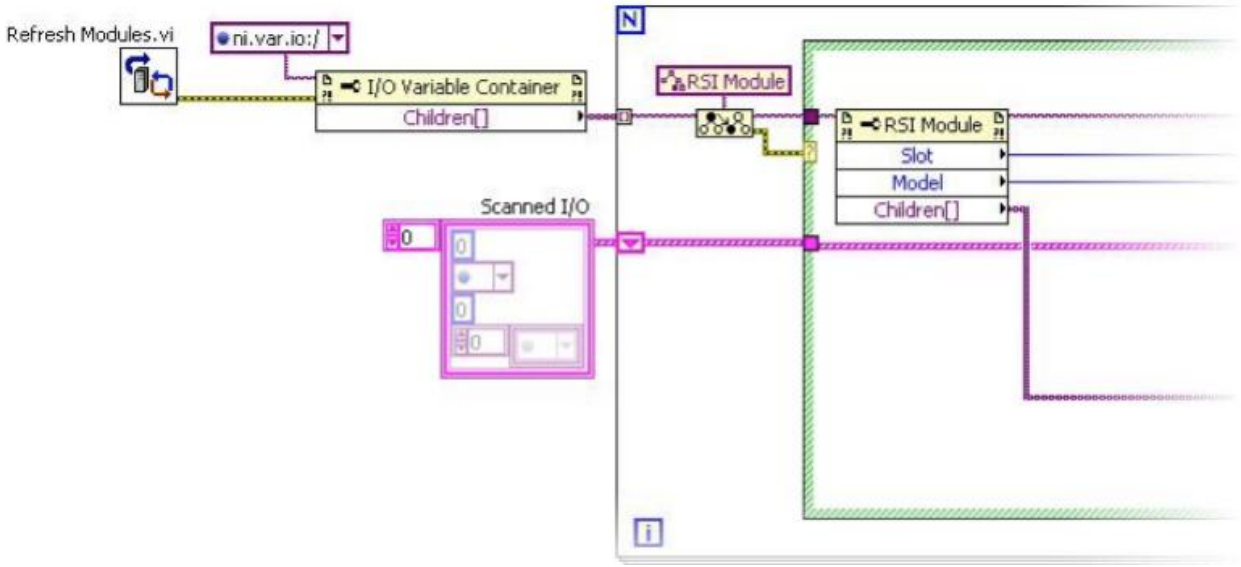


Рисунок 3.35. Код LabVIEW Real-Time определяет сканируемые модули ввода-вывода, которые есть в системе

Свойство Children[] каждого модуля ввода-вывода – это массив ссылок на переменные ввода-вывода, которые вы может использовать для чтения и записи в каждый канал. Прежде чем сделать это, используйте функцию To More Specific Class для приведения ссылок к переменным ввода-вывода.

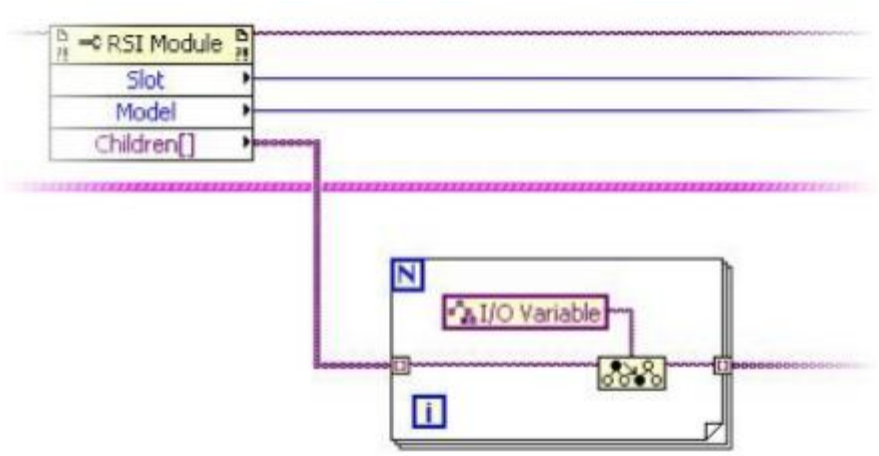


Рисунок 3.36. Используйте свойство Children[] сканируемого модуля для доступа к различным каналам ввода-вывода

Свойства модулей C-серии, обычно указанных в проекте LabVIEW и однажды обнаруженных, могут настраиваться программно. Кроме того, вы можете использовать ссылки на переменные ввода-вывода для чтения и записи в каналы ввода-вывода. Все это и было описано выше.

С этим инструментами вы можете разрабатывать и создавать гибкое самонастраивающееся приложение, адаптирующееся к различным требованиям к вводу-выводу в условиях эксплуатации без необходимости перепрограммирования системы.

Регистрация данных

В данном разделе приведен пример кода LabVIEW

До сих пор вы знакомились с архитектурами для сбора данных, их обработки и управления. Еще одно широко распространенное требование для встраиваемых приложений - запись данных на диск для последующего анализа и изучения. В этом разделе вы узнаете, как создать простую архитектуру регистрации данных. Используя эту архитектуру, пользователь может записывать данные в файл с определенной им самим скоростью, сохранять данные в формате, которым можно легко обмениваться с другими пользователями и/или приложениями, и передачи этих файлов данных из контроллера CompactRIO на хост-компьютер или сервер.

Первый шаг, который необходимо совершить до кодирования – ответить на несколько вопросов:

1. Как много данных необходимо хранить?
2. Какой требуется формат хранения?
3. Какая частота дискретизации регистрируемых данных – медленные одноточечные измерения или измерения сигналов?

Ответы на эти вопросы помогут вам выбрать из множества архитектур сохранения данных и носителей данных, включая внешние жесткие диски USB, внешние карты памяти SD и передачу данных на сервер по протоколу TCP/IP. Множество потенциальных архитектур – пример преимуществ модульной платформы CompactRIO. Рисунок 3.37 иллюстрирует сравнительные характеристики различных устройств и архитектур хранения данных.

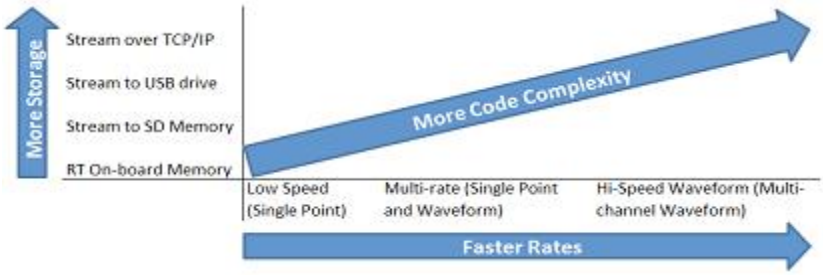


Рисунок 3.37. Множество архитектур для записи данных в CompactRIO

More Storage - увеличение объема, More Code Complexity – увеличение сложности кода, Faster Rates – увеличение скоростей , Stream over TCP/IP – передача по TCP/IP, Stream to USB Drive – передача на USB-диск, Stream to SD Memory – передача на карту памяти SD, RT On-board Memory – встроенная память реального времени , Low Speed (Single Point) – низкая скорость (одноточечные), High Speed (Single Point and Waveform) – высокая скорость (одноточечные и сигналы), High Speed Waveform (Multi-channel Waveform) – высокоскоростные сигналы (многоканальные осциллограммы).

Системы CompactRIO имеет встроенную твердотельную энергонезависимую память емкостью до 2 Гб. Эта память аналогична жесткому диску на ПК; однако она обладает большей надежностью - в ней нет движущихся частей, поддерживается расширенный температурный диапазон, используется более надежная файловая система Reliance. Reliance – файловая система с поддержкой транзакций, разработанная Datalight. Она устойчива к перебоям в питании и устраняет риск повреждения файлов при записи в файловую систему FAT во время отключения питания. CompactRIO также предлагает несущий модуль для съемной памяти SD, а некоторые системы CompactRIO имеют USB-порт, поддерживающий жесткие диски или карты памяти с интерфейсом USB.

В этом разделе делается упор на наиболее распространенной архитектуре регистрации данных, используемой в управляющих системах. Здесь показано, как записывать низкоскоростные, одноточечные данные в файл встроенной памяти контроллера реального времени. В нескольких документах на сайте ni.com подробно рассматриваются другие архитектуры, но в данном разделе делается упор на доскональном обзоре низкоскоростной архитектуры регистрации данных, в том числе:

1. Чтение из табличной памяти ввода-вывода на скорости, определяемой пользователем
2. Запись данных в файл на контроллер реального времени
3. Передача данных по FTP из контроллера реального времени автоматически или вручную

Вы можете выбирать различные форматы файлов при записи тестовых данных на диск. У каждого формата есть свои преимущества и недостатки. В данном разделе рассматриваются два формата, TDMS и ASCII.

Сначала рассмотрим создание кода для регистрации данных, запускающегося в контроллере CompactRIO и сохраняющего данные в файл TDMS.

Регистрация данных во встроенной памяти реального времени, файл TDMS

Technical Data Management Streaming (TDMS) - формат файлов, разработанный специально для инженеров и ученых для потоковой передачи хорошо задокументированных и организованных данных на диск. Файлы TDMS - двоичные файлы, включающие три уровня иерархии, облегчающих сохранение пользовательских свойств на каждом из этих уровней, придавая тестовым файлам большую структурированность. TDMS файлами легко обмениваться в пределах программной платформы NI, а также с другими распространенными приложениями наподобие Microsoft Excel и OpenOffice. Благодаря своей естественной иерархии и высокой скорости формат TDMS рекомендуется NI для хранения результатов измерений.

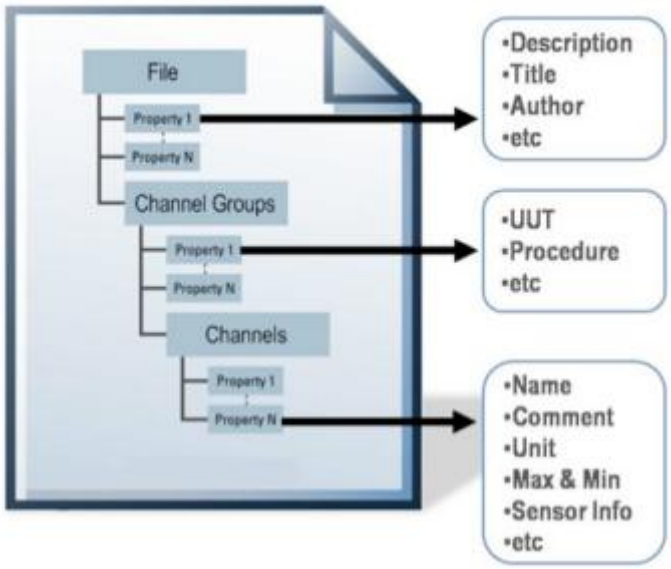


Рисунок 3.38. Файлы TDMS содержат естественную иерархию и свойства
File - файл, Channel Groups – группы каналов, Channels - каналы, Property 1 (N) – свойство 1 (N), Description - описание, Title - наименование, Author - автор, Etc – прочее, UUT – объект испытаний, Procedure - процедура, Name - имя , Comment – комментарий, Unit – единица измерения, Max&Min – максимум и минимум, Sensor Info – информация о датчике

Инициализация файла

Сначала необходимо инициализировать файл TDMS, в который записываются данные. На этом шаге выберите имя файла, которое будет полезно для определения соответствия с тестом, результаты которого подлежат сохранению.

В устройствах реального времени используются буквы для указания различных дисков, как и в настольных компьютерах. Системы CompactRIO обращаются к внутренней энергонезависимой памяти при помощи буквенного обозначения C, а к внешним USB-дискам при помощи буквенного обозначения U. Чтобы получить доступ к файлу в памяти устройства реального времени, используйте путь, начинающийся с C:\, как и на обычном ПК. Если вы хотите зарегистрировать данные, собранные устройством реального времени, в файл на том же устройстве, вы можете использовать TDMS Write.vi (Запись в TDMS) в LabVIEW и путь C:\[имя папки]\[имя файла] в качестве входного параметра VI.

В следующем примере используется константа пути для задания требуемого местоположения папки "c:\datalogging", затем она преобразуется в строку. Также там используется функция "Get Date/Time" (Получить дату/время) для создания строки с датой – в данном случае, "Oct 6, 11:30:05 AM". Затем, используя функции конкатенации ("Concatenate String") и преобразования строки в путь ("Convert string to path"), создается желаемое имя файла. И, наконец, этот путь подается на вход функции "TDMS Open". (Открыть TDMS).

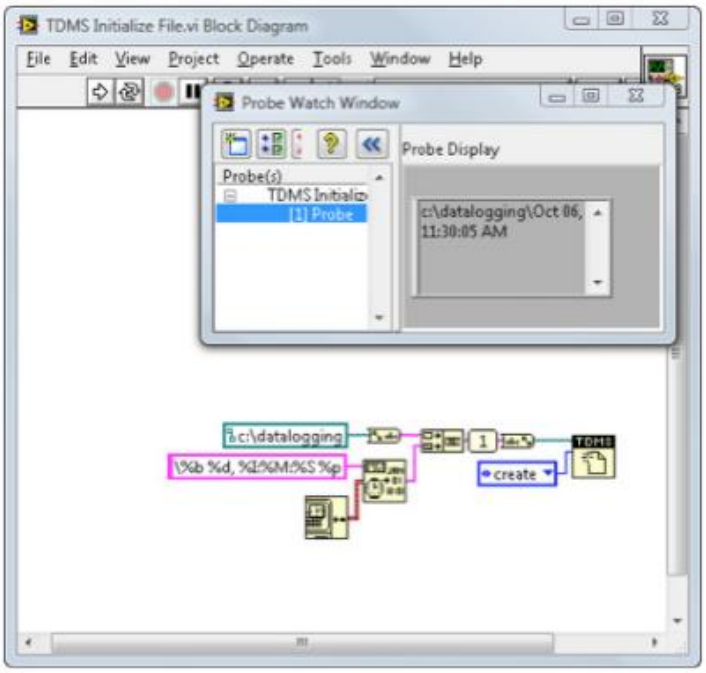


Рисунок 3.39. Инициализация файла TDMS согласованным именем для потоковой записи данных

Запись данных в файл

В следующем фрагменте кода, выделенном красной рамкой, вы записываете данные в файл. В этом фрагменте текущие значения ввода-вывода, запомненные в таблице отображения ввода-вывода (I/O Mapping Table), считываются с определяемой пользователем скоростью и записываются в файл. Эта архитектура состоит из отдельного цикла, в котором выполняется чтение значений ввода-вывода и запись их в файл. Она также содержит функцию синхронизации для обеспечения того, чтобы цикл выполнялся с заданной пользователем частотой.

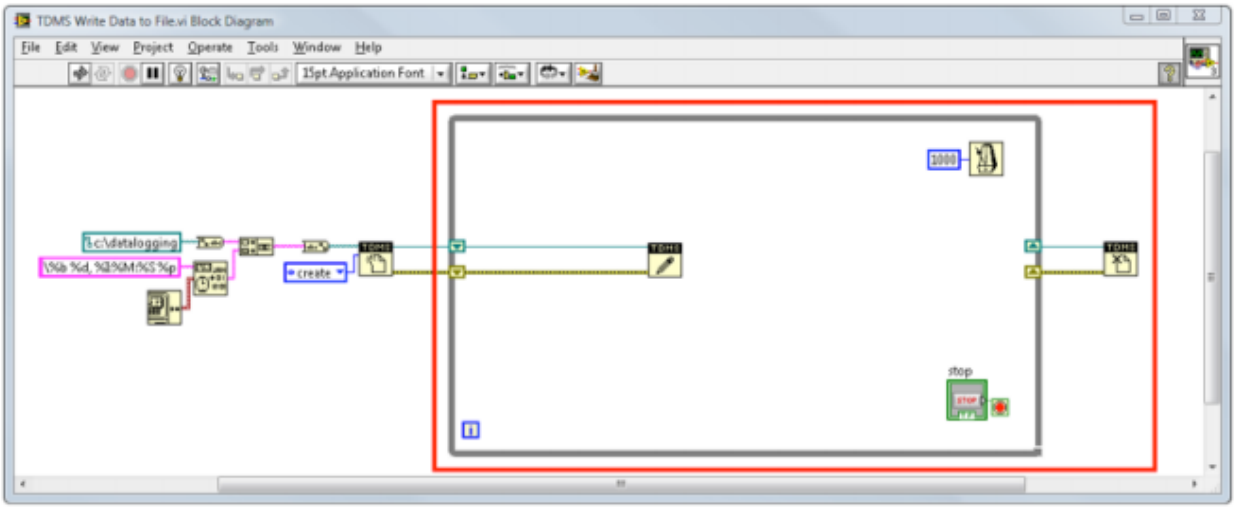


Рисунок 3.40. Запись данных в файл

Считывание данных

Третий шаг, показанный в красной рамке – чтение текущих значений из табличной памяти и запись их в файл TDMS. На этом шаге поместите переменные общего доступа, которые нужно записать в файл, внутрь цикла while. Затем поместите значения в массивы тех же типов данных и запишите их в файл.

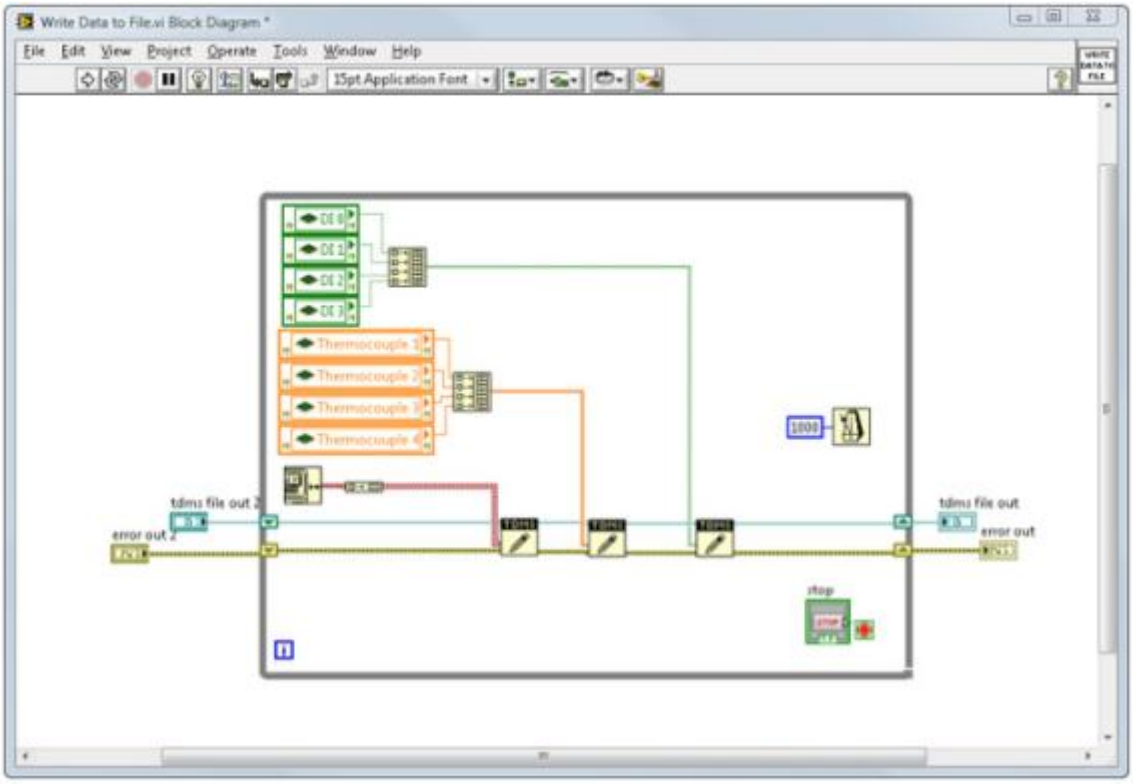


Рисунок 3.41. Считывание и форматирование данных

Как было упомянуто выше, файлы TDMS имеют три уровня иерархи и поддерживают сохранение свойств на каждом из этих уровней. Для лучших документирования и организации данных пользователь может группировать данные в пределах файла, основываясь на типах данных, видах измерений и так далее, и добавлять такие свойства, как идентификаторы датчиков, дату, и т.п. Для получения дополнительной информации о формате и иерархии файлов TDMS посетите сайт visit ni.com/tdms.

Динамическое создание новых файлов

Далее вы можете изменить код, чтобы динамически закрывать существующий файл и создавать новый файл через определяемые пользователем интервалы. Затем пользователь может передавать по FTP старые данные из контроллера CompactRIO на хост-компьютер для обработки, хранения и/или записи на SQL-сервер. Имея фрагмент кода, который автоматически закрывает существующий файл и создает новый с определенным пользователем периодом, пользователь может получать доступ к ранее записанным файлам до завершения выполнения приложения. Чтобы реализовать такую возможность, используйте экспресс-VI "Elapsed Time" (Прошедшее время) для наблюдения за тем, сколько прошло времени. По завершении определенного временного интервала времени (в данном примере – один час), он закрывает существующий файл и создает новый, используя те же функции, что были приведены в разделе "Инициализация файла" этого документа.

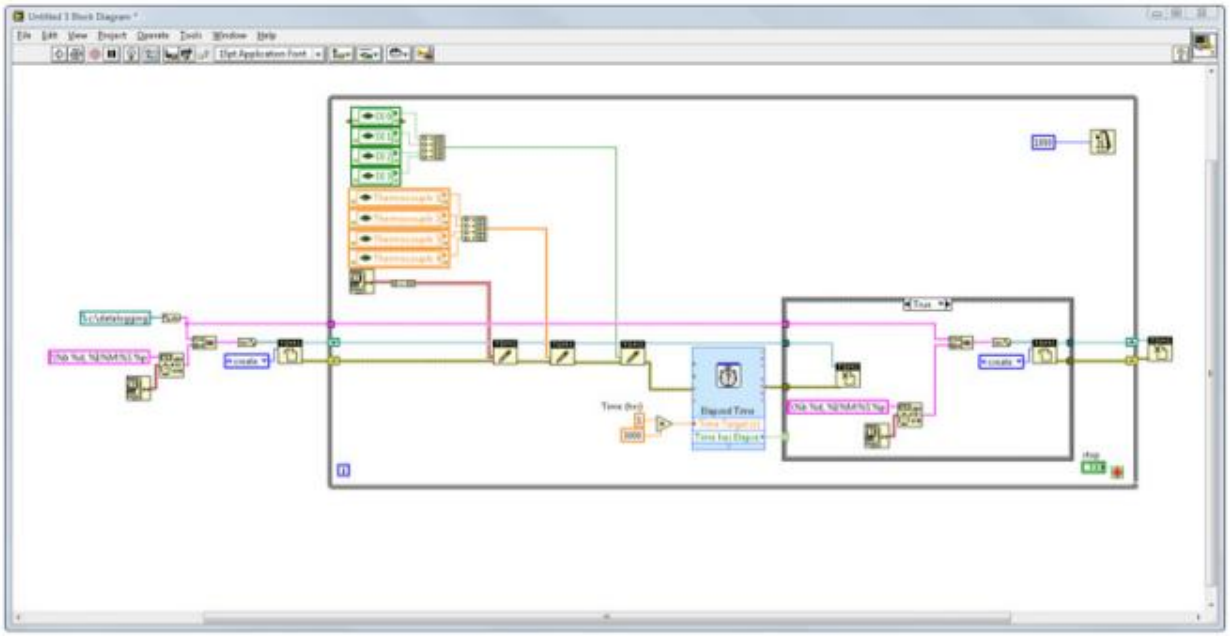


Рисунок 3.42. Динамическое создание новых файлов

Регистрация данных во встроенную память реального времени, ASCII файл

Файлы ASCII - один из самых распространенных способов сохранения данных на диске благодаря простоте и взаимозаменяемости этого формата. Файлы ASCII легко могут быть прочитаны любым табличным или текстовым редактором. Минусы записи в ASCII файлы в том, что они, как правило, больше двоичных фалов и обладают значительно меньшей скоростью записи.

Инициализация файла

Сначала необходимо инициализировать файл ASCII, в который будут записываться данные. На этом шаге важно выбрать имя файла, полезное для определения хранимых тестовых данных.

В устройствах реального времени используются буквы для указания различных дисков, как и в настольных компьютерах. Системы CompactRIO обращаются к внутренней энергонезависимой памяти при помощи буквенного обозначения C, а к внешним USB-дискам при помощи буквенного обозначения U. Чтобы получить доступ к файлу в памяти устройства реального времени, используйте путь, начинающийся с C:\, как и на обычном ПК. Если вы хотите регистрировать данные, собранные устройством реального времени, в файл на том же устройстве, вы можете использовать Write to Text File.vi (Запись в текстовый файл) в LabVIEW и путь C:\[имя папки]\[имя файла] в качестве входного параметра VI.

В следующем примере используется константа пути для задания требуемого местоположения папки "c:\datalogging", а затем константа преобразуется в строку. Также там используется функция "Get Date/Time" (Получить дату/время) для создания строки с датой, в данном случае – "Oct 6, 11:30:05 AM". Затем, используя функции конкатенации строк ("Concatenate String") и преобразования строки в путь ("Convert string to path"), создается желаемое имя файла. И, наконец, этот путь подается на вход функции "Open/Create/Replace File" (Открыть/Создать/Заменить файл).

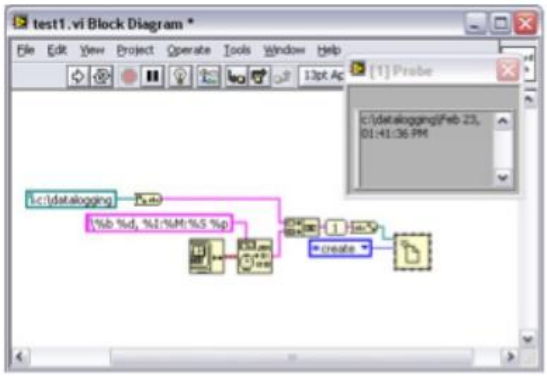


Рисунок 3.43. Инициализация ASCII-файла согласованным именем для потоковой записи данных

Запись данных в файл

В следующем фрагменте кода, выделенном красной рамкой, вы записываете данные в файл. В этом фрагменте текущие значения ввода-вывода, хранимые в таблице отображения ввода-вывода (I/O Mapping Table), считываются с определяемой пользователем скоростью и записываются в файл. Эта архитектура состоит из отдельного цикла while, в котором значения ввода-вывода считываются и записываются в файл. она Здесь также размещена функция синхронизации для обеспечения заданного пользователем времени выполнения цикла.

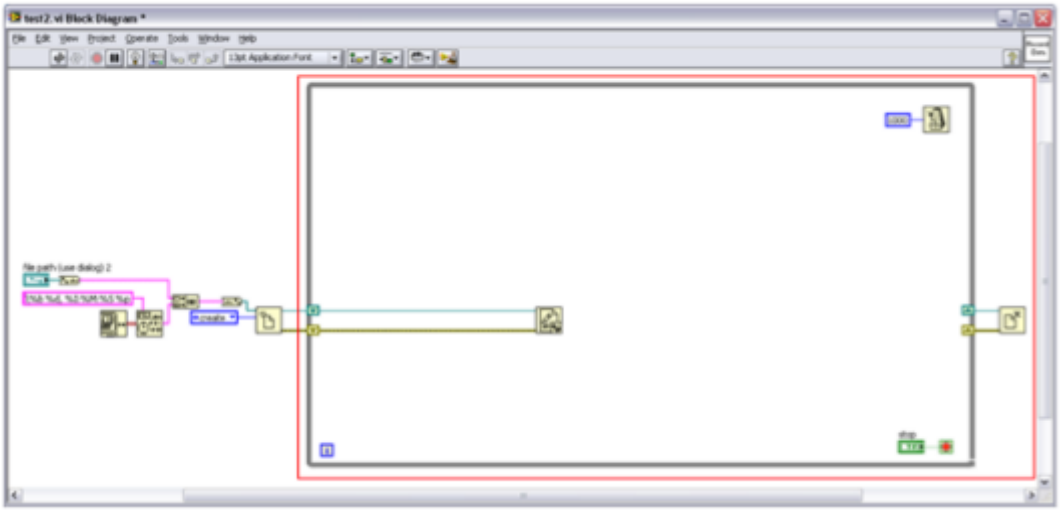


Рисунок 3.44. Запись данных в файл

Считывание и форматирование данных

Третий шаг, показанный в красной рамке – чтение текущих значений из табличной памяти и форматирование их в в строку. На этом шаге поместите переменные общего доступа, которые нужно записать в файл, внутрь цикла while. Затем объедините значения в массивы, преобразуйте их в строку ASCII и запишите их в файл.

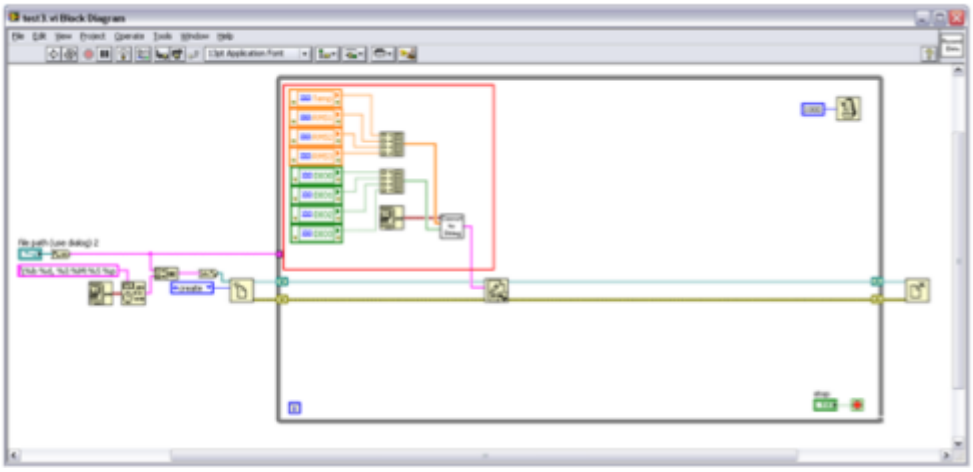


Рисунок 3.45. Считывание и форматирование данных

Эти переменные могут быть любого типа данных; однако вы должны преобразовать их в строку, прежде чем записывать в файл. В этом примере преобразование происходит в subVI. В этом subVI аналоговые значения типа dbl обрабатываются отдельно от цифровых логических значений. В конце, до записи в файл ASCII, должна быть создана единственная строка, содержащая все значения ввода-вывода, разделенные табулятором или запятой. Символ табуляции или запятая должны быть между каждым значением для их разделения, а символ "конец строки" – должен использоваться для указания конца строки данных. Это позволяет табличным редакторам извлекать из файла столбцы и строки.

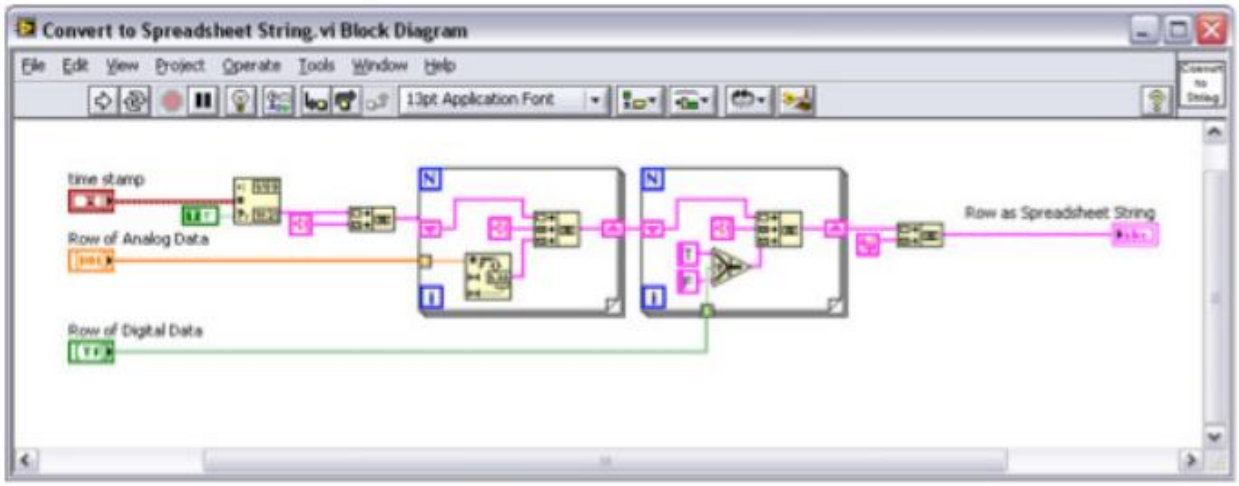


Рисунок 3.46. Форматирование данных в строку ASCII

Динамическое создание новых файлов

В последнем фрагменте добавляется код, динамически закрывающий существующий файл и создающий новый файл через определяемые пользователем интервалы времени. Затем пользователь может передавать по FTP старые данные из контроллера CompactRIO на хост-компьютер для обработки, хранения и/или записи на SQL-сервер. Если пользователь непрерывно пишет в единственный файл, то невозможно получать данные с контроллера, прежде чем он закончит работу, кроме того, это приводит к ограничениям на объем сохраняемых данных.

Имея фрагмент кода, который автоматически закрывает существующий файл и создает новый с определенной пользователем интервалом времени, пользователь может получать доступ к ранее записанным файлом до завершения выполнения приложения. Чтобы реализовать эту возможность, используйте экспресс-VI "Elapsed Time" (Прошедшее время) для контроля за прошедшим временем. По завершении определенного интервала времени (в данном примере – один час), он закрывает существующий VI и создает новый, используя те же функции, что были приведены в разделе "Инициализация файла" этого документа.

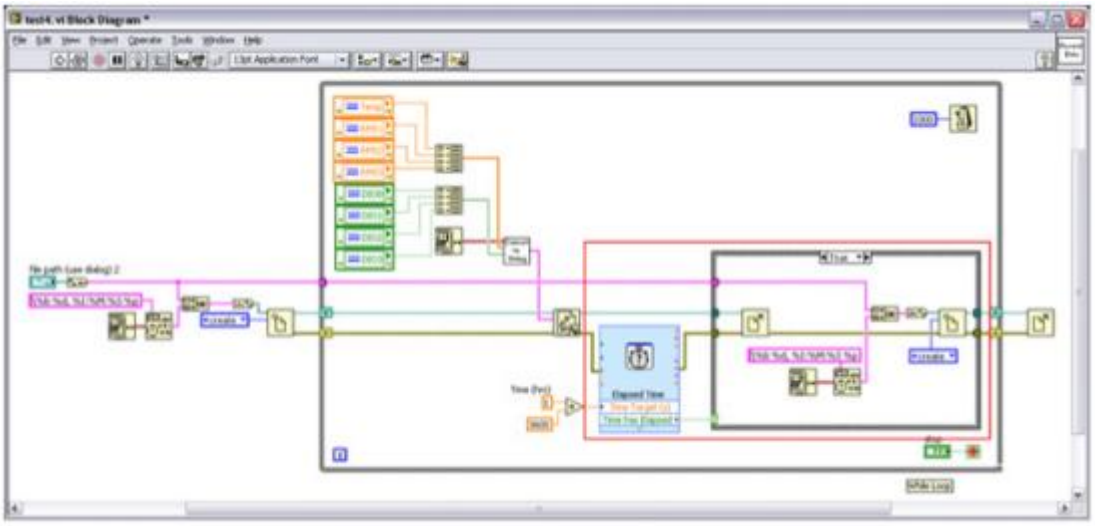


Рисунок 3.47. Динамическое создание новых файлов

Интеграция кода регистрации данных с управляющей архитектурой

Теперь, когда вы детально изучили архитектуру регистрации данных, рассмотрим, как интегрировать ее с управляющим кодом.

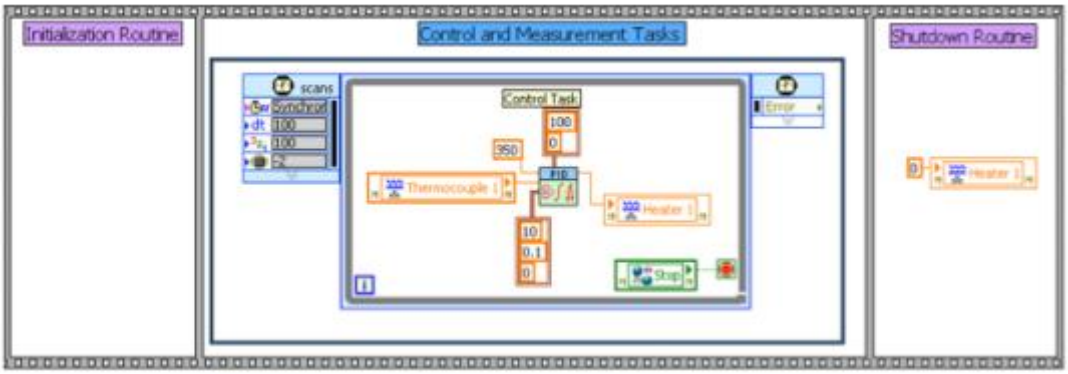


Рисунок 3.48. Управляющая архитектура без регистрации данных

Для добавления регистрации данных к этой управляющей архитектуре, просто добавьте приведенный ранее код регистрации данных параллельно задаче управления.

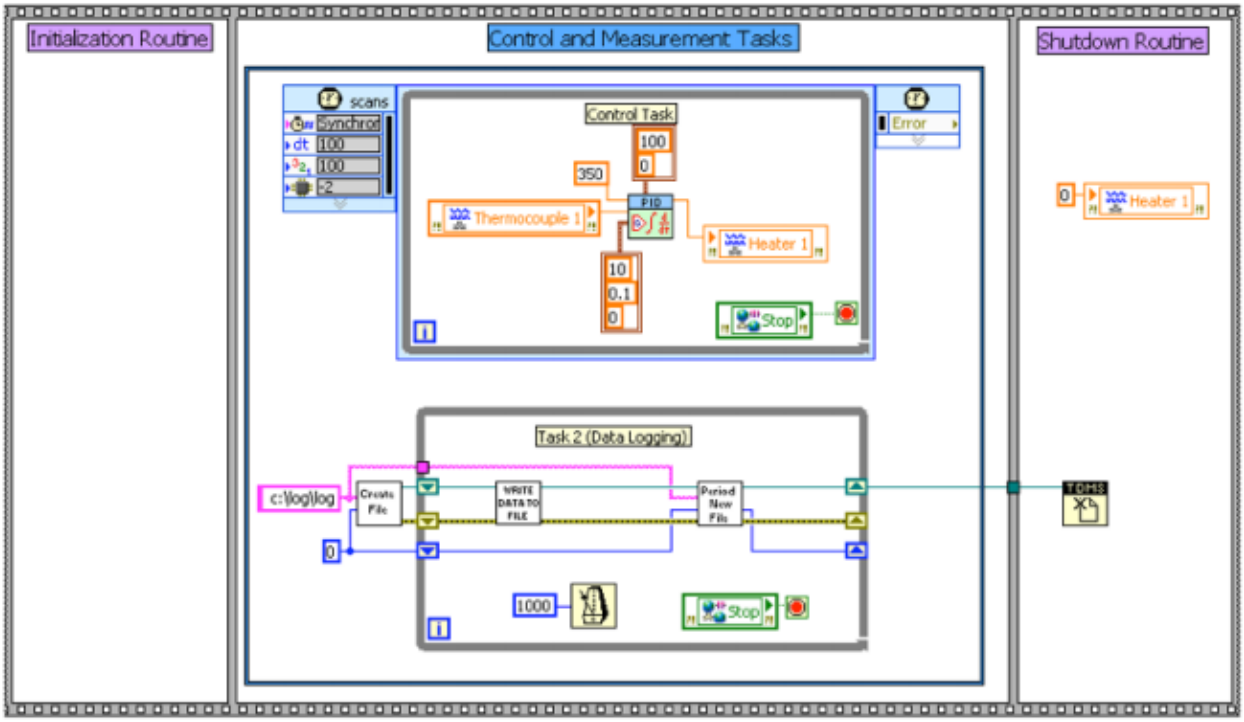


Рисунок 3.49. Управляющая архитектура с регистрацией данных

Извлечение зарегистрированных данных

На контроллерах CompactRIO выполняется FTP server, так что вы можете просматривать файлы, зарегистрированные в контроллере, с помощью любого FTP-клиента, включая Web-браузеры.

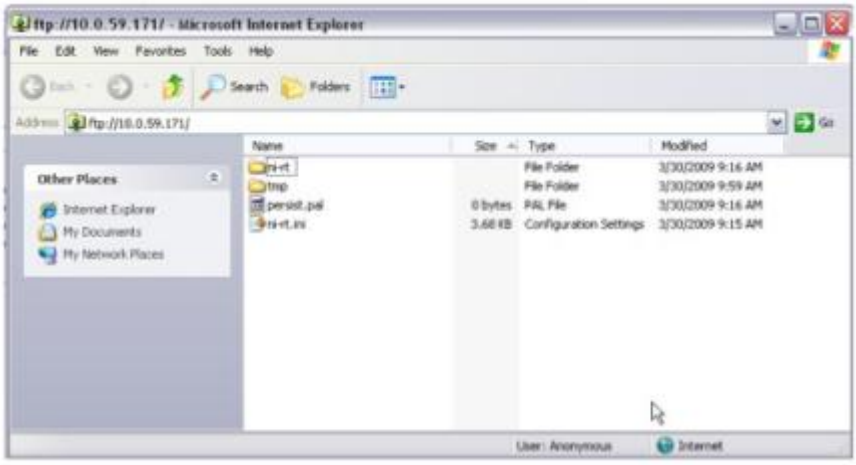


Рисунок 3.50. Используя FTP, вы можете напрямую извлекать данные из контроллера CompactRIO

Web-браузеры

Большинство Web-браузеров сохраняют в кэш-памяти информацию с посещаемых вами веб-страниц, чтобы не загружать информацию каждый раз, когда вы посещаете эту страницу. Вместо этого они загружают страницы из кэша на вашем жестком диске. Поэтому вы не видите новых файлов в вашем контроллере CompactRIO. Вам потребуется перезагрузить или обновить страницу в вашем веб-браузере или же изменить настройки кэша, чтобы браузер каждый раз проверял новое содержимое.

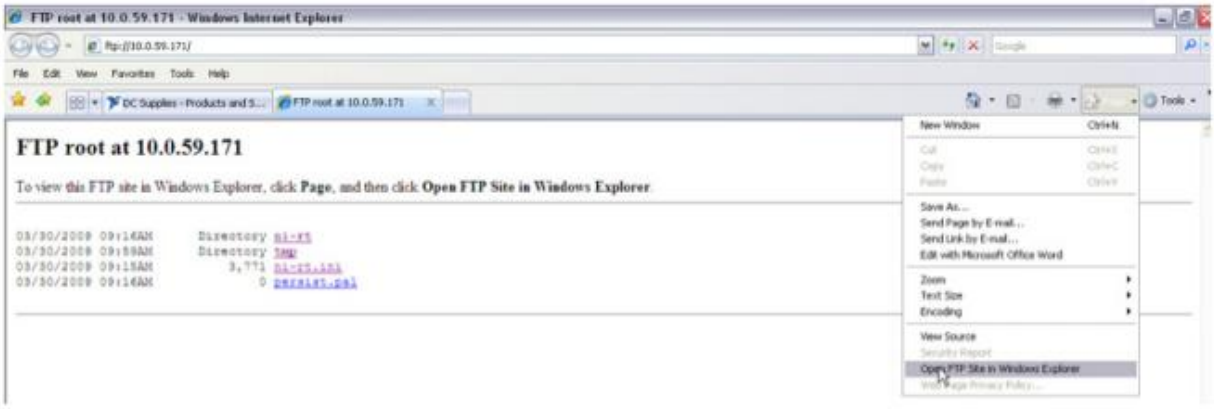


Рисунок 3.51. Подключение к контроллеру CompactRIO через Windows Internet Explorer

Программный способ

Для автоматической передачи файлов из контроллера CompactRIO в ПК под управлением Windows, вы можете использовать VI FTP, имеющийся в LabVIEW Internet Toolkit. Запустите программу на ПК, которая периодически получает новые файлы с контроллера. Ниже приведен простой пример, как переместить файл из устройства реального времени на хост-компьютер. Для устройств реального времени пользователь по умолчанию – аноним, а пароль – пустая строка. Хост – это IP-адрес устройства реального времени.

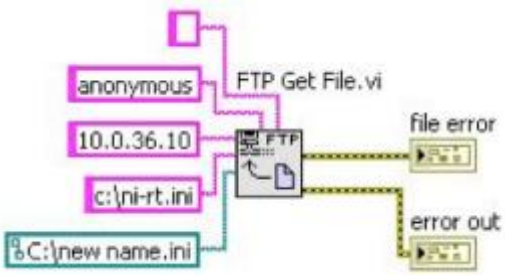


Рисунок 3.53. LabVIEW Internet Toolkit предоставляет VI для работы с FTP, которые вы можете использовать при создании Windows-приложения, программно получающего файлы из системы CompactRIO

Ошибки и сбои

Кластер ошибок в LabVIEW – важный инструмент для отслеживания и контроля ошибок. Хорошо зарекомендовавший себя и хорошо задокументированный инструмент для отслеживания проблем с программным обеспечением и аппаратными средствами, кластер ошибок идеален для систем, где всегда присутствует оператор. Для сложных управляющих приложений, которые часто запускаются автономно, вы можете усовершенствовать кластер ошибок возможностями передачи ошибок между параллельными задачами, управления циклами, осуществляющими некие действия при возникновении ошибок и возможностью удаленного контроля ошибок. В LabVIEW 8.6 системы CompactRIO, работающие со механизмом сканирования NI Scan Engine, введен новый механизм – NI Fault Engine (обработчик ошибок).

Обработчик ошибок

Обработчик ошибок – область памяти, которая записывает и передает ошибочные состояния системы CompactRIO. По умолчанию он записывает и сообщает о любых ошибках сканирования ввода-вывода. Вы можете добавить в обработчик новые сбои и наблюдать за ними программно.

Кроме того, вы можете наблюдать и сбрасывать ошибки из Менеджера распределенных систем (NI Distributed System Manager).

Чтобы использовать обработчик сбоев, вы должны

- Записывать все ошибки
- Определить подходящую логику для всех ошибок
- Создать цикл обработки ошибок для наблюдения и реагирования на все ошибки

Запись ошибок

Для добавления новых ошибок в обработчик сбоев, используйте Set Faults.vi. На вход ему подается кластер ошибок. Если происходит ошибка, Set Faults.vi автоматически регистрирует сбой. Вы можете также вводить сбои, определенные пользователем, создав и подключив определенный пользователем кластер ошибок. В Менеджере распределенных систем (NI Distributed System Manager), появляется текст ошибки, если вы создадите пользовательский файл ошибок и сохраните его на каждой машине, работающей с менеджером систем.

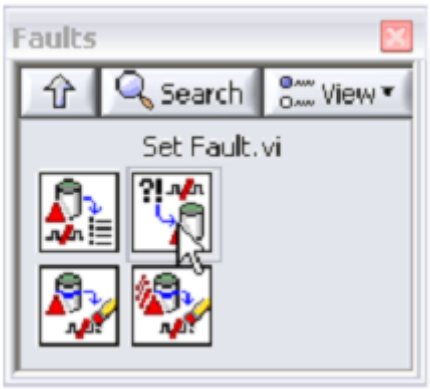


Рисунок 3.53. Палитра VI работы со сбоями

Логика ошибок

Для каждого приложения вам нужно определить подходящее действие при возникновении ошибки. Ошибки всегда указывают на то, что с процессом что-то неправильно, но предпринимаемые действия могут различаться в зависимости от серьезности проблемы. Например, если ошибка указывает на то, что у вас отказала термопара, вы можете решить уведомить соответствующую службу, но продолжать управляющий процесс, используя другие значения для оценки температуры. Но если вы получите ошибку, что управление движением не следует профилю программы, вам потребуется немедленно выключить машину, пока она не будет исправлена. Вы можете считывать код ошибок для любых заданных типов сбоев и определять, какое действие выполнять. В этом простом примере вам необходимо отключить машину, если одна и та же ошибка произойдет десять раз.

Цикл обработки сбоев

Вам нужно создать цикл для наблюдения за сбоями и выполнения логики обработки сбоев. Этот цикл должен иметь высокий приоритет, чтобы он выполнялся всегда и не прерывался задачами с низким приоритетом.

Пример кода с циклом обработки сбоев

В данном разделе приведен пример кода LabVIEW

В этом простом примере вы наблюдаете за сбоями в главной задаче. Отметим, что вы создаете сбой, определенный пользователем, если синхронизируемый цикл завершается с опозданием (то есть вы не успели выполнить логику вовремя). Программа обработки сбоев считывает все сбои и выполняет вашу логику. Она может принять решение выключить контроллер, послав соответствующую команду. В программе инициализации вы также должны добавить код, чтобы очистить все сбои и записать сбои, возникающие при инициализации.

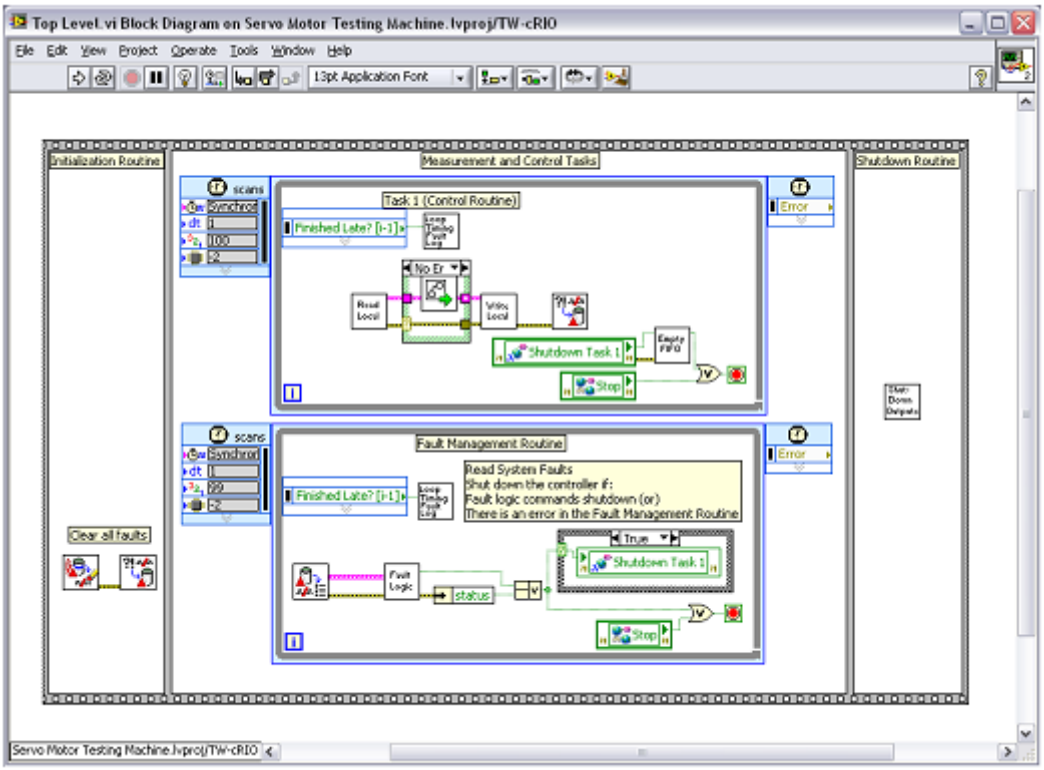


Рисунок 3.54. Завершенное приложение с обработкой сбоев

Сторожевой таймер реального времени

Чтобы обеспечить дополнительную защиту от ошибок программирования, например, утечки памяти или нарушения приоритетов, все контроллеры NI реального времени имеют встроенный аппаратный таймер, обычно называемый сторожевым таймеров. Вы можете использовать сторожевой таймер для инициализации процедуры восстановления, если программ не отвечает.

Осмысление аппаратно-программного интерфейса

Сторожевой таймер – это аппаратный счетчик, который взаимодействует со встроенным программным приложением, обнаруживает ошибки программы и восстанавливает процесс ее выполнения. Во время нормальной работы программное приложения инициализирует аппаратный счетчик для обратного отчета от заданной отметки с определенным инкрементом и определяет выполняемое действие при достижении нуля. После того, как приложение запускает сторожевой таймер, периодически это приложение перезагружает таймер, чтобы он никогда не достиг нуля.

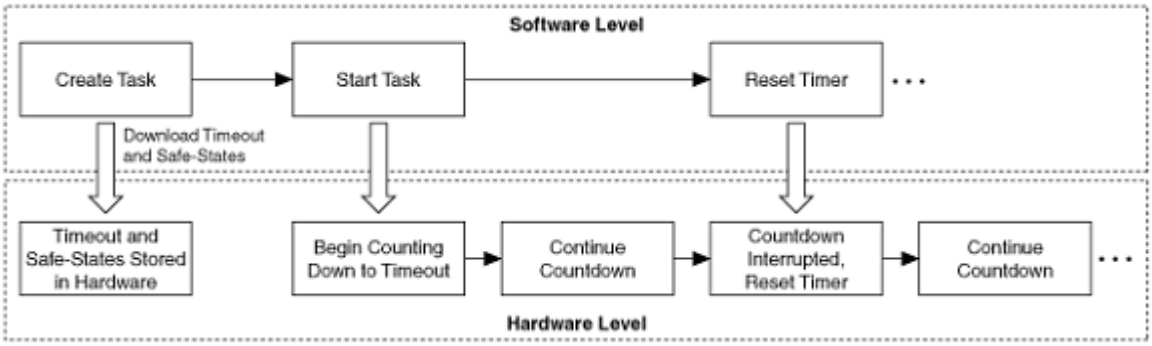


Рисунок 3.55. Сторожевой таймер – аппаратный механизм, который может выполнять какие-то действия, если программа перестанет отвечать

Software Level – уровень программного обеспечения , Hardware Level – уровень аппаратных средств, Create Task – создание задачи, Start Task – запуск задачи, Reset Timer – сброс таймера, Download Timeouts and Safe-States – загрузка тайм-аутов и безопасных состояний, Timeout and Safe-States Stored in Hardware – тайм-ауты и безопасные состояния, хранимые в аппаратуре, Begin Counting Down to Timeout – начало обратного отсчета тайм-аута, Continue Countdown – продолжение обратного отсчета, Countdown Interrupted, Reset Timer – прерывание обратного отсчета, сброс таймера

Если сбой программы не даст приложению сбросить таймер, в конце концов время тайм-аута истечет, поскольку аппаратный счетчик не зависит от программного обеспечения и продолжает обратный отсчет, пока не достигнет нуля. Когда это произойдет, аппаратура запустит процедуру восстановления.

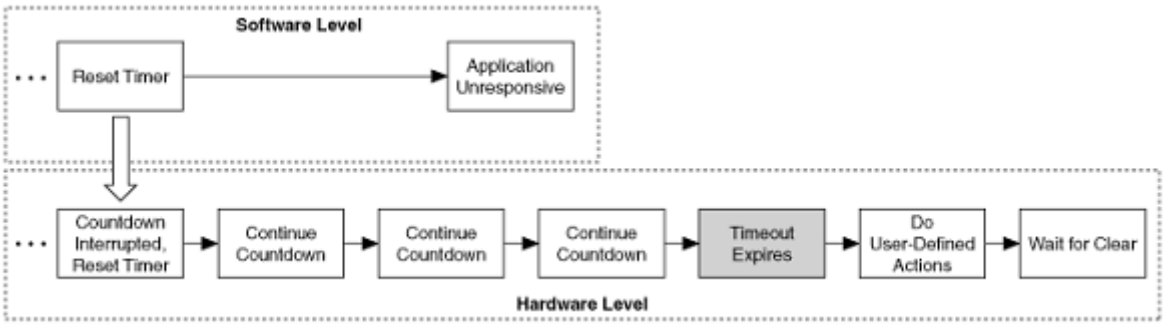


Рисунок 3.56. Если программное обеспечение не сбросит аппаратный триггер, аппаратные средства могут перезапустить контроллер или запросить прерывание

Software Level – уровень программного обеспечения , Hardware Level – уровень аппаратных средств, Continue Countdown, Reset Timer – продолжение обратного отсчета, сброс таймера, Continue Countdown – продолжение обратного отсчета, TimeOut Expires – окончание тайм-аута, Do User-defined Actions – выполнение определенных пользователем действий, Wait for Clear – ожидание сброса в исходной состояние

Вы можете получить доступ к встроенному сторожевому таймер с помощью VI палитры Real-Time Watchdog. Используйте Watchdog Configure VI (Настройка сторожевого таймера) для установки тайм-аута сторожевого таймера и определения действия или действий, которые необходимо выполнить по завершении тайм-аута.

Используйте Watchdog Whack VI для периодического сброса таймера, прежде чем завершится тайм-аут.

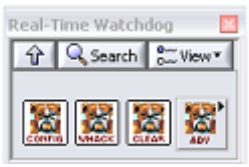


Рисунок 3.57. Палитра сторожевого таймера реального времени

Выбор подходящей настройки тайм-аута

Адекватный диапазон значений тайм-аута зависит от конкретных характеристик производительности и требований ко времени готовности встроенного приложения. Вы должны

установить тайм-аут достаточно длинным, чтобы он не завершался при приемлемом значении джиттера системы, однако достаточно коротким, чтобы система могла оправиться от отказа достаточно быстро, чтобы удовлетворять требованиям ко времени готовности.

Аппаратный сторожевой таймер

Модифицируйте код обработчика ошибок для разрешения работы аппаратного сторожевого таймера. В подпрограмме инициализации сконфигурируйте сторожевой таймер. В этом примере вы настраиваете таймер на одну секунду и определяете действие по перезагрузке контроллера. В цикле сбоев помещена функция wack the dog, которая сбрасывает таймер. Наконец, в подпрограмме выключения выполняется очистка, если успешно завершена операция установки выходов в состояние "отключено".

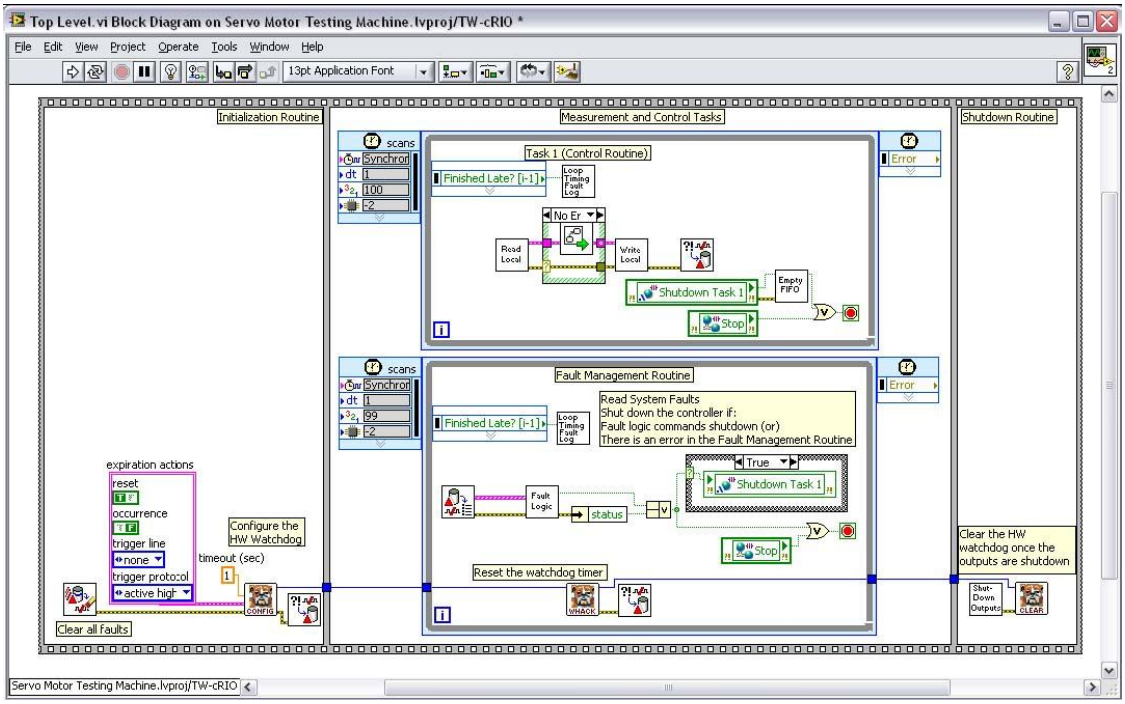


Рисунок 3.58. В этом простом приложение надежность увеличена благодаря использованию аппаратного сторожевого таймера

РАЗДЕЛ 4
Обмен информацией с системами CompactRIO

Обзор способов обмена информацией

Приложения управления механизмами часто содержат различный набор систем, которые должны обмениваться информацией друг с другом. Центральный контроллер может читать данные из периферийных измерительных приборов, получать данные вводимые оператором через HMI или посылать результаты тестирования в базу данных управления предприятием на корпоративном сервере. Аппаратные средства CompactRIO предлагают несколько возможностей для коммуникаций, включая следующие:

- Связь непосредственно через контроллер реального времени
  - С использованием портов Ethernet (TCP, UDP, переменные общего доступа, Modbus/TCP, EtherNet/IP, EtherCAT)
  - С использованием последовательных портов (RS232, Modbus, пользовательские протоколы)
- Связь с использованием встраиваемых модулей
  - Последовательных портов (RS232/RS422/RS485)
  - CAN
  - PROFIBUS
  - Для необработанных данных ввода-вывода (Raw I/O) (для пользовательских протоколов)

При выборе способа обмена информацией рассмотрите природу и тип информации, которой нужно обмениваться. Вообще приложения управления механизмами могут содержать следующие типы коммуникаций:

- Коммуникации, основанные на сообщениях
- Обмен данными по ходу процесса
- Потоковые/буферизированные данные

Способы обмена информацией, основанные на командах или сообщениях

Обмен информацией, основанный на командах или сообщениях происходит относительно нечасто и вызывается неким особым событием. Например, если пользователь нажимает кнопку HMI, чтобы остановить конвейерную ленту. От HMI к контроллеру должно быть послано сообщение, которое останавливает конвейер. При обмене информацией, основанном на сообщениях, важно гарантировать своевременную доставку сообщения. В предыдущем примере, когда оператор нажимает кнопку "стоп", он ожидает немедленной реакции (человеческое восприятие "немедленной реакции" - порядка десятых долей секунды). Архитектура, основанная на командах и использованная вами в предыдущем разделе для запуска параллельных циклов, может быть модифицирована для передачи сообщений по сети.

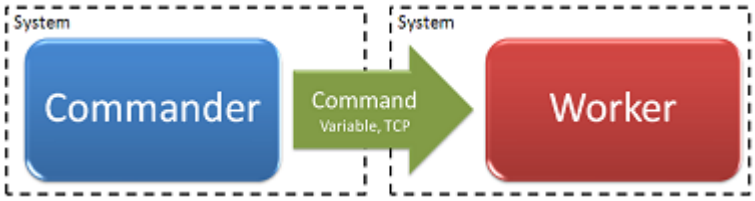


Рисунок 4.1. Вы можете использовать различные технологии для передачи команд по Ethernet, включая переменные общего доступа и TCP/IP
Commander - Начальник, Worker - Исполнитель, Command – команда, System - система

Способ обмена, основанный на сообщениях, подходят также для передачи больших наборов данных фиксированного размера, например, результатов теста, результатов тысяч измерении, выполненных в приложении контроля вибрацией, или кадра изображения. Большинство приложений, где требуется передача больших объемов данных, могут быть разделены на периодические задачи, где наборы данных фиксированного размера передаются с помощью сообщений.

Способы обмена данными по ходу процесса

Передача данных по ходу процесса обычно состоит из текущих значений, периодически пересылаемых между контроллерами или от контроллера - HMI. Необязательно гарантировать каждую передачу данных, поскольку контроллер или HMI всегда интересует последнее значение, а не буферизированное. Пример – данные, передаваемые в HMI для отображения позиции детали, двигающейся по конвейерной ленте.

В то время как управляющее приложение может требовать более высоких скоростей, приложения HMI, как правило, имеют относительно низкую скорость обновления данных. Поскольку данные видят только люди, обычно достаточно частоты обновления от 1 до 30 Гц. Более высокие скорости только расходуют полосу пропускания и ресурсы процессора. Цифровые индикаторы становится сложно наблюдать уже при частоте более 1-2 Гц, аналогично тому, как различать тысячные доли чисел, когда вы заправляете свой автомобиль. Обновления графических индикаторов с частотой 30 Гц "сглаживают" изображения, и это находится уже за пределами человеческого восприятия.

Потоковые/буферизированные коммуникации

Обмен буферизированными данными выполняется непрерывно, но необязательно в реальном времени. Это полезно для сигнальных данных, когда вам необходимо захватить каждую точку данных. Например, представьте передачу данных о вибрациях от непрерывно работающего приложения в системе мониторинга состояния машины, когда все данные сохраняются на сетевой жесткий диск для последующего анализа. Поскольку объем данных не фиксирован, для их передачи сложно использовать способ коммуникации, основанный на сообщениях. Вам нужно открыть соединение и постоянно буферизировать данные, чтобы не потерять их. Потоковая передача выходит за рамки основ управления механизмами и не рассматривается подробно в данном документе.

Способы обмена данными	**Характеристики**	**Требования**
Основанные на сообщениях	Управляемый событиями, команды	Низкое время ожидания, гарантированная доставка
Данные по ходу процесса	Одноточечные, текущие значения	Последнее значение вместо гарантированной доставки
Потоковые/буферизированные	Непрерывная передача данных	Высокая пропускная способность, гарантированная

		доставка

Таблица 4.1. Обобщенная характеристика способов обмена информацией в системах управления механизмами

Обмен данными с использованием публикуемых в сети переменных общего доступа

До сих пор вы рассматривали переменные общего доступа типа Single Process, использующие табличную память для детерминированной передачи данных между циклами или передачи команд в пределах одного контроллера. Передавать данные по Ethernet вы можете, выбрав другой тип переменной общего доступа – публикуемую в сети переменную общего доступа (network-published shared variable). Использовать эту переменную можно для обмена данными по ходу процесса (совместное использование данных) и для обмена данными с использованием сообщений (передачи команд) по сети Ethernet между контроллерами, HMI и ПК, работающих под LabVIEW.

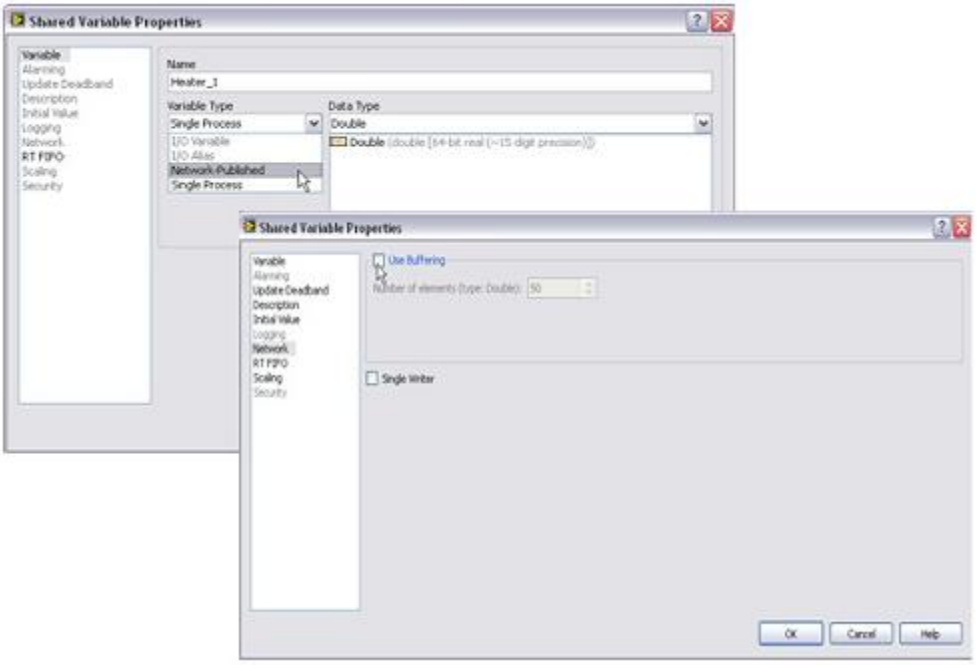


Рисунок 4.2. Для передачи данных по Ethernet можно использовать публикуемые в сети переменные общего доступа

Основные сведения о публикуемых в сети переменных общего доступа

Термин "*сетевая переменная*" относится к программному элементу сети, который может передавать данные между программами, приложениями, удаленными компьютерами и аппаратными средствами.

Три важных компонента обеспечивают работу сетевой переменной в LabVIEW: узлы сетевой переменной (network variable nodes), механизм продвижения переменных общего доступа (Shared Variable Engine) и протокол NI Publish-Subscribe (опубликования/ подписки).

Узлы сетевой переменной

Вы можете использовать узлы переменной на блок-диаграмме для выполнения операций чтения и записи. Каждый узел переменной считается ссылкой на программный элемент в сети (актуальную сетевую переменную), обслуживаемый механизмом переменных общего доступа. На рисунке 4.3 показана переменная общего доступа, ее сетевой путь и соответствующий ей элемент в дереве проекта.

Рисунок 4.3. Узел сетевой переменной и соответствующий элемент проекта

Механизм обслуживания переменных общего доступа

Механизм продвижения переменных общего доступа – программный компонент, обслуживающий данные, публикуемые через Ethernet. Этот механизм может работать на устройствах реального времени или на ПК под управлением Window. В Windows механизм продвижения переменных общего доступа – служба, запускаемая при старте системы.

В устройствах реального времени механизм продвижения переменных общего доступа – инсталлируемый запускаемый компонент, который загружается при загрузке системы.

Для использования сетевых переменных механизм продвижения переменных общего доступа должен работать хотя бы на одной системе в сети. Любое устройство LabVIEW в сети может читать или записывать в сетевые переменные, публикуемые механизмом продвижения переменных общего доступа. На рисунке 4.4 приведен пример распределенной системы, в которой механизм продвижения переменных общего доступа работает на настольной машине, а множество контроллеров реального времени обмениваются данными через сетевую переменную.

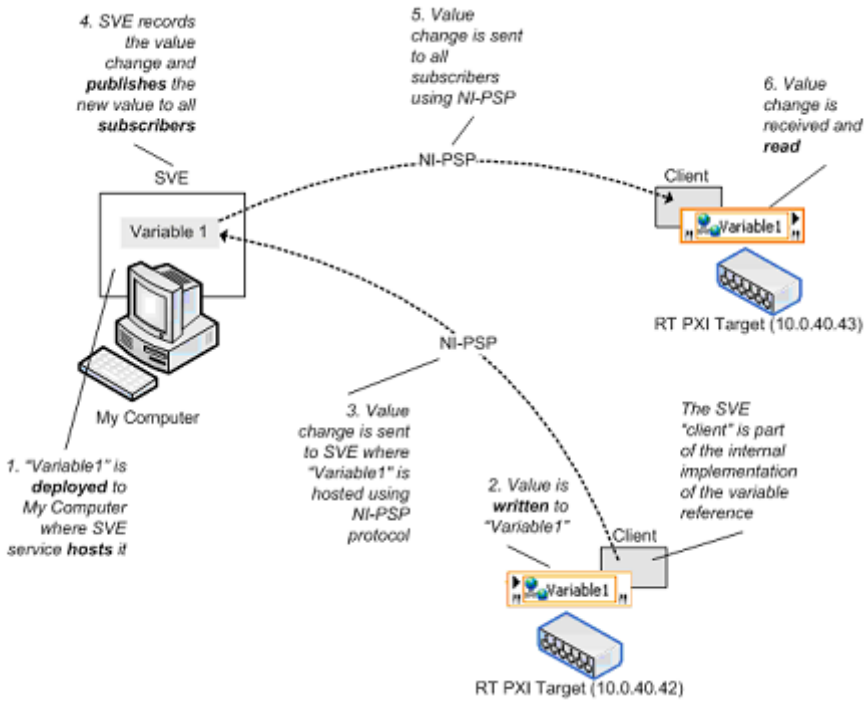


Рисунок 4.4. Распределенные системы для обмена данными используют публикуемые в сети переменные общего доступа

Variable 1 – переменная 1, My Computer – мой компьютер, RT PXI Target – целевое PXI устройство реального времени, Client – клиент SVE.

1. Переменная 1 развертывается на "моем компьютере", где ее обслуживает SVE.
2. Значение записывается в " Variable 1".
3. Изменение значения посылается в SVE, где " Variable 1" обслуживается с использованием протокола PSP.
4. SVE регистрирует измененную переменную и публикует новое значение всем подписчикам.
5. Измененная переменная посылается всем подписчикам по протоколу NI-PSP.
6. Измененное значение получено и прочитано.

Протокол PSP (публикации – подписки)

Механизм продвижения переменных общего доступа использует протокол NI Publish-Subscribe Protocol (NI-PSP) для обмена данными. NI-PSP - сетевой протокол на основе TCP, оптимизированный для надежного обмена множеством элементов данных по сети Ethernet. Для минимизации загрузки полосы пропускания Ethernet каждый клиент подписывается на индивидуальные элементы данных. Далее протокол реализует процедуру обмена, управляемую событиями, когда данные передаются подписавшимся клиентам только при их изменении. Протокол также собирает множество сообщений в один пакет для минимизации ресурсов, расходуемых при передаче служебных данных по Ethernet. Кроме того, он предоставляет тактовые импульсы для обнаружения потерянных соединений и обладает возможностью автоматического переподключения, если в сеть добавляются устройства. Сетевой протокол NI-PSP использует URL-адрес для передачи данных по сети.

Свойства публикуемых в сети переменных общего доступа

Буферизация

Вы можете использовать буферизацию для реализации способа обмена данными, основанного на сообщениях. Буферизация не должна использоваться при обмена данными в ходе процесса.

Когда вы конфигурируете сетевой буфер для публикуемой в сети переменной общего доступа, на самом деле вы настраиваете размеры двух разных буферов, на стороне сервера и на стороне клиента. Буфер на стороне сервера, обозначенный как буфер внутри прямоугольника под названием SVE на рисунке 4.5, автоматически создается и настраивается на такой же размер, как буфер на стороне клиента. Буфер на стороне клиента (на правой стороне рисунка 4.5) отвечает за поддержание очереди предыдущих значений. Каждый читатель публикуемой в сети переменной общего доступа получает свой собственный буфер, так что читатели не взаимодействуют друг с другом.

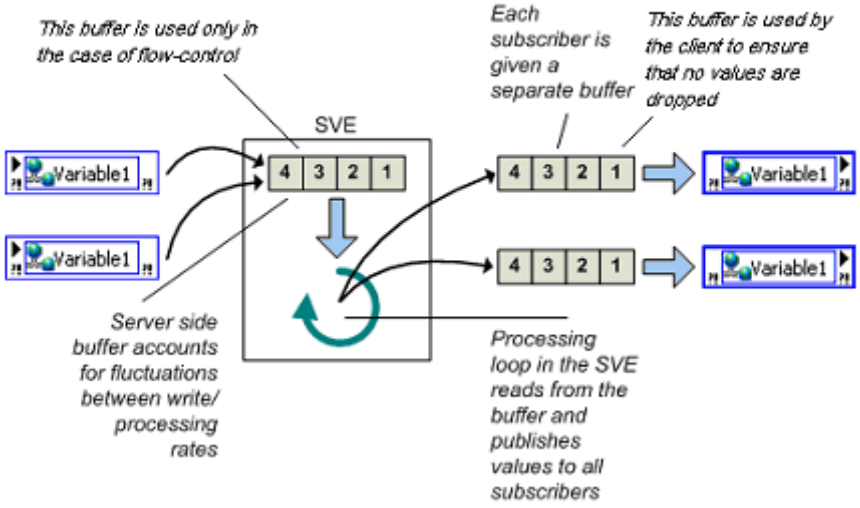


Рисунок 4.5 Буферизированные сетевые переменные

Буфер в SVE используется только для управления потоком данных.

Буфер на стороне сервера согласовывает флуктуации между скоростями записи/обработки данных.

Цикл обработки в SVE считывает данные из буфера и публикует переменные всем подписчикам.

Каждому из подписчиков выделяется отдельный буфер. Этот буфер используется клиентом, чтобы убедиться, что ни одно значение не потеряно.

Поскольку буферизация выделяет буфер каждому подписчику, во избежание лишнего использования памяти, применяйте буферизацию только при необходимости и ограничивайте размер буферов.

Детерминизм

Публикуемые в сети переменные общего доступа весьма гибки и конфигурируемы. Вы можете создать подобную переменную, обладающую буфером FIFO реального времени, для объединения возможностей переменных общего доступа, публикуемых в сети (Network published) и применимых внутри одного процесса (Single Process). Когда вы это сделаете, LabVIEW автоматически запустит фоновый цикл для копирования сетевых данных в FIFO реального времени.

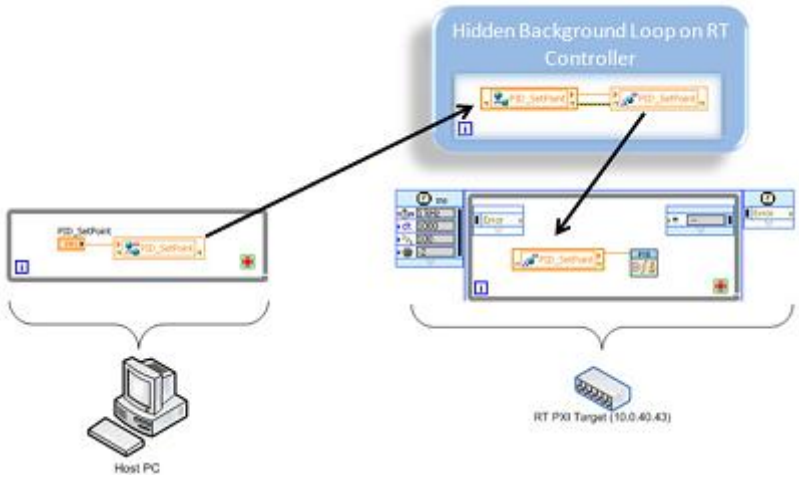


Рисунок 4.6. Когда вы включаете FIFO реального времени для публикуемых в сети переменных общего доступа, в устройстве реального времени запускается скрытый фоновый цикл для копирования значений из сети в FIFO реального времени

Host PC – хост-компьютер, RT PXI Target – устройство PXI реального времени, Hidden Background Loop on RT Controller – скрытый фоновый цикл в контроллере реального времени

Эта свойство может облегчить вам программирование, однако у нее есть и некоторые ограничения:

- Некоторые возможности публикуемых в сети переменных общего доступа будут недоступны
- Управление ошибками становится более сложным, поскольку ошибки в сети распространяются программой на индивидуальные узлы
- Усложняется дальнейшее изменение программы для использования других сетевых коммуникаций.

Это расширенная возможность, и, ради понятности изложения, в этом документе она не рассматривается. Вместо этого для обмена данными вы будете использовать только публикуемые в сети переменные общего доступа и продолжите использовать переменные общего доступа типа Single Process, которые предоставляют табличную память с детерминированными обменом данными между циклами в одном контроллере. В этом разделе объясняется, как создать цикл для копирования данных из публикуемой в сети переменной общего доступа в переменную общего доступа типа Single Process.

Время существования

Все переменные общего доступа являются частью библиотеки проекта. По умолчанию механизм продвижения переменных общего доступа развертывает и публикует всю библиотеку переменных общего доступа, как только вы запускаете VI, ссылающийся на любую из переменных. Останов VI не удаляет переменную из сети. Кроме того, если вы перезапустите компьютер, на котором работает механизм продвижения переменных общего доступа, эти переменные будут снова доступны в сети, как только машина закончит загрузку. Если вам нужно удалить переменную общего доступа из сети, вы должны напрямую остановить продвижение (undeploy) переменной или библиотеки из окна обозревателя проекта (Project Explorer) или менеджера распределенных систем (NI Distributed System Manager).

Свойства SCADA

Модуль LabVIEW Datalogging and Supervisory Control (DSC) дополняет основные функции публикуемых в сети переменных общего доступа набором функций SCADA, включая следующие:

- Запись истории в базу данных Citadel
- Регистрация тревог и событий
- Масштабирование
- Определяемые пользователем настройки безопасности
- Создание пользовательских серверов ввода-вывода

Публикуемые в сети переменные и псевдонимы ввода-вывода

По умолчанию переменные ввода-вывода и псевдонимы ввода-вывода публикуются в сети для удаленного мониторинга ввода-вывода через протокол PSP. Они публикуются потоком с нормальным приоритетом, который связан с механизмом сканирования NI Scan, и с частотой, заданной вами в свойствах контроллера. Вы можете настроить, какие переменные ввода-вывода будут публиковать свои состояния в диалоге **Shared Variable Properties** (Свойства переменных общего доступа).

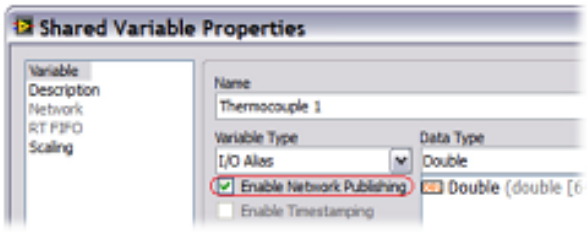


Рисунок 4.7. Разрешение публикации в сети переменной ввода-вывода

Публикуемые переменные ввода-вывода оптимизированы для мониторинга ввода-вывода и не поддерживают все возможности публикуемых в сети переменных общего доступа, а также не поддерживаются всеми устройствами LabVIEW. Для максимальной гибкости совместного использования данных приложениями LabVIEW следует использовать публикуемые в сети переменные общего доступа.

Хостинг и мониторинг публикуемых в сети переменных общего доступа

Хостинг

Для использования публикуемых в сети переменных общего доступа, механизм продвижения переменных (Shared Variable Engine) общего доступа должен быть запущен по крайней мере на одном из узлов распределенной системы. Любой узел в сети может считывать или записывать в переменные общего доступа, публикуемые Shared Variable Engine. Все узлы могут обращаться к переменной без инсталляции Shared Variable Engine. В случае контроллеров реального времени, требуется небольшой устанавливаемый клиентский компонент для ссылок на переменные, обслуживаемые в других системах.

У вас может быть несколько систем, одновременно запускающих Shared Variable Engine и которые позволяют приложениям, если потребуется, развертывать переменные общего доступа из разных мест.

Чтобы решить, на каком из вычислительных устройств (устройствах) в распределенной системе развертывать и обслуживать публикуемые в сети переменные общего доступа, вы должны учитывать следующие факторы.

Совместимость Shared Variable Engine

Некоторые компьютерные устройства в вашей распределенной системе могут не поддерживать хостинг Shared Variable Engine, в том числе Macintosh, Linux и Windows CE. Обратитесь к разделу "NI-PSP Networking Technology" в справке LabVIEW Help, в котором приведен перечень совместимых систем и платформ.

Доступные ресурсы

Хостинг большого количества сетевых переменных может занимать значительные ресурсы системы, так что для больших распределенных приложений NI рекомендует выделить отдельную систему для Shared Variable Engine.

Требуемые возможности

Если в приложении требуется функционал DSC, тогда эти переменные должны размещаться на машине под управлением Windows, на которой запущен Shared Variable Engine.

Доступность

Некоторые переменные главного процесса могут быть критическими для функционирования распределенного приложения, так что разумно запустить их на машине с надежной встроенной ОС, наподобие LabVIEW Real-Time для повышения общей надежности системы.

Динамический доступ к переменным

К сетевой переменной можно также обратиться по имени пути, это позволяет создавать программы с динамическим выбором переменных для чтения или записи.

Имя пути подобно имени совместно используемого ресурса в сети Windows, например, \\machine\myprocess\item. В этом примере machine - имя компьютера, IP-адрес, или полностью определенное доменное имя сервера, где обслуживается переменная; myprocess содержит папки сетевых переменных или сами переменные и относится к процессу сетевой переменной; item - это имя сетевой переменной. Ниже приведены дополнительные примеры ссылок на сетевые переменные

\\localhost\my_process\my_variable

\\test_machine\my_process\my_folder\my_variable

\\192.168.1.100\my_process\my_variable

Мониторирование переменных

Менеджер распределенных систем NI предлагает централизацию мониторинга систем в сети и управления опубликованными данными.

Из менеджера систем вы можете обращаться к публикуемым в сети переменным общего доступа, не нуждаясь в среде проектирования LabVIEW.

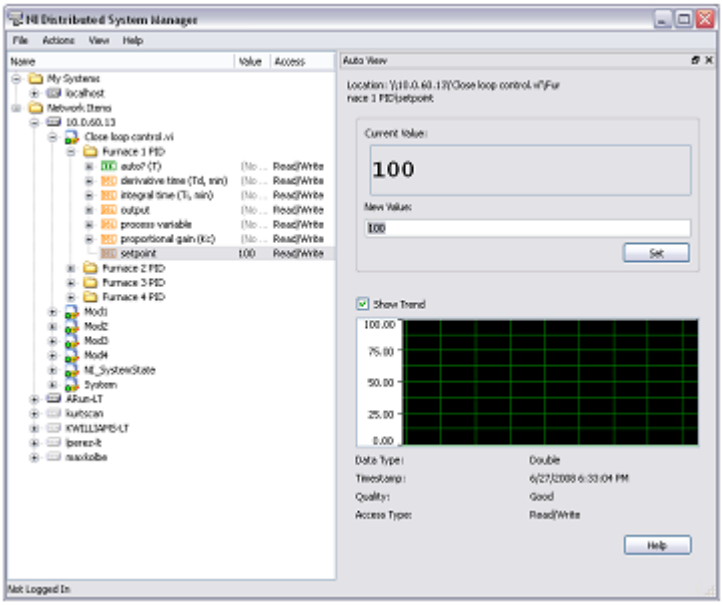


Рисунок 4.8. Менеджер распределенных систем NI (NI Distributed System Manager)

При помощи менеджера распределенных систем вы можете записывать в публикуемые в сети переменные общего доступа, чтобы удаленно регулировать настройки процесса без особой необходимости специального HMI. Вы можете также использовать NI Distributed System Manager для наблюдения и управления за сбоями контроллера и системными ресурсами на устройствах реального времени.

Выберите в меню LabVIEW **Tools»Distributed System Manager** для запуска NI Distributed System Manager.

Использование публикуемых в сети переменных общего доступа для совместного использования данных процесса

В данном разделе приведен пример кода LabVIEW

Вы легко можете использовать сетевые переменные для публикации переменных в ходе процесса. В примере ПИД-регулятора из раздела 2 этого документа вы жестко задавали уставку ПИД. Теперь изменим этот пример так, чтобы вы могли непрерывно обновлять уставку с другого устройства LabVIEW в сети.

Сначала создайте публикуемую в сети переменную общего доступа под названием

SV_PID_SetPoint. Перейдите на закладку Network (Сеть). По умолчанию буферизация разрешена, так что отключите ее (OFF). Сохраните переменную в библиотеке коммуникаций. Когда вы развернете код в системе CompactRIO, то автоматически развернете и эту библиотеку, и переменная станет доступна в сети.

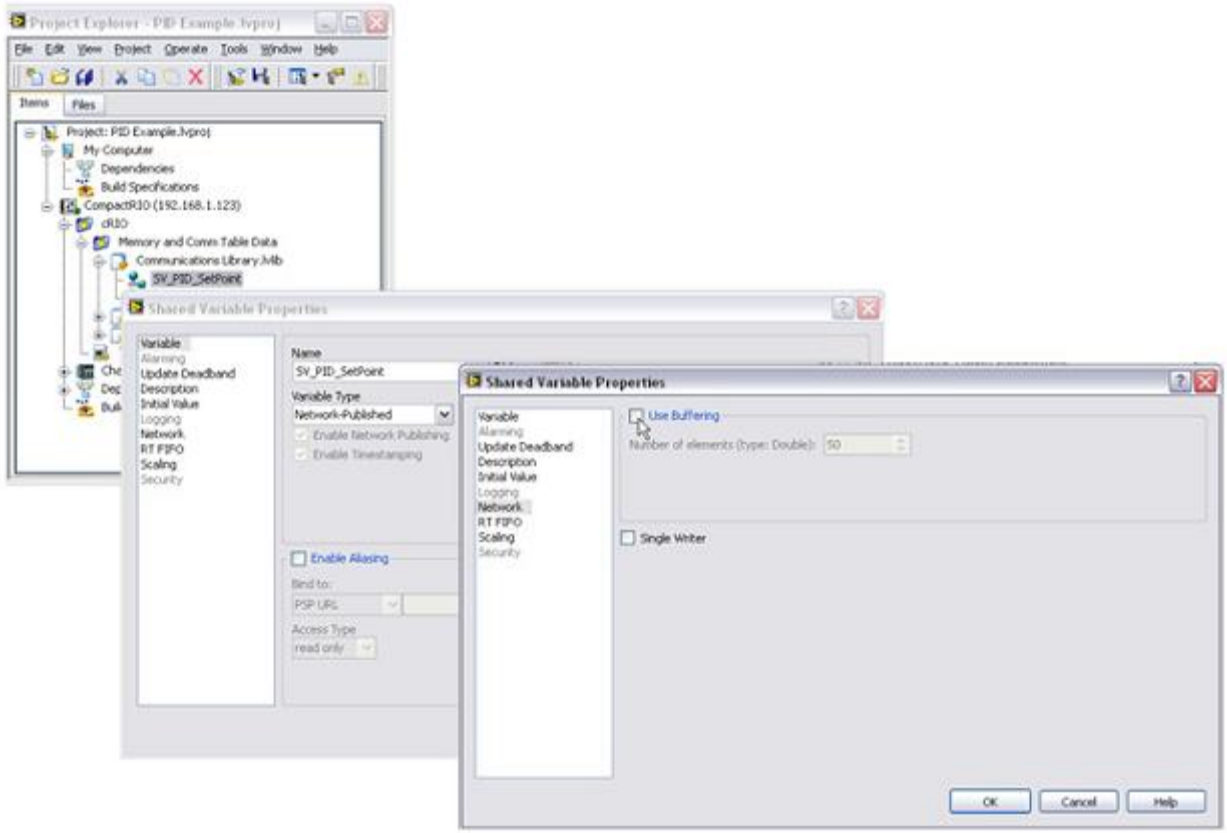


Рисунок 4.9. Публикуемые в сети переменные общего доступа позволяют обмениваться данными между узлами LabVIEW по сети Ethernet. Убедитесь, что отключили сетевую буферизацию

Поскольку сетевые коммуникации через Ethernet не являются детерминированными, считывайте SV_PID_SetPoint в цикле, параллельном контуру регулирования. Добавьте также в вашу таблицу данных элемент для передачи данных из цикла коммуникаций в управляющий цикл. Создайте переменную под названием PID_SetPoint типа Single Process с FIFO реального времени.

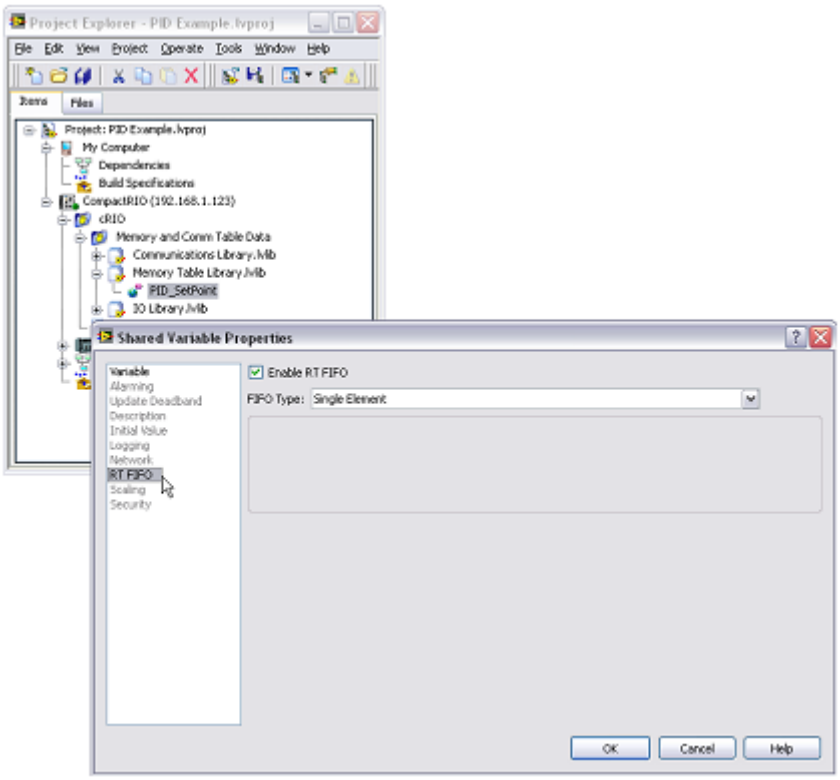


Рисунок 4.10. Вы можете использовать переменную типа Single Process с FIFO реального времени для детерминированного обмена данными между циклами

Теперь модифицируйте блок-диаграмму, добавив цикл с нормальным приоритетом, в котором считываются значения SV_PID_SetPoint, проверяются на наличие ошибок или предупреждений и записывают в PID_SetPoint. Запустите этот цикл с частотой 100 Гц.

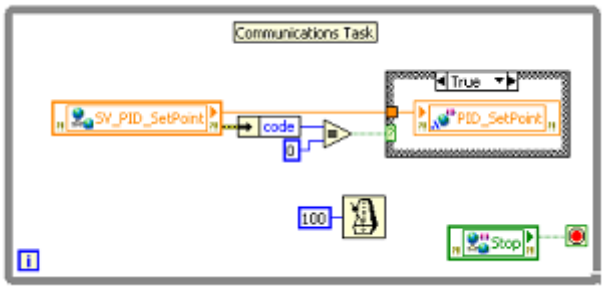


Рисунок 4.11. Цикл коммуникаций передает данные из сети в локальную табличную память, если нет ошибок или предупреждений

Наконец, добавьте блок ПИД, чтобы он считывал значение из PID_SetPoint, а в подпрограмме инициализации заносил в переменную PID_SetPoint значение по умолчанию.

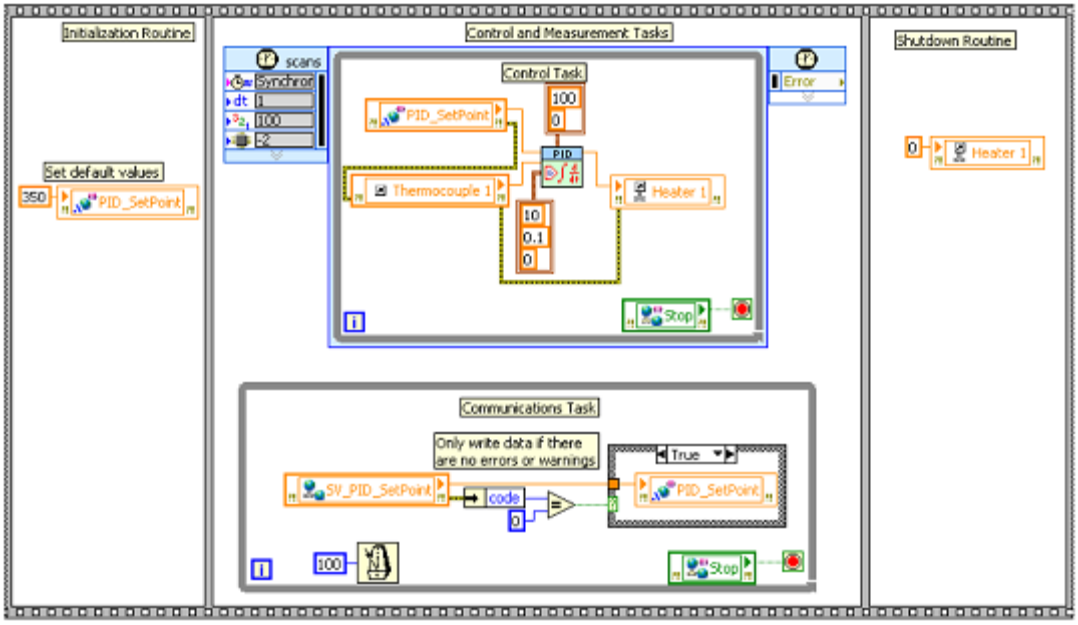


Рисунок 4.12. Завершенное приложение с сетевой коммуникацией

Теперь вы можете непрерывно обновлять уставку с любого узла LabVIEW в сети.

Использование публикуемых в сети переменных общего доступа для передачи команд

В предыдущем разделе было описано, как создать основанную на командах архитектуру для выполнения действий, таких, как запуск задач. Вы можете использовать ту же концепцию для генерации команд из других устройств в сети. Сетевые технологии позволяют распределенным системам публиковать своим статусы и координировать свои действия. В случае командной архитектуры вы можете использовать сетевые технологии для коммуникации между системами.

Рисунок 4.13. Простая командная архитектура с использованием сетевых переменных
Commander - Начальник, Worker - Исполнитель, Command – команда, Network Variable –сетевая переменная

Модель публикации-подписки сетевых переменных упрощает и реализацию систем с несколькими Начальниками.

Рисунок 4.14. Система с несколькими Начальниками и одним Исполнителем
Commander - Начальник, Worker - Исполнитель

Публикуемые в сети переменные общего доступа предлагают практичный способ создания канала связи для пересылки командных сообщений по сети. Для построения масштабируемого приложения вы должны разработать вашу систему- Исполнитель таким образом, чтобы она имела единственную задачу анализатора команд, которую вы можете использовать для интерпретации и перераспределения команд при необходимости. Это обеспечит, что выполняющиеся в настоящее время критические задачи не будет нарушать прием команды, а также упростит модификацию кода для обработки дополнительных команд.

Чтобы отправить данные, используйте одну публикуемую в сети переменную общего доступа с разрешенной буферизацией. Без буферизации следующие друг за другом команды переписывают друг друга, что ведет к потере команд. Буферизация критична также при поддержке нескольких Начальников, поскольку позволяет обслуживать команды по порядку.

Публикуемая в сети переменная общего доступа предлагает механизмы буферизации на каждом уровне реализации. Механизм продвижения переменных общего доступа помещает значения в буфер и передает их всем подписчикам, где каждый экземпляр узла переменной получает собственный буфер.

Для использования одной переменной для множества команд создайте переменную с типом данных U32.

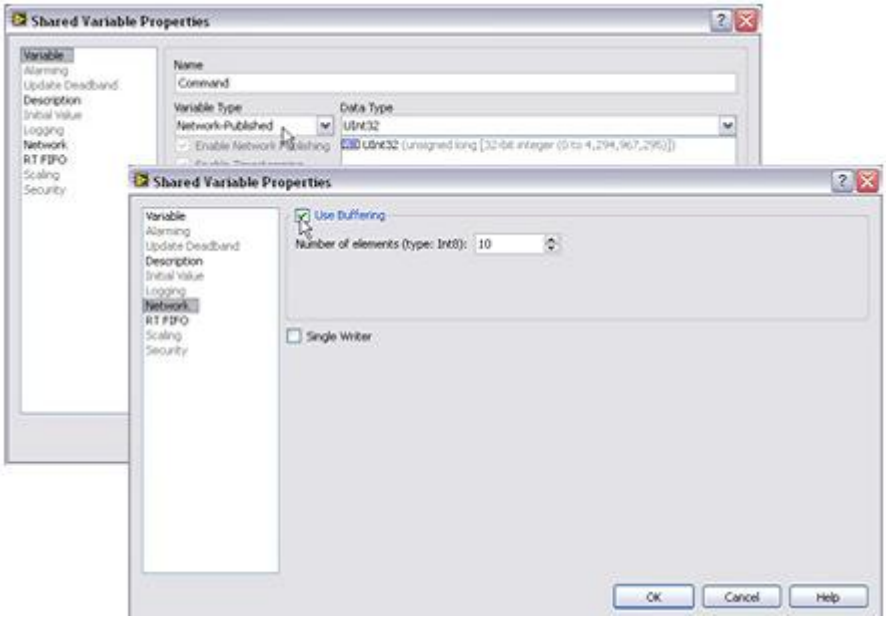


Рисунок 4.15. Для сетевых команд создайте публикуемую в сети переменную общего доступа с разрешенной буферизацией

Для обеспечения масштабируемости вы должны иметь возможность добавления в систему новых команд без необходимости серьезной перестройки приложения или причинения вреда производительности. Чтобы упростить сопровождение кода, создайте определение типа enum (U32) с записью для каждой команды. Определение перечислительного (enum) типа в форме LabVIEW - эффективный способ определения команд, которые могут быть напрямую переданы в сетевые переменные.

Рисунок 4.16. Передача команды через сетевую переменную

Определение типа enum предоставляет упорядоченный список доступных команд. Если вы сделаете определение типа, изменения автоматически распространятся в коде.

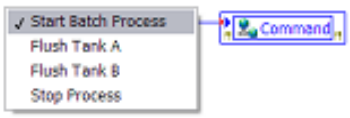


Рисунок 4.17. Список перечисленных команд

Архитектура Начальника

Начальник – любой источник команды, на которую должен реагировать Исполнитель. Типичный начальник – обработчик событий пользовательского интерфейса, являющегося частью HMI, который воспринимает события пользовательского интерфейса и транслирует их в команды для контроллера. В случае сетевой архитектуры Начальник заботится о распределении этих команд, а также о выполнении локальных действий, например, временно запрещать восприятие элементов пользовательского интерфейса и формирование сообщений о статусе.

Вы легко можете реализовать архитектуру Начальника для событий, которые происходят в интерфейсе пользователя, используя стандартные шаблоны LabVIEW по обработке событий пользовательского интерфейса. Просто транслируйте событие пользовательского интерфейса в подходящую команду, создав командное сообщение и записав его в сетевую переменную.

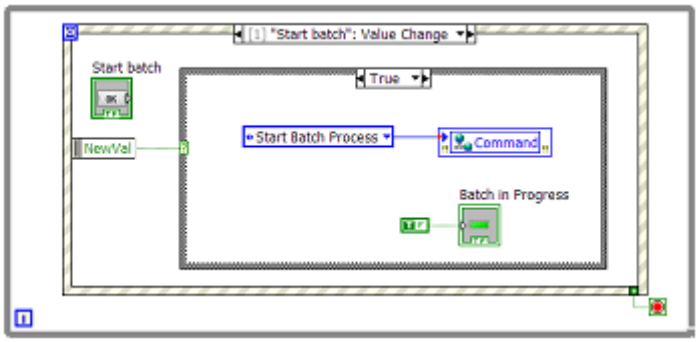


Рисунок 4.18. Простой VI-Начальник, управляемый событиями пользовательского интерфейса

Обратитесь к разделам, посвященным созданию человеко-машинного интерфейса для получения дополнительной информации о надлежащей архитектуре построения HMI.

Архитектура Исполнителя

В системе Исполнителя в задаче анализатора команд вы можете читать публикуемые в сети переменные общего доступа для проверки на новые значения. Эта ситуация требует, чтобы входящая команда была интерпретирована и применена без влияния на детерминизм контура регулирования. Кроме того, обслуживание команд происходит не на той же скорости, что обслуживание управляющего алгоритма. Поэтому используйте отдельную задачу для анализа входящих команд и распределения необходимой информации и событий по всему приложению. Назовем это задачей Анализатора команд (*Command Parser task*).

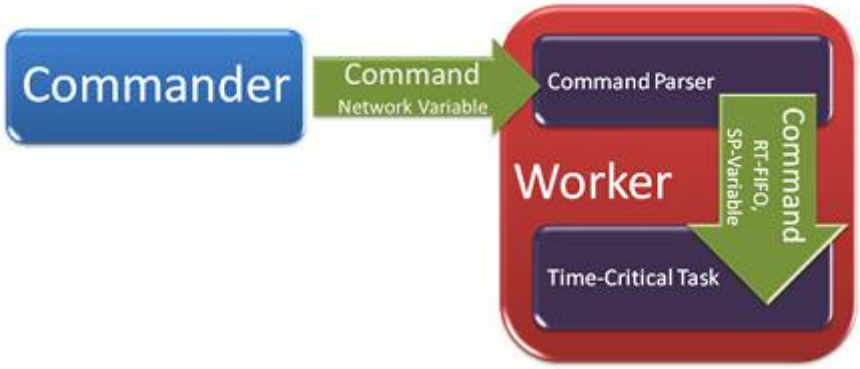


Рисунок 4.19. Централизованная архитектура анализатора команд
Commander - Начальник, Worker - Исполнитель, Command – команда, Network Variable –сетевая переменная, Command Parser – Анализатор команд, Time-Critical Task – критическая ко времени задача

Для реализации задачи Анализатора команд просто добавьте параллельный процесс для приема и обработки команд. Отдельный цикл анализатора команд обеспечивает масштабируемость, потому что:

- Позволяет приложению ограничить доступ к некоторым переменным процесса, основываясь на происхождении команды, диапазона значении параметра и текущего состояния приложения
- Позволяет проводить дополнительную обработку и реинтерпретацию принимаемых команд, прежде чем они будут перенаправлены на соответствующие задачи
- Позволяет сложным командам упаковывать параметры, формируя связанный набор данных для применения в качестве части команды.
- Упрощает модификацию программы для обработки различных сетевых типов, например, необработанных сообщений TCP.
- Позволяет командную архитектуру с квитированием, когда команда и обработка квитанций выполняются в одной задаче.

Построение архитектуры Исполнителя

Чтение тайм-аутов

Для уменьшения загрузки процессора (CPU) вы можете использовать блокировку чтения публикуемых в сети переменных общего доступа, задав значение тайм-аута (в миллисекундах). Вы можете разрешить блокировку на каждом узле, выбрав Show Timeout (Показать тайм-аут) из контекстного меню узла переменной.

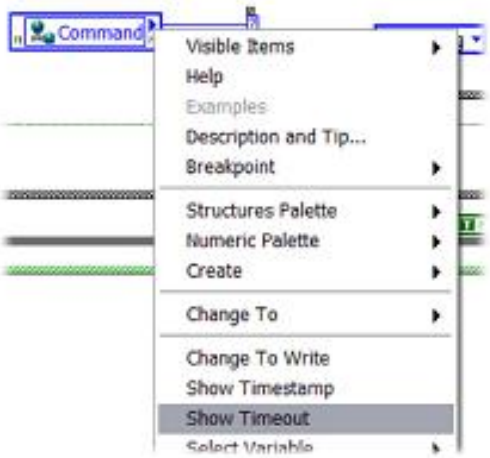


Рисунок 4.20. Разрешение функционала блокировки чтения из узла сетевой переменной

Вы можете подключить значение тайм-аута ко входу тайм-аута переменной для задания времени ожидания нового значения узлом переменной. Если новых значений нет в буфере, и они не появились до завершения тайм-аута, узел переменной возвращает последнее известное значение переменной и устанавливает булевский терминал в положение TRUE. После чего вы можете обработать тайм-аут соответствующим образом.

Обработка ошибок

На данной стадии вы должны рассмотреть поведение приложения при появлении ошибки. Узел сетевой переменной может возвращать предупреждения или ошибки для сигнализации о состоянии своих основных критических составляющих, среди которых механизм продвижения переменных общего доступа, соответствующие процессы хостинга переменных и состояние сети. Вот наиболее важные факторы, которые нужно учитывать для защиты от ошибок на принимающей стороне командной архитектуры:

A. При возвращении ошибки сетевая переменная возвращает также и значение по умолчанию, соответствующее типу данных. В случае булевской переменной это FALSE, для числового типа – 0, для перечислительного типа - первый элемент.

- Вы хотите, чтобы приложение подавляло возвращаемое значение при появлении ошибки.

B. Как только установлено соединение, создается буфер для каждого узла переменой и текущее значение переменной передается в него первым элементом. Это создает проблему для подхода, основанного на командах, потому что вы не можете сказать точно, может ли Исполнитель опираться на текущее командное значение, или же оно просто осталось в буфере с последнего раза выполнения приложения.

- Считывая значение переменной, все читатели должны очищать буфер переменной для гарантии известного начального состояния, пока переменная не сообщит, что ее буфер пуст, вернув тайм-аут.

Эти требования специфичны для архитектуры, основанной на командах, при работе с сетевой переменной; вы можете выполнить их, изолируя узел переменной специальными компонентами, которые обслужат приведенных выше соображениях, в то же время скрыв некоторые детали реализации канала коммуникаций. Вы можете использовать следующий фрагмент кода, извлеченный из шаблона Command Reader, для реализация вашей собственной архитектуры, основанной на командах.

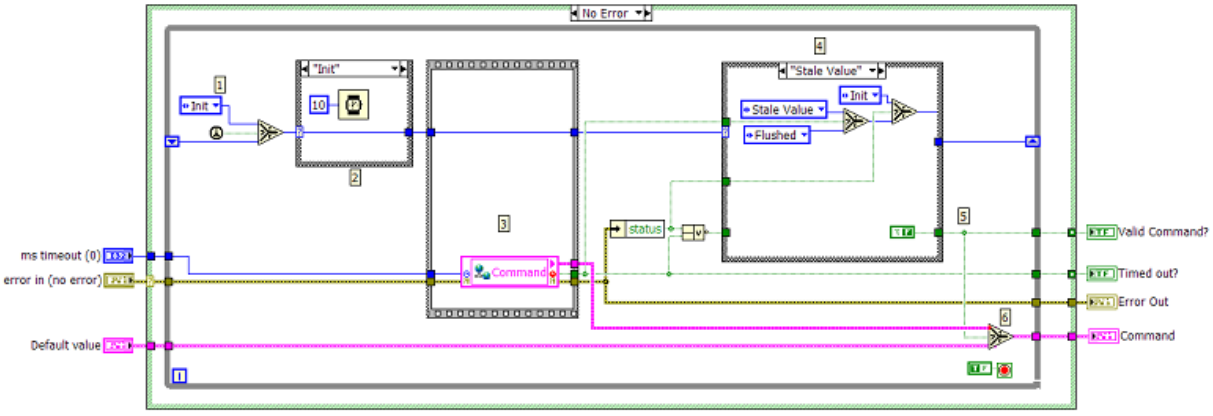


Рисунок 4.21. VI – шаблон читателя команд (Command Reader)

- Код обслуживает проблему A, возвращая при появлении ошибки заданное значение по умолчанию
- Код обслуживает проблему B, очищая буфер переменной путем чтения его до полного опустошения, и затем ожидая команды.

Пример архитектуры, основанной на командах, с использованием публикуемых в сети переменных общего доступа

В данном разделе приведен пример кода LabVIEW

Теперь изменим пример ПИД-регулятора с коммуникациями для реализации сетевой команды. Модифицируем пример анализатором команд для останова каждого цикла.

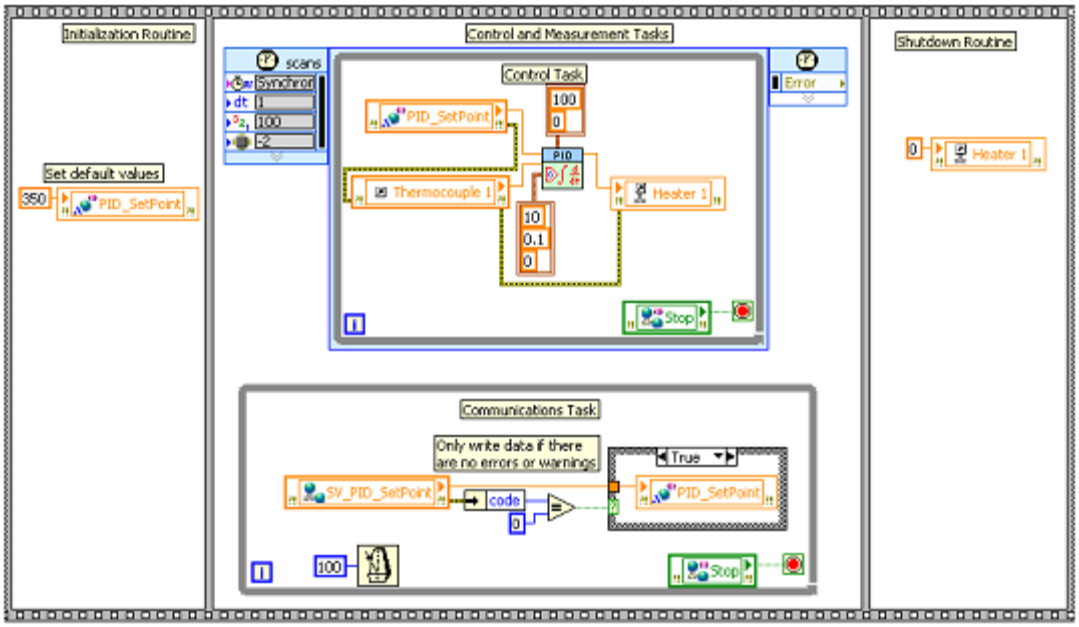


Рисунок 4.22. В этом примере добавим задачу анализатора команд

Сначала создадим определение типа enum для команды. Поскольку приведенный пример очень простой, у вас есть только одна команда "Стоп".

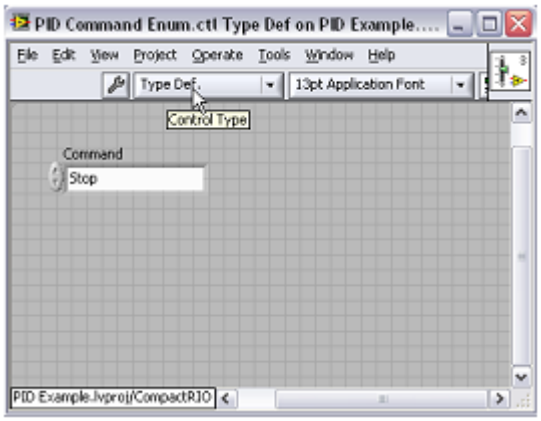


Рисунок 4.23. Определение типа enum обеспечивает масштабируемость

Далее создадим публикуемую в сети переменную общего доступа под названием “Command” с разрешением буферизации.

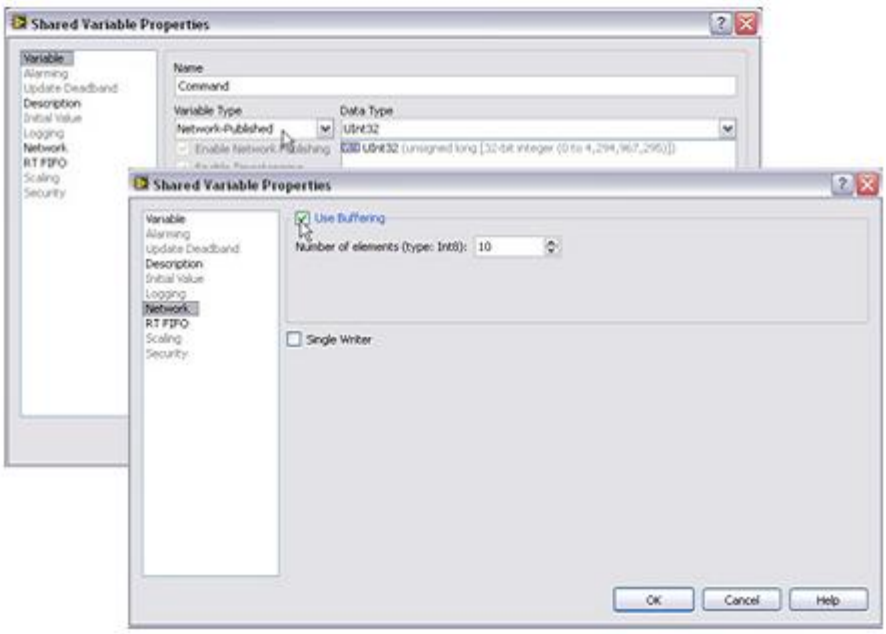


Рисунок 4.24. Для сетевых команд создадим публикуемую в сети переменную общего доступа с разрешенной буферизацией.

Далее изменим VI Command Reader, заменим вход "Default Value" и выход "Command" на тип enum. Изменим также переменную общего доступа так, чтобы она считывала переменную общего доступа "Command".

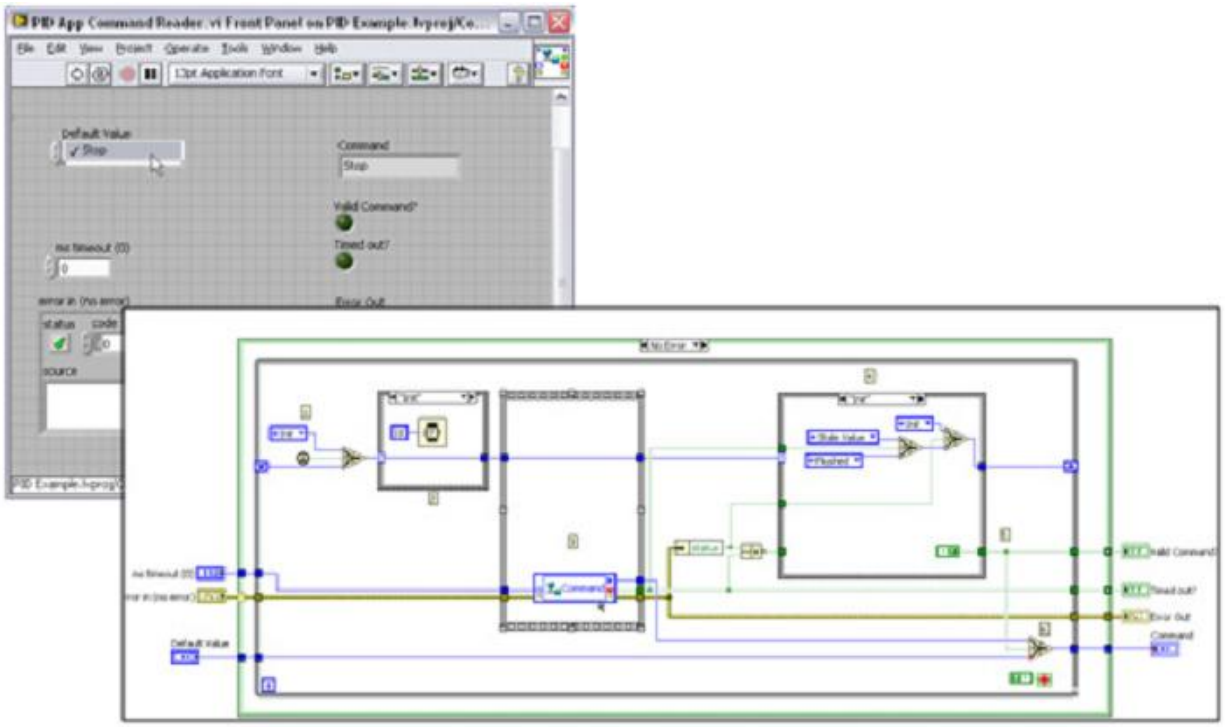


Рисунок 4.25. При первом вызове VI Command Reader очищает сетевую очередь

Теперь создадим переменные общего доступа типа Single Process, которые будут служить командами для выхода из каждого цикла.

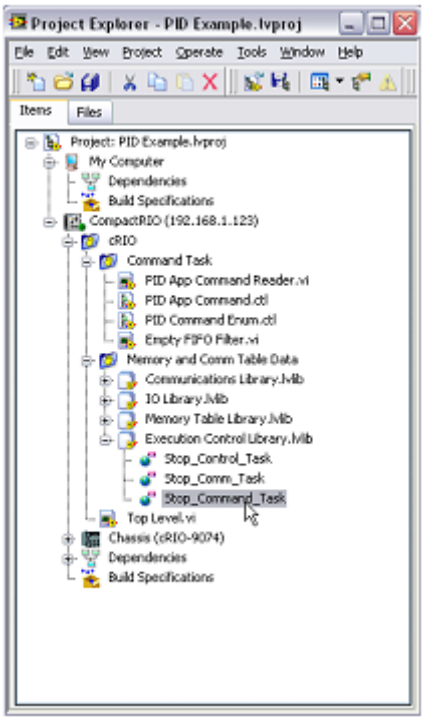


Рисунок 4.26. Команды передаются внутри контроллера с использованием переменных общего доступа типа Single Process с мультиэлементным FIFO реального времени, как было показано ранее

Наконец, добавим еще один цикл для анализа команд. Этот цикл подключен к структуре Case, которая выполняет логику, если команда верна. Если получена верная команда, вторая структура Case запускает логику для каждой команды. В этом простом примере у вас только одна команда – "Стоп". Эта команда, используя архитектуру для обработки внутренних команд, рассмотренную в разделе 3, заставляет каждый из трех циклов остановиться.

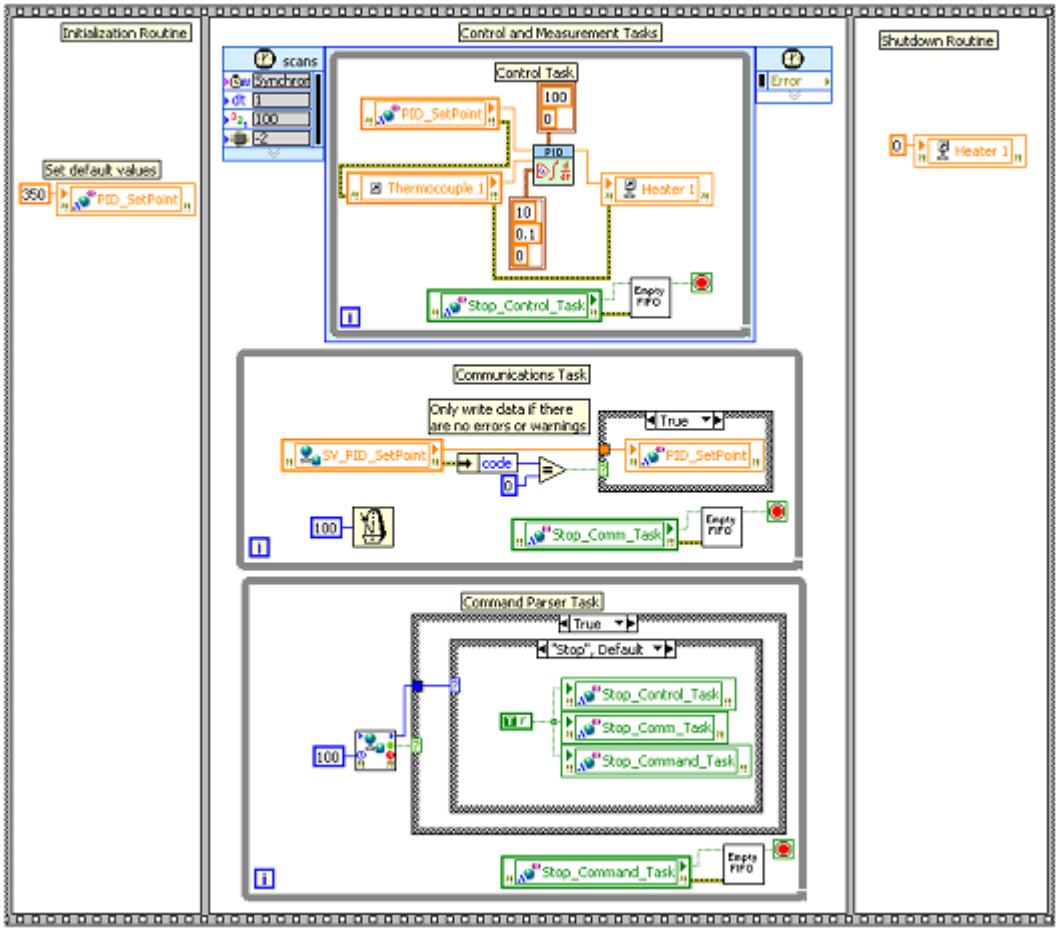


Рисунок 4.27. Завершенная блок-диаграмма с задачей анализатора команд и командами для останова каждого цикла

Улучшенные архитектуры, основанные на командах

В данном разделе приведен пример кода LabVIEW

Архитектура, основанная на командах - встраивание данных и команд

Вы можете расширить команды для включения общих параметров, так что вместе с командой может передаваться дополнительная информация. Это помогает создавать более сложные команды и обеспечивает согласованность данных. Вы можете задать тип данных для публикуемой в сети переменной общего доступа, что помогает вам легко расширить содержимое командного сообщения для включения некоторых значений, которые можно интерпретировать как параметры для заданной команды.

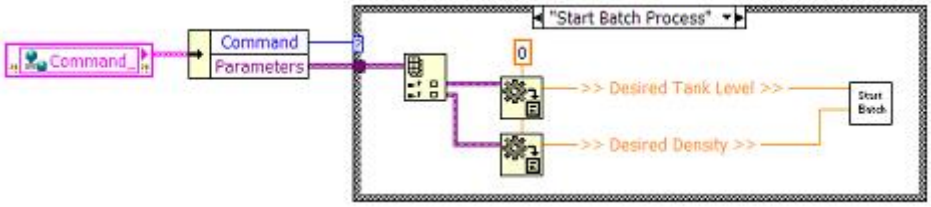


Рисунок 4.28. Обработка команды с расширенными параметрами

Подобное объединение команды и ее параметров обеспечивает связность параметров с командой и помогает избежать гонок (состязаний).

Для хранения информации используйте кластер с командой перечислительного типа и массивом для параметров данных. Массив может быть численного типа, но если вы сделаете его переменного типа (variant), то сможете использовать функции flatten и unflatten для передачи любых типов данных.

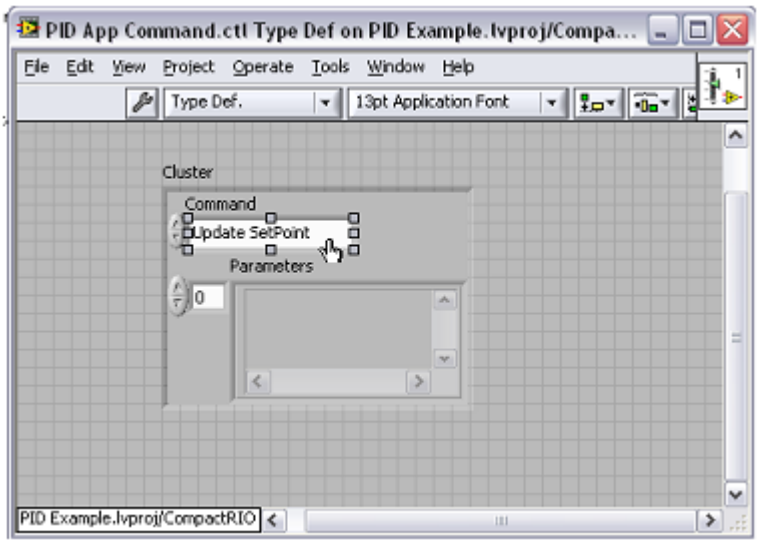


Рисунок 4.29. Кластер может содержать одновременно команду и связанные с ней данные

Теперь вы должны обновить тип публикуемой в сети переменной общего доступа, чтобы сделать ее кластером. Вы можете это сделать, изменив тип на "From Custom Control" (Пользовательский элемент управления) и выбрав созданный вами пользовательский элемент управления.

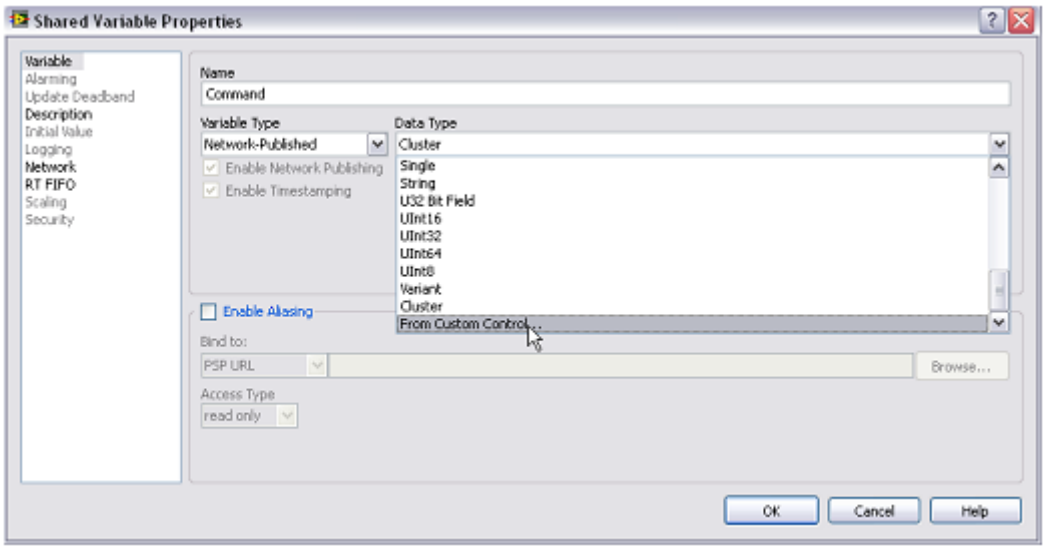


Рисунок 4.30. Публикуемые в сети переменные общего доступа могут быть пользовательских типов для соответствия кластеру команды

Чтобы при развертывании не зависеть от пользовательских типов, переменные общего доступа не создают постоянную ссылку на пользовательские элементы управления, используемые для определения их типов. Это значит, что если вы измените определение типа команды, например, чтобы добавить больше типов команд, вам потребуется заново выполнить привязку переменной общего доступа к пользовательскому элементу управления для обновления типа данных переменной. Чтобы исключить эту проблему, вы можете сделать другой кластер для переменной общего доступа, в котором используете тип U32 вместо типа enum. Вы можете использовать определение типа enum во всех приложениях, где читаете или записываете данные в переменную общего доступа.

Теперь измените задачу анализа команд для чтения команды и подключите ее к структуре case. Внутри каждого кадра структуры Case производится преобразование формы представления параметров (unflatten). В этом примере убран цикл коммуникаций, потому что уставка обновляется не из сети а по входной команде.

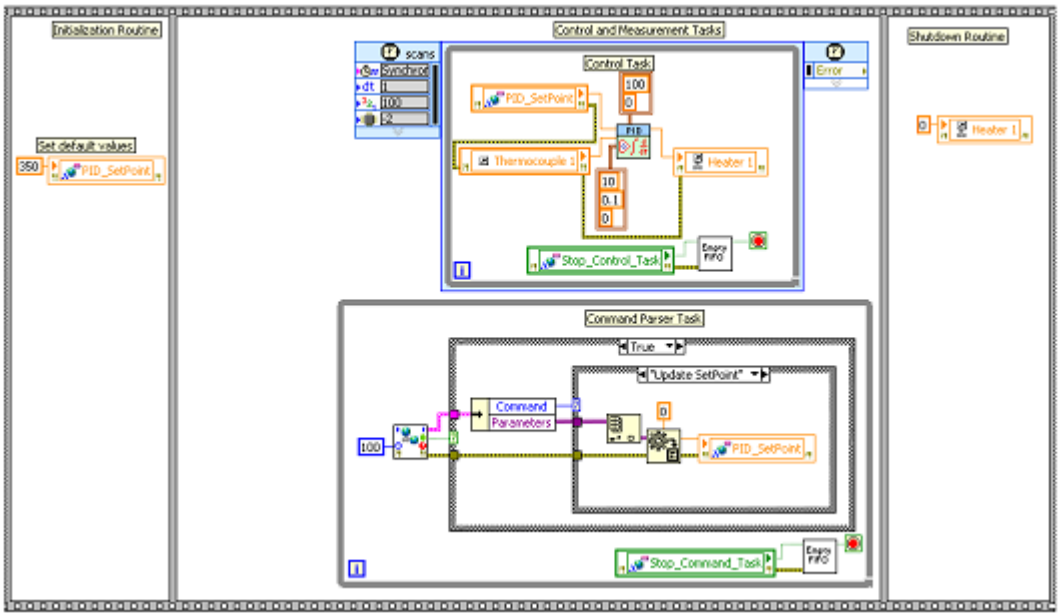


Рисунок 4.31. Завершенная блок-диаграмма с задачей анализа команд, которые содержат ассоциируемые с командами данные

Основанная на командах архитектура с квитированием

Протокол PSP обеспечивает встроенное управление соединениями для всех подписчиков. Узел переменной возвращает информацию о соединении и статусу состояния через ошибки и предупреждения в выходном терминале ошибок. Однако в некоторых приложениях выгодно реализовать более сложный механизм коммуникаций для получения подтверждения того, что контроллер не только получил команду, но и был в состоянии ее выполнить. Чтобы добавить функцию подтверждения, используйте вторую переменную для канала квитирования.

Рисунок 4.32. Упрощенная командная архитектура с квитированием на основе сетевых переменных
Commander - Начальник, Worker - Исполнитель, Command – команда, Network Variable – сетевая переменная, Acknowledge – подтверждение приема

Типы квитирования

В разных приложениях по разному определяются сообщения подтверждения. В то время, как одному приложению может быть безразлично, доставлено ли на самом деле сообщение (некритическая информация), другому может быть нужно получить подтверждение, что сообщение было действительно получено, а третьему может быть нужно получить подтверждение, что сообщение (или команда) было принято и на его основе были произведены какие-то действия. Поэтому используются следующие типы квитирования в контексте вашей командной архитектуры:

- По доставке (Upon Delivery): Цикл-Исполнитель прочитал команду из сети. Команда считается полученной.
- По приему (Upon Acceptance): Исполнитель оценил команду и определил, принять ее к исполнению или отказаться выполнять.
- По завершению (Upon Completion): Исполнитель отвечает только, когда команда выполнена, или сообщает Начальнику об ошибке при выполнении команды.

Для любой команды начальник может выбрать ожидание любого из этих уровней подтверждения получения.

Простой способ реализовать это – использовать определение типа enum для перечисления различных типов подтверждения.

Рисунок 4.33. Перечисление типов подтверждения

Помимо типа подтверждения, вы должны рассмотреть следующие элементы при реализации командной архитектуры с квитированием:

Идентификатор команды

Подтверждая прием нескольких экземпляров одной и той же команды, вы должны прикрепить к команде номер серии или идентификатор последовательности, чтобы можно было различать и подтверждать прием отдельных команд.

Идентификатор Начальника

Если в системе несколько начальников, команда должна уточнить свое происхождение с помощь уникального идентификатора, чтобы квитанция могла быть направлена соответствующему Начальнику.

На основании вышесказанного, вы можете разработать формат вашего командного сообщения, как показано ниже.

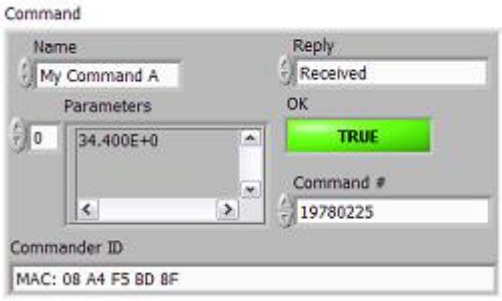


Рисунок 4.34. Примерный кластер данных для квитирования команд

Использование этого кластера как типа данных для вашей командной сетевой переменной позволит Исполнителю получать требуемую информацию и однозначно идентифицировать команду и ее происхождение. Таким образом, Начальник предоставляет номер команды в последовательности и идентификатор Начальника вместе с надлежащей командной информацией.

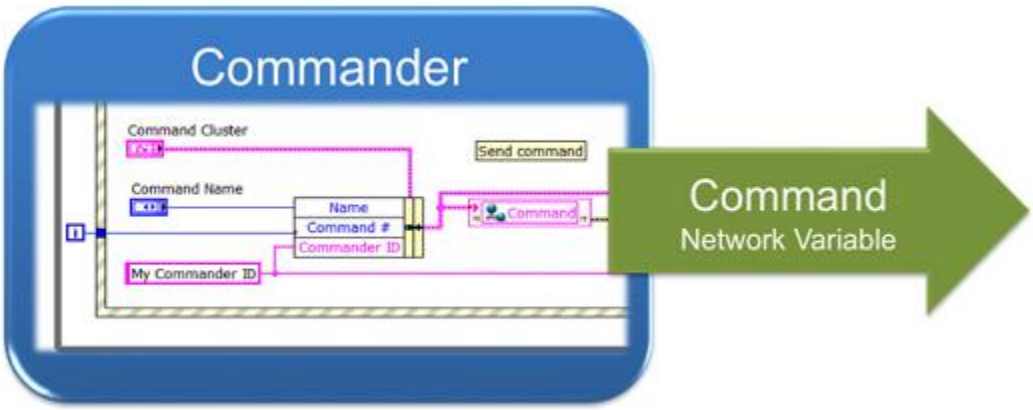


Рисунок 4.35 Отправка квитируемой команды
Commander - Начальник, Command – команда, Network Variable – сетевая переменная

Исполнитель принимает команды, используя ту же технологию, о которой говорилось в предыдущих разделах, и посылает в ответ квитанции. Созданное вами определение типа кластера без труда может быть повторно использовано для ответного сообщения. Вы реализуете квитирование, установив значение перечислительного типа reply (ответ) и статус OK, чтобы дать сигнал, что команда была принята успешно. Сообщение затем заносится в выделенную для квитирования сетевую переменную reply, которая аналогична переменной command.

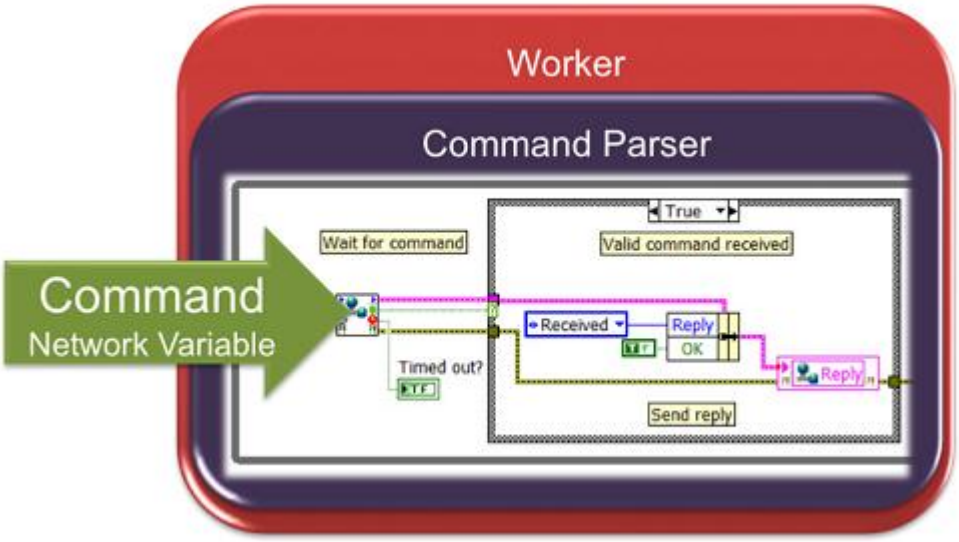


Рисунок 4.36. Получение и подтверждение приема команд
Command – команда, Network Variable –сетевая переменная, Worker - Исполнитель, Command Parser – анализатор команд

Ожидание квитанции – более сложная тема, поскольку различные ожидаемые квитанции могут прибывать не по порядку (разные команды могут подтверждать прием на различных стадиях). Пример основанной на командах архитектуры показывает, как VI может ожидать несколько квитанций одновременно. Используя этот код в качестве примера, вы можете построить простую версию Начальника, управляемого событиями.

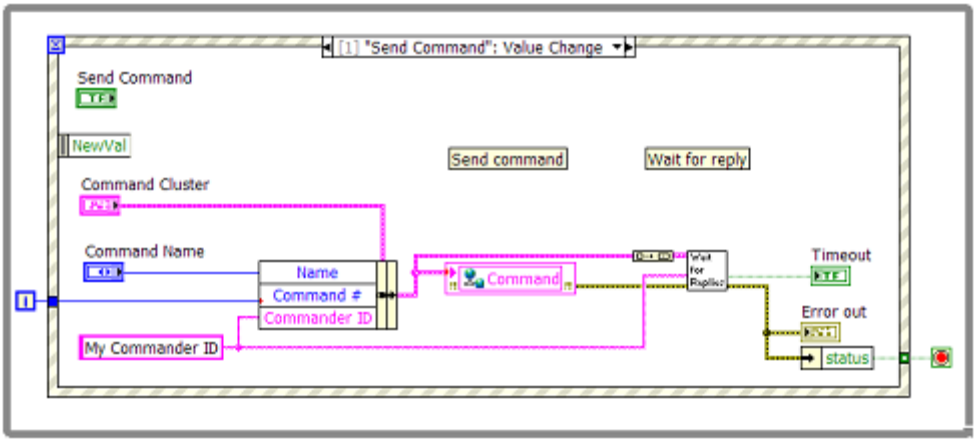


Рисунок 4.37. Простой VI Начальника с квитированием

Важные моменты при использовании сетевых переменных для команд

Связывание типа переменной

Чтобы при развертывании переменная общего доступа оставалась независимой от пользовательских типов, не следует создавать постоянную ссылку на пользовательские элементы управления, используемые для определения их типов. Это означает, что если вы измените определение типа для команды, например, чтобы добавить команды, вам потребуется заново выполнить привязку переменной общего доступа к пользовательскому элементу управления для обновления типа данных переменной.

Технология буферизации

Поскольку некоторые реализации командной переменной, показанные в предыдущем разделе, используют как тип пользовательские элементы управления, важно отметить, что для этих типов размер буфера, показанный на странице Network properties (Сетевые свойства) определяется в байтах, а не в элементах. Вам необходимо настроить этот размер, основываясь на предположениях о требованиях к приложению и учитывая максимальное число ожидающих выполнения команд, которые должен хранить буфер, количество Начальников и средний размер команды, включая возможные параметры с варьирующимся размером, которые могут быть инкапсулированы в команде.

Ограниченный буфер команд

Буферизированная сетевая переменная гарантирует, что последовательные значения буферизируются и посылаются всем подписчикам. Эта гарантия работает до тех пор, пока не переполнится какой-либо из отдельных буферов каждого из подписчиков. Переполнение буфера обычно вызывает предупреждения на уровне узла переменной, но в модели публикации-подписки сложно восстановиться после этих предупреждения.

История команд

Одним из побочных эффектов необходимости очистки канала коммуникаций для того, чтобы ни одна просроченная команда не выполнилась повторно, заключается в том, что любая из ожидающих выполнения команд, отправленная до запуска процесса-Исполнителя, выполнена не будет. Основанная на командах архитектура с квитированием позволяет смягчить эту ситуацию, предоставляя всем Начальникам обратную связь относительно доставки команд, в случае, если процесс-Исполнитель временно недоступен.

Необработанные посылки Ethernet (TCP/UDP)

В то время как публикуемые в сети переменные общего доступа - рекомендуемый метод для обмена данными между узлами LabVIEW, альтернативой использованию переменных общего доступа для сетевых коммуникаций является необработанные посылки TCP или UDP (raw TCP or UDP). TCP и UDP - компоненты низкого уровня всех стандартов Ethernet. Естественно, инструменты для необработанных TCP и UDP поддерживаются практически во всем программном обеспечении, включая и NI LabVIEW. Они предлагают низкоуровневые функции связи, которые более гибки, но менее удобны для пользователя. Функции наподобие установления соединения и упаковки данных должен быть обработаны на уровне приложения.

TCP или UDP - хорошие варианты, если нужен очень низкий уровень управления над коммуникационным протоколом. Они могут также использоваться, если вам нужно установить связь с устройствами сторонней фирмы, которые не поддерживают связь через переменные общего доступа, однако другие стандартные протоколы типа Modbus TCP проще в использовании.

TCP обеспечивает коммуникации точка-точка с обработкой ошибок для гарантированной доставки пакета. UDP может вести широковещательную рассылку одной и той же информации сразу нескольким устройствам. Широковещательные сообщения UDP могут быть отфильтрованы сетевыми коммутаторами и не гарантируют доставки пакета.

Коммуникации по TCP следуют схеме клиент-сервер, когда сервер слушает определенный порт, через который открыто соединение с клиентом. Как только будет установлено соединение, вы можете свободно обмениваться данными с помощью базовых функций чтения и записи. В LabVIEW функции TCP все данные передают, как строки. Это означает, что вы должны привести в строковый формат логические или числовые данные для записи и выполнить обратные преобразования после прочтения. Поскольку сообщения могут быть разной длины, программист должен знать, сколько данных содержится в конкретном сообщении и считывать надлежащее количество байт. Обратитесь к примерам LabVIEW Data Server.vi и Data Client.vi для обзора основ клиент-серверной обмена данными в LabVIEW.

Создание собственного коммуникационного протокола

Если вы собираетесь использовать необработанные посылки TCP или UDP для передачи и приема данных, вам нужно создать ваш собственный протокол, определяющий способы упаковки и распаковки данных.Для масштабируемости коммуникационный протокол должен иметь следующие характеристики:

- Простые упаковка и анализ данных
- Абстрагирование деталей реализации транспортного уровня (TCP/IP, UDP, и т.п.)
- Минимизация сетевого трафика путем посылки данных только тогда, когда нужно
- Минимизация накладных расходов и снижения пропускной способности
- Пригодность для коммуникаций с приложениями, созданными не только в LabVIEW (C, C++ и т.п.)

В каждом протоколе сообщений имеются некоторые служебные данные (метаданные), связанные с анализом потока данных на принимающей стороне. Посылка полного набора метаданных с каждым пакетом добавляет значительные накладные расходы. Поскольку, скорее всего, вам нужно высокопроизводительное приложение, вам захочется минимизировать издержки, посылая ограниченный набор метаданных с каждым пакетом.

Коммуникации сообщений и процессов

В основе протокола TCP лежат сообщения, так что ваш протокол должен выполнять дополнительную логику для передачи данных процесса. Эта логика может быть простой периодической пересылкой всех данных или же может реализовывать обнаружение изменений для уменьшения количества передаваемых по сети данных.

Пример пользовательского протокола коммуникаций

В данном разделе приведен пример кода LabVIEW

Разработчики, создающие пользовательские коммуникационные протоколы, обычно стараются оптимизировать код под конкретное приложение. Поэтому здесь не могут быть предоставлены пошаговые инструкции по созданию пользовательского протокола. Однако мы можем рассмотреть пример разработанного в LabVIEW пользовательского протокола, который продемонстрирует, как правильно создавать протокол. Этот пример разработан таким образом, чтобы вы могли использовать его непосредственно, если вам нужно обратиться к устройствам вне LabVIEW, или он может послужить опорной точкой для создания вашего собственного протокола.

Простой протокол передачи сообщений (Simple Messaging Protocol, STM) абстрагируется от некоторых сложностей коммуникации необработанного TCP. Доставка сообщений гарантируется как результат буферизации и квитирования на уровне протокола, и используется основанный на именах механизм передачи сообщений. Он предоставляет определенный заголовок для каждого сообщения, включающий размер пакета данных и индекс, связывающий сообщение с метаданными, которые были заранее определены для описания содержимого сообщения.

Этот протокол реализован так, что клиент и сервер обмениваются метаданными, когда вы впервые установите сообщение. Это предотвращает передачу излишней информации и сохраняет относительно небольшой (6 байт) размер служебной информации, передаваемой с каждым сообщением. В то время как записывающий элемент отвечает за преобразование данных в строку, читатель отвечает за то, что знает, как интерпретировать данные конкретного сообщения. После прочтения сообщения вы можете анализировать его или преобразовать к соответствующему типу данных на основании содержащейся в метаданных информации.

Протокол STM – реализация в LabVIEW

STM – настраиваемый протокол Ethernet для LabVIEW. На сайте ni.com вы можете найти инсталлятор компонента Simple Messaging Reference Library (STM), который включает программный интерфейс приложения (API) и код примера.

Некоторые примеры ориентированы на приложения LabVIEW Real-Time, но большинство запускаются на любой платформе LabVIEW.

Метаданные

Метаданные реализованы как массив кластеров. Каждый элемент массива содержит свойства данных, необходимые для упаковки и декодирования одного значения переменной. Вы определили только свойство Name, но вы можете использовать кластер для настройки STM, добавляя метасвойства (например, тип данных) в соответствии с требованиями вашего приложения. Кластер метаданных определяется типом данных, так что добавление свойств не испортит вам код.

На рисунке 4.38 показан пример кластера метаданных, настроенного на две переменные: Iteration и RandomData.

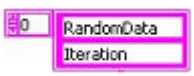


Рисунок 4.38. Массив строк метаданных

До передачи каждой переменной создается пакет, включающий поля для размера данных, идентификатора метаданных и самих данных. На рисунке 4.8 приведен формат пакета.

Data Size - размер данных (32 бита)	Metadata ID - идентификатор метаданных (16 бит)	Data - данные

Рисунок 4.39. Формат пакета

В поле идентификатора метаданных заносится индекс элемента массива метаданных, соответствующего переменной данных. Получающий узел использует идентификатор метаданных для индексации массива метаданных и получения свойств данных сообщения.

Простой программный интерфейс приложения TCP/IP для передачи сообщений

Программный интерфейс приложения (API) STM очень прост. Для базовых операций он состоит из VI Read VI и VI Write. Также есть два вспомогательных VI для помощи в передаче метаданных, но их использование не обязательно. Каждый из главных VI является полиморфным, что позволяет применять их на разных транспортных уровнях. API для каждого уровня очень похожи. Далее приведено краткое описание основных VI программного интерфейса приложения (только для версии TCP/IP). Дополнительно установленные VI описаны в справке по VI.

STM Write Message (Запись сообщения)

Используйте этот VI для передачи любого типа данных удаленному узлу. Этот VI создает пакет из данных, имени данных и метаданных. При вызове этого VI он возвращает индекс переменной, заданной параметром Name в массиве метаданных. Далее он собирает пакет сообщения и посылает его удаленному хосту через TCP/IP, используя идентификатор соединения.

Connection Info (Информация о соединении) состоит из ссылки на транспортный уровень и массив метаданных. Этот выход есть и в VI STM Read Metadata (Чтение метаданных), и в VI STM Write Metadata (Чтение метаданных).

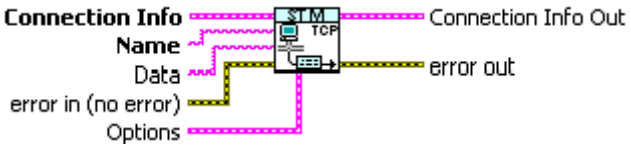


Рисунок 4.40. STM Write Message (TCP).vi

STM Read Message (Чтение сообщения)

Используйте этот VI для получения любого типа данных с удаленного узла. Это VI читает и извлекает из пакета индекс метаданных и преобразует в строковые данные. Он ищет метаэлемент и возвращает его вместе со строкой данных. Затем приложение может преобразовать строковые данные в тип данных сообщения, используя имя или другие метасвойства.

Этот VI обычно используется в цикле. Поскольку нет гарантий, что данные прибудут в заданное время, параметр "timeout" (тайм-аут) позволяет циклу запускаться периодически.

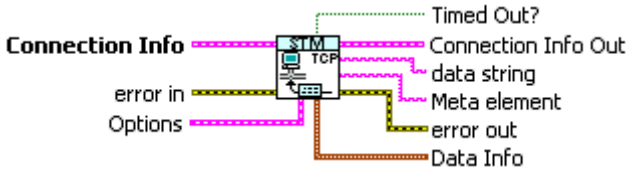


Рисунок 4.41. STM Read Message (TCP).vi

STM Write Metadata (Запись метаданных)

Используйте этот VI для передачи метаданных удаленному узлу. Для правильной интерпретации сообщения метаданные должны согласоваться и на приемной и передающей стороне. Вместо сохранения копий данных на каждом хосте, вы можете хранить метаданые на сервере и использовать этот VI для передачи их клиентам при соединении.

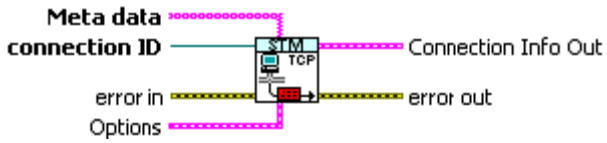


Рисунок 4.42. STM Write Metadata (TCP).vi

STM Read Metadata (Чтение метаданных)

Используйте этот VI для получения метаданных с удаленного компьютера. Этот VI считывает и распаковывает массивы метаданных, которые могут быть переданы VI чтения и записи.

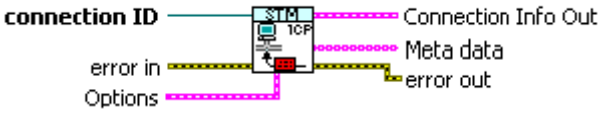


Рисунок 4.43. STM Read Metadata (TCP).vi

Использование STM API для передачи данных

На рисунке 4.44 показан пример сервера данных с использованием программного интерфейса приложения STM. Обратите внимание, что эта программа посылает метаданные удаленному хосту сразу после установления соединения. Этот пример записывает два значения: счетчик итераций и массив элементов типа double. Метаданные состоят из описания этих двух переменных.

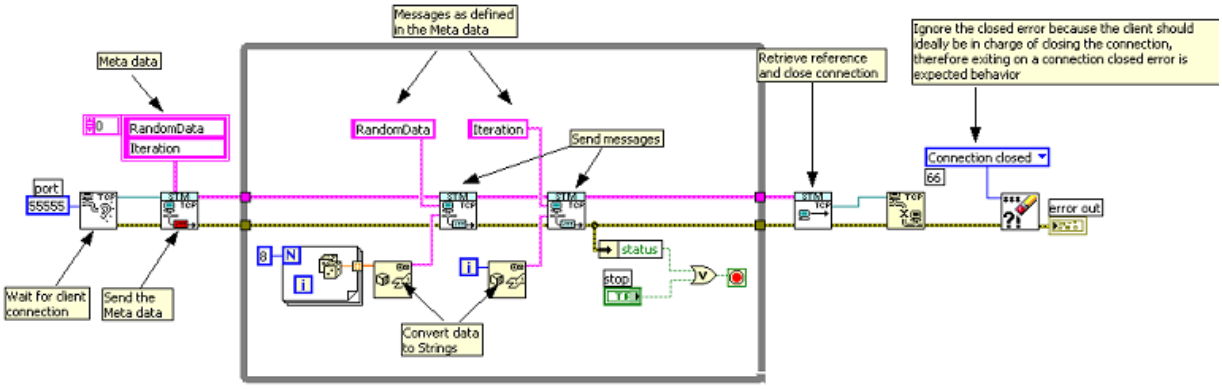


Рисунок 4.44. Базовый пример сервера STM

Использование STM API для приема данных

Прием данных также очень прост. В шаблоне проекта показано ожидание метаданных при установлении соединения с сервером. Далее используется STM Read Message VI для ожидания входящих сообщений. Когда сообщение получено, VI преобразует данные и присваивает им локальное значение в соответствии с именем метаданных.

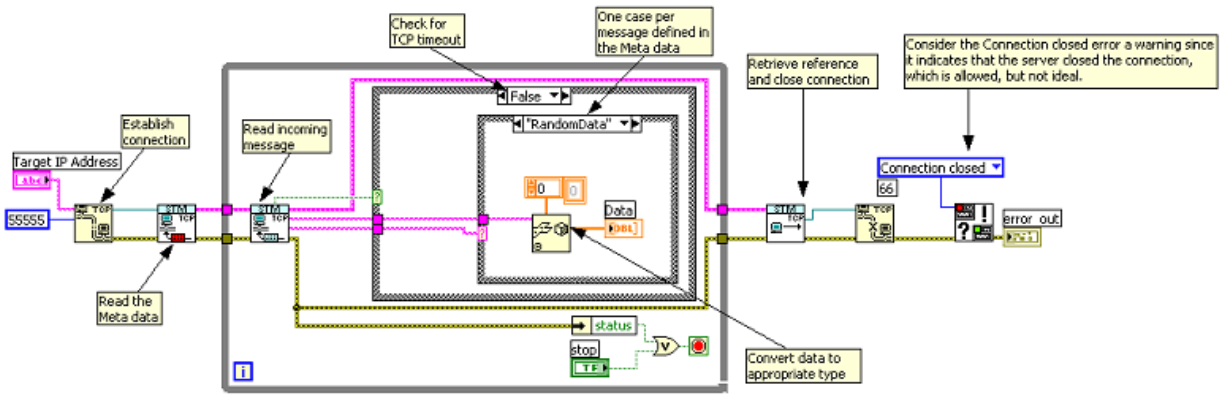


Рисунок 4.45. Базовый пример клиента STM

Для большинства приложений LabVIEW, обменивающихся данными между собой, National Instruments рекомендует использовать публикуемые в сети переменные общего доступа, а для связи с системами сторонних фирм - высокоуровневые протоколы, подобные Modbus TCP.

Однако LabVIEW дает вам возможность создать собственные коммуникационные протоколы, подобные протоколу STM.

Коммуникации через последовательный порт CompactRIO

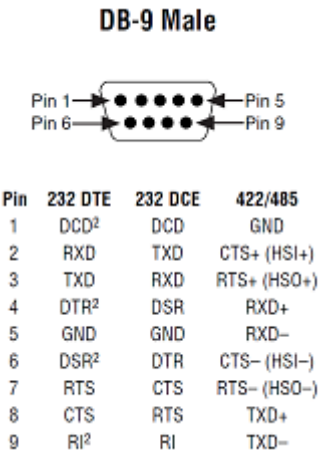
Порты интерфейса RS232 (известного также, как EIA232 или TIA232), являющегося стандартом уже более 40 лет, могут быть найдены во всех системах CompactRIO и широком наборе промышленных контролеров, ПЛК и прочих устройств. RS232 остается стандартном де-факто для недорогих коммуникаций из-за простоты, незначительных накладных расходов, низких требований к обработке и умеренной полосы пропускания. В то время, как RS232 исчезает из индустрии ПК в пользу новых шин USB, IEEE 1394 и PCI Express, он продолжает оставаться популярным в промышленном секторе благодаря своей простоте, низкой стоимости реализации и широкого распространения портов RS232.

Спецификации RS232 охватывают реализацию аппаратного обеспечения, синхронизацию на битовом уровне и доставку байт, но не определяют спецификации передачи сообщений. Это упрощает реализацию базового побайтового чтения и записи, а более сложная функциональность может различаться в разных устройствах. Существует набор протоколов, использующих основанную на RS232 побайтовую передачу сообщений, например, Modbus RTU/ASCII, DNP3 и IEC-60870-5.

В этом разделе рассматривается, как программируются коммуникации байтового уровня через последовательный порт, встроенный в контролеры реального времени NI CompactRIO, с использованием NI-VISA API.

Введение в технику RS232

RS232 – стандарт коммуникаций точка-точка, означающий, что только два устройства могут быть соединены друг с другом. Стандарт определяет несколько сигнальных линий, в различной степени используемых разными приложениями. Как минимум, все последовательные устройства используют линии передачи (TXD), приема (RXD), и заземления. Другие линии - это линии управления потоком данных, запрос на передачу (Request to Send) и очистка для передачи (Clear to Send), которые иногда используются для улучшения синхронизации между устройствами и предотвращает потерю данных, когда приемник не успевает за передатчиком. Оставшиеся линии, как правило, используются в приложениях модем-хост и редко нужны в промышленных приложениях, за исключением радиомодемов.



Pin	232 DTE	232 DCE	422/485
1	DCD[2]	DCD	GND
2	RXD	TXD	CTS+ (HSI+)
3	TXD	RXD	RTS+ (HSO+)
4	DTR[2]	DSR	RXD+
5	GND	GND	RXD–
6	DSR[2]	DTR	CTS– (HSI–)
7	RTS	CTS	RTS– (HSO–)
8	CTS	RTS	TXD+
9	RI[2]	RI	TXD–

Рисунок 4.46. Схема расположения выводов 9-контактного разъема D-Sub

Как несимметричная шина, RS232 идеален при коммуникациях на небольшие расстояния (до 10 м). Экранированные кабели уменьшают помехи на более длинных дистанциях. Для больших расстояний и более высоких скоростей предпочтительно использование RS485, поскольку там для уменьшения помех используется дифференциальный сигнал. В отличие от шин, подобных USB и IEEE 1394, RS232 не может служить источником питания оконечных устройств.

Контролеры CompactRIO имеют порт RS232 со стандартным 9-контактным разъемом D-Sub (известным также как DE-9 или DB-9). Вы можете использовать этот порт для мониторинга диагностических сообщений из контроллера CompactRIO если в вашем приложении есть консоль или переключатель для связи с дешевыми устройствами RS232.

Монтаж и прокладка кабелей RS232

Существуют две стандартных схемы расположения выводов для портов RS232. Устройства RS232 делятся на оконечные устройства (DTE), аналогичные главному устройству (master или host), и коммуникационные устройства (DCE), аналогичные подчиненным (slave). Порты DCE подключаются "инверсно", так что входные контакты DCE подключаются к выходным контактам DTE и наоборот. Последовательный порт в контроллерах CompactRIO - порт DTE, а последовательный порт таких устройств, как GPS, сканер штрих-кода, модем или принтер - порт DCE.

При подключении порта CompactRIO DTE RS232 к типичному оконечному устройству с портом DCE RS232 используется "прямой" кабель. При соединении двух устройств DTE или DCE, например, контроллера CompactRIO с ПК, вам нужно использовать нуль-модемный кабель, в котором подключение сигналов передачи и приема изменено.

Устройство 1	Устройство 2	Тип кабеля
DTE	DTE	Нуль-модемный
DTE	DCE	Однопроходный
DCE	DTE	Однопроходный
DCE	DCE	Нуль-модемный

Таблица 4.1. Выбор кабеля для подключения различных портов RS232

Заглушка для тестирования

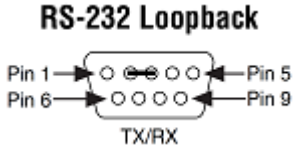


Рисунок 4.47. Заглушка для тестирования RS232

Обычно для проверки правильности функционирования и диагностики неисправностей последовательного порта применяется тестовая заглушка. На базовом уровне вам нужно только соединить контакты TXD и RXD. Используйте стандартный прямой или перекрестный кабель RS232 и небольшую изогнутую скрепку для замыкания контактов 2 и 3. Когда установлен кабель с заглушкой, байты, посылаемые функцией записи, могут быть прочитаны функцией чтения. Вы можете удостовериться, что программное обеспечение, настройки последовательного порта и драйвера работают верно, используя эту технику.

Обмен данными в LabVIEW через последовательный порт

Последовательный порт CompactRIO RS232 подключен непосредственно к процессору реального времени CompactRIO, так что вы должны поместить код, обращающийся к последовательному порту, в VI LabVIEW Real-Time. Чтобы передавать и получать данные через последовательный порт, используйте функции NI-VISA.

NI-VISA - это драйвер, обеспечивающий интерфейс для связи с интерфейсами байтового уровня, подобные RS232, RS485, GPIB и т.п. Код с функциями NI-VISA может использоваться на любой машине с последовательным портом и установленным NI- VISA. Это означает, что вы может писать и тестировать последовательный Serial VI на ПК с Windows в LabVIEW, а затем повторно использовать тот же код в LabVIEW Real-Time на CompactRIO. Вы можете также непосредственно использовать тысячи уже написанных и проверенных инструментальных драйверов с сайта ni.com/idnet.

Для начала работы с последовательным портом обратитесь к функциям Virtual Instrument Software Architecture (VISA) в палитре **Data Communication»Protocols»Serial** (Обмен данными»Протоколы»Последовательный).

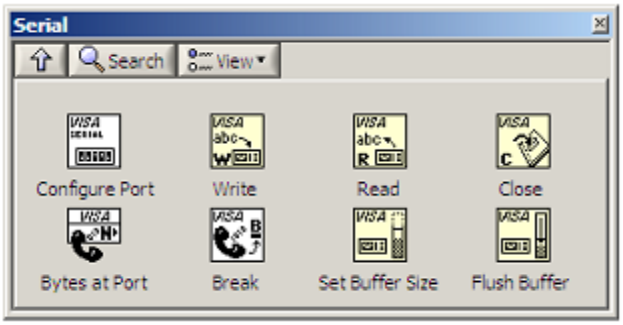


Рисунок 4.48. Функции VISA

Для большинства простых измерительных приложений вам понадобятся только две функции VISA, VISA Write (Запись) и VISA Read (Чтение).

Посмотрите VI Basic Serial Write and Read (Базовые операции чтения и записи последовательного порта) в библиотеке labview\examples\instr\smplserl.llb для ознакомления с примерами использования функций VISA.

Для большинства устройств сначала требуется послать информацию в виде команды или запроса, прежде чем вы сможете считывать информацию из устройства. Поэтому функция VISA Write обычно предшествует функции VISA Read. Поскольку коммуникации через последовательный порт требуют от вас настройки дополнительных параметров, таких, как скорость передачи в бодах, вы должны начать коммуникации через последовательный порт с VI VISA Configure Serial Port (Конфигурирование последовательного порта), который инициализирует порт, по идентификатору - имени ресурса VISA.

Важно отметить, что имя ресурса VISA относится к ресурсам компьютера, на котором вы будете запускать VI.

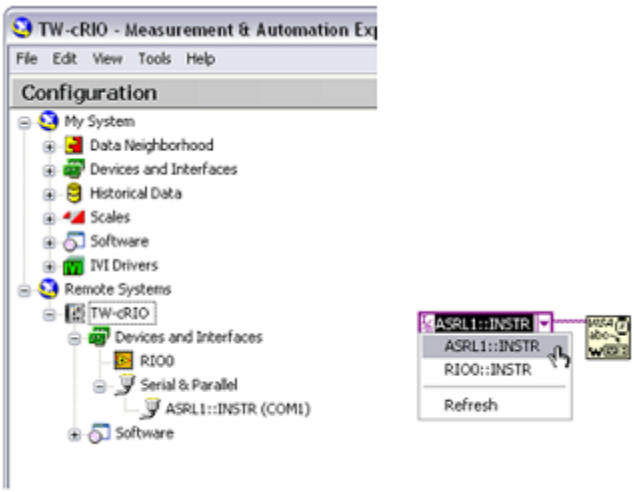


Рисунок 4.49. Вы можете видеть имена ресурсов VISA в элементе управления ресурсами VISA в LabVIEW или в Measurement & Automation Explorer

В нижнем левом углу окна VI можно увидеть, с каким целевым устройством связан VI. В CompactRIO используйте встроенный порт COM1. COM 1 в контроллере CompactRIO обозначается ASRL1::INSTR. Если вы подключены к CompactRIO, вы можете непосредственно просматривать имена ресурсов VISA в элементе управления ресурсами VISA в LabVIEW или с помощью Measurement and Automation Explorer (загружаемая утилита, используемая для настройки всех устройств NI).

Тайм-аут устанавливает значение времени ожидания для коммуникаций через последовательный порт. Скорость передачи данных в бодах, количество бит данных, паритет и управление потоком определяют индивидуальные параметры последовательного порта. Вход и выход кластеров ошибок позволяют обрабатывать ошибочные состояния VI.

Когда порт работает на уровне байт, функций чтения и записи соединяют со строками. В памяти строка – просто набор байт с указанной длиной, что облегчает программе работу с многими байтами.

Строковые функции в LabVIEW предоставляют возможность разбиения, манипуляции и комбинирования данных, полученных из последовательного порта.

Вы также можете преобразовать строковые данные в массив байт, используя функции массивов LabVIEW для работы с низкоуровневыми последовательными данными.

Поскольку при работе с последовательным портом имеют дело со строками разной длины, соответствующие задачи являются недетерминированными. Кроме того, последовательный интерфейс по своей природе асинхронен и не имеет механизма для гарантии своевременной доставки данных. Чтобы обеспечить детерминизм вашего управляющего цикла при программировании приложений с последовательным портом, поместите код коммуникаций в отдельный от управляющего кода цикл.

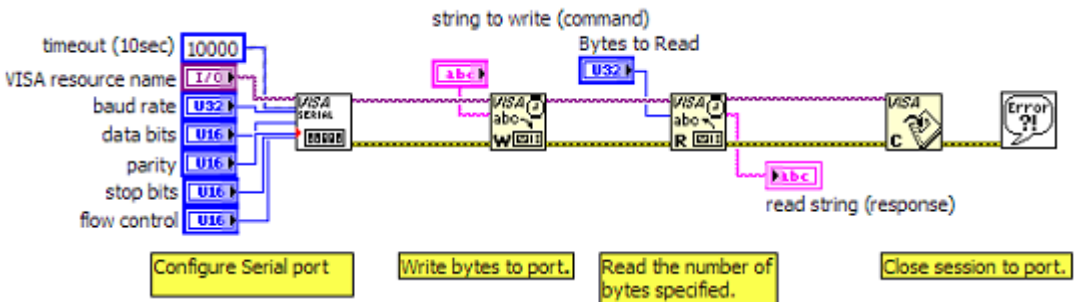


Рисунок 4.50. Простой пример последовательных коммуникаций

Функция VISA Read ждет, пока количество затребованных байтов не поступит в последовательный порт или пока не произойдет тайм-аут. Для приложений, где количество байт известно, такое

поведение приемлемо, но в некоторых приложениях встречаются различные объемы поступающих данных. В этом случае вы должны сосчитать количество доступных для чтения байтов, и прочитать только эти байты. Вы можете добиться этого с помощью свойства Bytes at Serial (Байты в последовательном порте), доступного из палитры Serial.

Вы можете получать доступ к параметрам последовательного порта, переменным и состояниям с помощью узлов свойств. Если вы откроете VI VISA Configure Serial Port (Конфигурирование последовательного порта) и изучите его код, то увидите, что этот VI просто обращается к различным узлам свойств для настройки порта. Вы можете использовать эти узлы, которые предоставляют расширенные возможности работы с последовательным портом, включая дополнительные линии данных, ожидание байтов для чтения или записи, и размер памяти буфера. На рисунке 4.51 показан пример использования узла свойств для проверки количества байт, доступных в порту, перед операцией чтения.

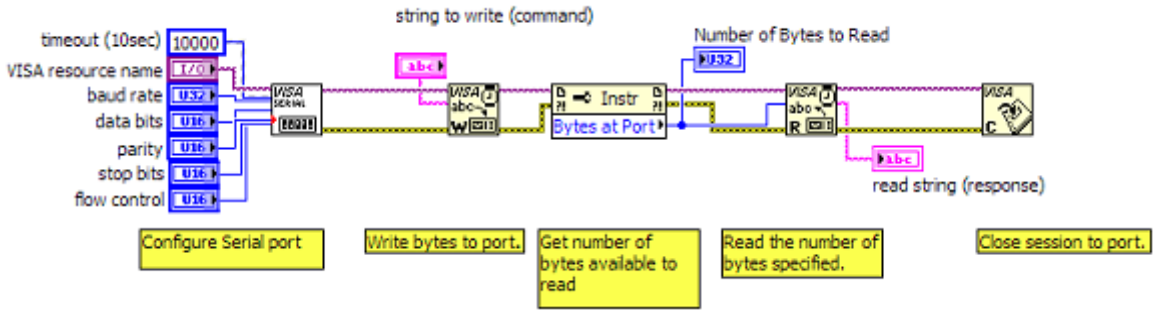


Рисунок 4.51. Использование узла свойств VISA для чтения количества байт

Вообще, работа с устройствами может варьироваться от простого чтения периодически передаваемых по радиоканалам байт, таких, как данные GPS, до работы со сложными командными структурами. Для некоторых устройств и приборов вы можете найти заранее запрограммированные инструментальные драйверы на сайте ni.com/idnet, что может упростить разработку. Поскольку эти драйверы используют функции NI-VISA,

они работают и со встроенным последовательным портом CompactRIO.

Сеть инструментальных драйверов

Чтобы помочь ускорить разработку, компания National Instruments работала с основными поставщиками измерительных и управляющих устройств по созданию библиотеки готовых к использованию драйверов для более чем 7000 устройств и приборов. Инструментальный драйвер – набор программ, которые управляют программируемым измерительным прибором. Каждая программа отвечает за программную операцию, например, конфигурирование, чтение, запись и запуск прибора. Инструментальные драйверы упрощают управление прибором и сокращают время разработки программы, исключая необходимость овладевать программным протоколом каждого устройства.

Где найти инструментальные драйверы и как загрузить их

Вы можете найти и загрузить инструментальные драйверы двумя способами. Если вы используете LabVIEW 8.0 или выше, самый легкий способ – воспользоваться Поисковиком инструментальных драйверов (NI Instrument Driver Finder). Если у вас более старая версия LabVIEW, то вы можете использовать Сеть инструментальных драйверов (Instrument Driver Network) по адресу ni.com/idnet.

Используйте Поисковик инструментальных драйверов для поиска, загрузки и установки драйверов LabVIEW, поддерживающих Plug and Play.

Выберите Tools»Instrumentation»Find Instrument Drivers (Приборы»Измерительные приборы»Поиск инструментальных драйверов) для запуска Поисковика инструментальных драйверов. Этот Поисковик выполняет поиск по IDNet, чтобы найти заданный драйвер. Обратитесь к рисунку 4.52, чтобы увидеть, как запустить Поисковик инструментальных драйверов из LabVIEW.

Вы можете использовать инструментальный драйвер для конкретного прибора, как есть, без изменений. Однако Plug and Play инструментальные драйверы LabVIEW поставляются вместе с блок-диаграммой исходного кода, так что вы можете адаптировать их для конкретного приложения. Вы можете создать приложения и системы для управления измерительным прибором, программно связав VI инструментального драйвера на блок-диаграмме.

Пример коммуникаций через последовательный порт в LabVIEW

В данном разделе приведен пример кода LabVIEW

Теперь измените исходный пример с ПИД-регулятором так, чтобы значения температуры поступали не из встроенного модуля с термопарой, а из измерительного прибора с последовательным портом. В нашем случае – это прибор Lake Shore Cryotronics 211.

Из IDNet вы можете найти и скачать инструментальный драйвер. Вы можете также искать драйвер непосредственно из LabVIEW.

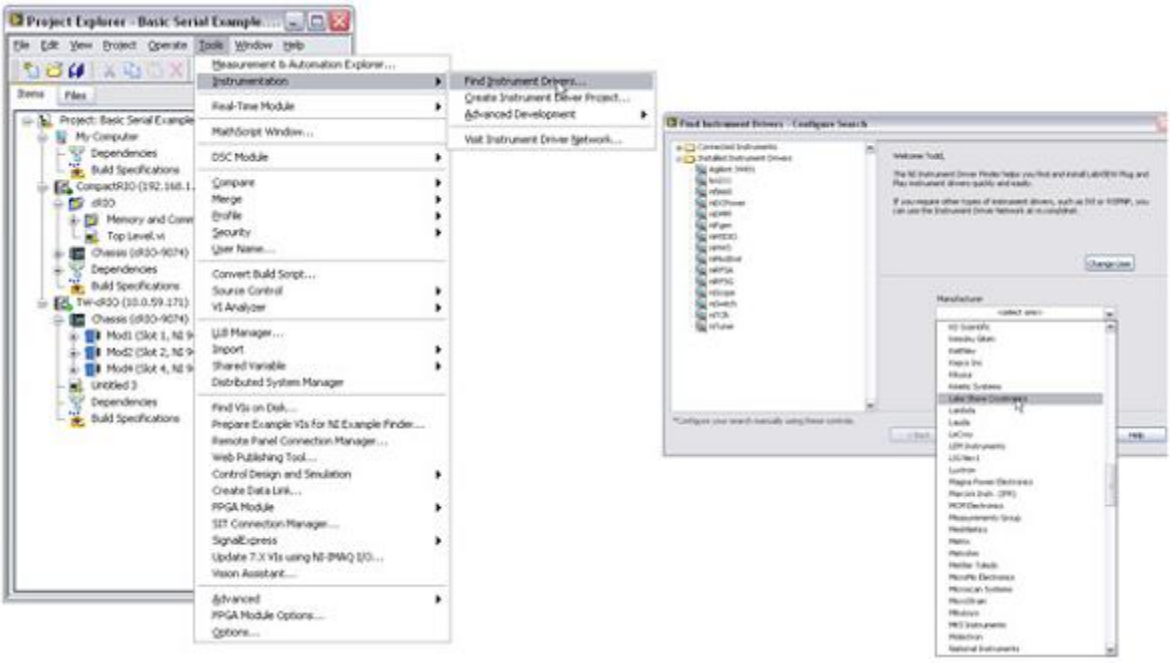


Рисунок 4.52. Из LabVIEW вы можете непосредственно искать и инсталлировать коммуникационные драйвера более чем для 7000 устройств

Утилита автоматически загружает и сохраняет драйвер в файл instr.lib в директории LabVIEW. Драйвера, сохраняемые в этой папке, автоматически показываются в палитре при запуске LabVIEW, так что в LabVIEW вы увидите новый элемент в палитре Instrument Drivers (инструментальные драйверы) для прибора мониторирования температуры LSCI 211.

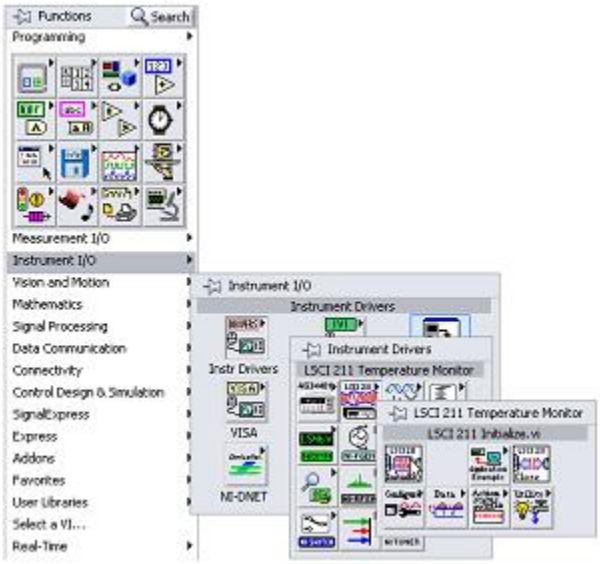


Рисунок 4.53. Вы можете увидеть инструментальные драйверы в палитре Instrument I/O

Далее модифицируйте ПИД-приложение так, чтобы Thermocouple 1 стала переменной общего доступа типа Single Process с FIFO реального времени (а не псевдонимом ввода-вывода). Установите значение по умолчанию для Thermocouple 1 в программе инициализации.

Наконец, добавьте второй цикл для задачи коммуникации через последовательный порт. Перед входом в цикл инициализируйте устройство последовательных коммуникаций, в цикле – считывайте значения температуры и записывайте в табличную память (переменную общего доступа типа Single Process). После завершения цикла закройте задачу последовательных коммуникаций.

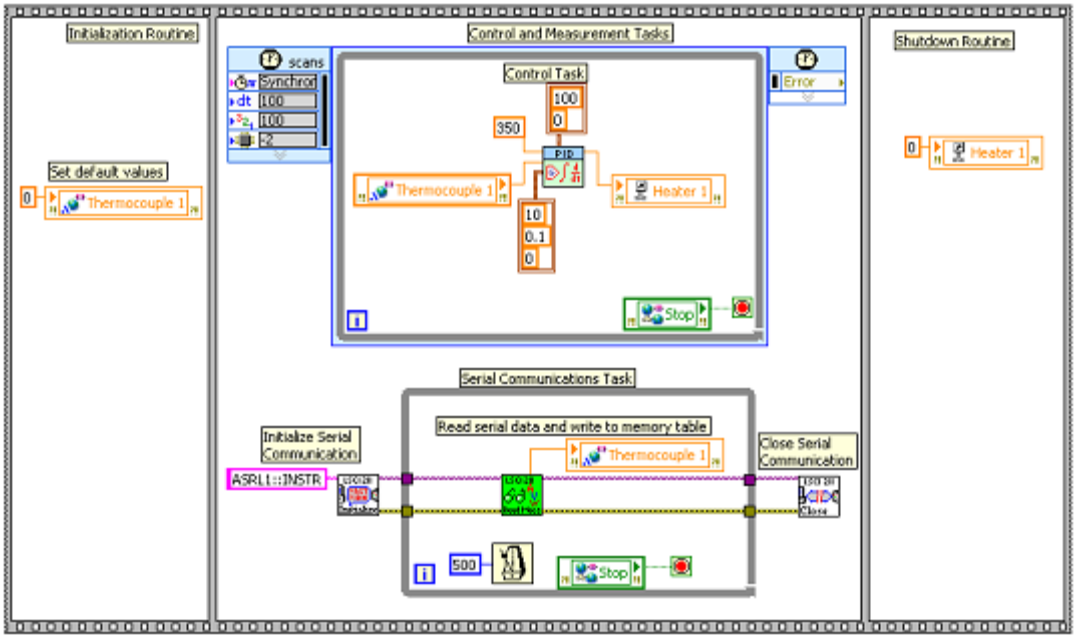


Рисунок 4.54. Завершенное приложение, считывающее результаты измерения температуры из устройства с последовательным портом и передающее их в управляющий цикл с высоким уровнем приоритета

Модули C-серии с 4-мя портами RS232 и RS422/RS485 для CompactRIO

Если в вашем приложении требуется высокая производительность или дополнительные порты RS232 и/или порты RS485/RS422, вы можете использовать модули CompactRIO последовательных интерфейсов NI 9870 и NI 9871. К этим платам обращаются напрямую из FPGA, используя LabVIEW FPGA. Программирование в LabVIEW FPGA подробно описано в следующем разделе.

Связь с ПЛК и другими промышленными сетевыми устройствами

Часто приложениям требуется интеграция программируемых контроллеров автоматизации (PAC), подобных CompactRIO, в существующую промышленную систему, или связь их с другими промышленными устройствами. Эти системы могут состоять из традиционных PLC, PAC или широкого набора специализированных устройств, таких как контроллеры двигателей, датчики и человеко-машинные интерфейсы. Эти устройства обладают широким диапазоном возможностей для коммуникаций, начиная с простого RS232 и заканчивая специализированными высокоскоростными промышленными сетями.

Существуют три метода для подключения CompactRIO к промышленной системе или PLC. Вы можете напрямую подключить CompactRIO к PLC, используя стандартный аналоговый или цифровой ввод-вывод. Этот метод прост, но подходит только для очень небольших систем. Для больших систем вы можете использовать промышленные коммуникационные протоколы для связи между CompactRIO и PLC. Для больших SCADA-приложений предпочтительно использование OPC. OPC может работать с очень большим количеством каналов, но использует технологию Windows, и, следовательно, требует наличия в системе Windows PC для обмена данными.

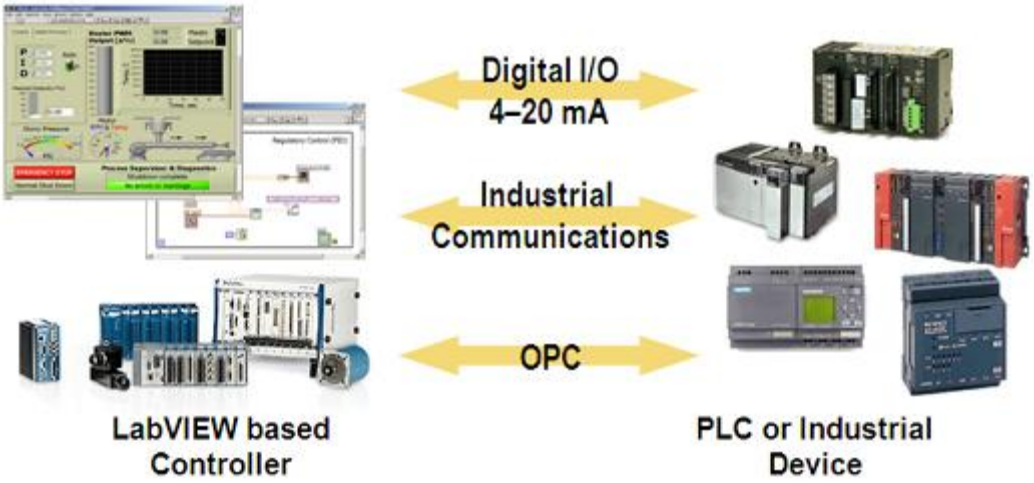


Рисунок 4.55. Три метода подключения контроллера CompactRIO к промышленному устройству

Промышленные коммуникационные протоколы

Наиболее распространенное средство подключения CompactRIO к другим промышленным устройствам, после стандартного ввода-вывода, это промышленные коммуникационные протоколы. Большинство протоколов используют стандартные физические уровни, такие, как RS232, RS485, CAN или Ethernet. Как обсуждалось ранее, шины наподобие RS232 и Ethernet обеспечивают физический уровень для простых коммуникаций, но у них нет предопределенных методов коммуникаций на верхних уровнях, поэтому пользователь может посылать и получать лишь отдельные байты и сообщения. При работе с большими объемами промышленных данных, обработка этих байтов и преобразование их в данные, подходящие для задач управления, быстро становятся очень трудоемкими. Промышленные коммуникационные протоколы обеспечивают основу для коммуникации данных. На базовом уровне они похожи на обсуждаемые ранее инструментальные драйверы, хотя обычно выполняют более сложные задания.

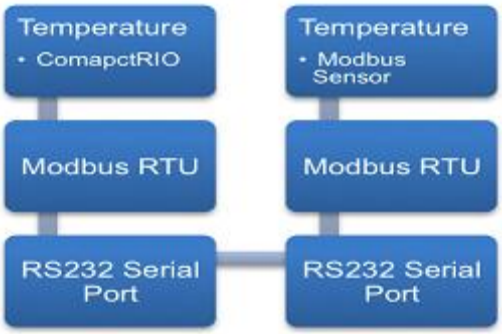


Рисунок 4.56. Типичная реализация промышленного протокола
Temperature - температура, Modbus Sensor - датчик с выходом в стандарте Modbus, Modbus RTU – разновидность протокола Modbus, RS232 Serial Port – последовательный порт RS232

Промышленные протоколы используют низкоуровневую шину - RS232 или TCP/IP, и добавляют предопределенные средства для коммуникации на верхнем уровне. Это очень похоже на то, как HTTP, протокол для доставки веб-контента, находится над TCP/IP. Можно разработать свой собственный HTTP-клиент, подобный Internet Explorer или Firefox, но это потребует больших усилий, а клиент может не соединяться со всеми серверами корректно без часов отладки и тестирования. Аналогично, устройства и контроллеры, поддерживающие промышленные сетевые стандарты, намного проще интегрировать на программном уровне и абстрагировать детали низкоуровневых коммуникаций от конечного пользователя, позволяя инженерам сосредоточиться на приложении.

Поскольку CompactRIO имеет встроенные последовательный порт и порт Ethernet, он может легко поддерживать многие протоколы, включая Modbus TCP, Modbus Serial, Ethernet-IP и другие. Существуют также встраиваемые модули, поддерживающие большинство промышленных протоколов, такие, как PROFIBUS, CANopen и другие.

Когда непосредственная коммуникация недоступна, можно использовать шлюзы. Шлюзы – это трансляторы протоколов, используемые для преобразования данных между различными промышленными протоколами. Например, если вы хотите подключиться к сети CC-Link, вы можете сделать это, используя шлюз. Шлюз может принимать Modbus TCP со стороны PAC и переводить в стандарт CC-Link.

Коммуникации Modbus

Modbus - наиболее распространенный в наши дни промышленный протокол. Он был разработан в 1979 и поддерживает как физический уровень последовательных портов, так и физический уровень Ethernet. Modbus - протокол сообщений уровня приложения для клиент-серверных коммуникаций между устройствами, соединенными шиной или сетью. Modbus может быть реализован с использованием асинхронной последовательной передачи данных для связи с сенсорными экранами, PLC и шлюзами, поддерживающими другие типы промышленных сетей.

Modbus может использоваться с CompactRIO и его встроенными последовательными или Ethernet портами. Заметим, что в CompactRIO встроен последовательный порт RS232, в то время, как некоторые устройства Modbus могут работать с электрическим уровнем RS485. В этом случае, может быть использован адаптер RS485 - RS232.

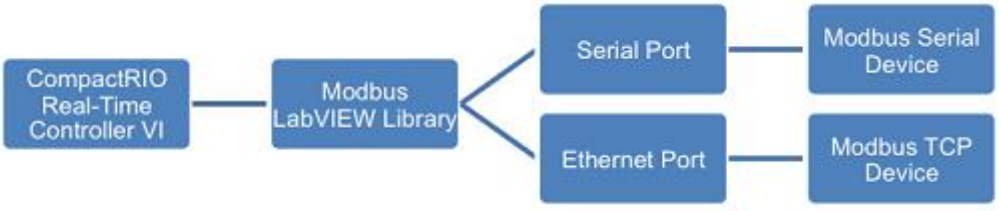


Рисунок 4.57. Программная архитектура коммуникаций с устройствами Modbus
CompactRIO Real-Time Controller VI – VI контроллера реального времени CompactRIO, Modbus LabVIEW Library – библиотека LabVIEW Modbus, Serial Port – последовательный порт, Ethernet Port – порт Ethernet, Modbus Serial Device –устройство Modbus с последовательным портом, Modbus TCP Device - устройство Modbus TCP

Протокол Modbus Serial основан на архитектуре клиент-сервер. Каждому подчиненному устройству назначается адрес от 1 до 247. Только одно главное устройство может быть в определенный момент подключено к шине. Подчиненные устройства не передают информацию, пока не получат запроса от главного устройства, и подчиненные устройства не могут связываться с другими подчиненными.

Информация передается между главным и подчиненным устройствами путем чтения и записи в регистры, расположенные в подчиненном устройстве.

Спецификации Modbus различают использование четырех таблиц регистров, в каждой до 65536 элементов, типы регистров и доступ на чтении и запись. В LabVIEW адреса таблиц регистров не перекрываются.

Таблицы	Тип объекта	Тип доступа	Комментарии
Discrete Inputs (Дискретные входы)	Одноразрядный	Только чтение	Главное устройство может только читать. Только подчиненное может изменять значение регистров.
Coils (Обмотки реле)	Одноразрядный	Чтение и запись	И главное, и подчиненное могут читать и записывать в эти регистры
Input Registers (Входные регистры)	16-битное слово	Только чтение	Главное устройство может только читать. Только подчиненное может изменять значение регистров.
Holding Registers (Регистры хранения)	16-битное слово	Чтение и запись	И главное, и подчиненное могут читать и записывать в эти регистры

Таблица 4.2. Передача информации между главным и подчиненным устройствами происходит путем чтения и записи в регистры, расположенные на подчиненном устройстве.

Более подробные сведения о Modbus можно найти на сайте ni.com. Вы можете также загрузить свободно распространяемую библиотеку Modbus для LabVIEW.

После инсталляции использование библиотеки Modbus аналогично использованию функций последовательных коммуникаций NI-VISA. Палитра устанавливается в субпалитру **Functions»User Libraries»NI Modbus** (Функции»Библиотеки пользователя»NI Modbus). Примеры приведены в третьем и четвертом столбцах палитры.

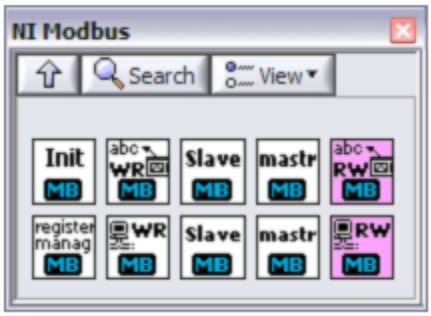


Рисунок 4.58. LabVIEW предлагает библиотеку Modbus для связи главного и подчиненных устройств

Прежде, чем начать программирование Modbus вы должны определить несколько важных параметров вашего устройства Modbus, сверяясь с документацией на устройство:

1. Устройство главное (Master) или подчиненное (Slave)? Если подчиненное, настройте ваш контроллер CompactRIO как главное устройство. Это наиболее распространенная конфигурация. Аналогично, подключение к главному устройству Modbus (например, другому PLC) требует конфигурирования CompactRIO, как подчиненного устройства.
2. Порт последовательный или Ethernet? Устройства Modbus Ethernet иногда называются также устройствами “Modbus TCP”.
3. Для Modbus устройства с последовательным портом:
   - RS232 или RS485? Многие устройства Modbus используют RS232, но некоторые – более мощную шину RS485 для передачи на большие расстояния. Устройства RS232 могут подключаться непосредственно к системе CompactRIO, а устройствам RS485 для правильного функционирования требуются преобразователя RS232-RS485.
   - Режим RTU или ASCII? RTU/ASCII – это способы представления данных для передачи по последовательной шине. Данные RTU - необработанные двоичные данные, в то время как данные в формате ASCII доступны для прочтения человеком. Этот параметр необходим для VI Modbus.
4. Каковы адреса регистров? Числовым адресам каждого устройства Modbus поставлены в соответствие его регистры ввода-вывода, обмотки реле и дискретные каналы для чтения и записи. Согласно соглашению Modbus, адрес регистра всегда на единицу меньше, чем имя

регистра. Это аналогично тому, что индекс первого элемента массива равен 0. VI из библиотеки Modbus в LabVIEW используют адреса, а не имена регистров.

Пример Modbus

В этом примере показано использование CompactRIO в наиболее распространенной реализации главного устройства Modbus RTU с последовательным портом

RS232. Для простоты примера параметры конфигурации не подключены, и поэтому используются значения параметров по умолчанию.

- Скорость передачи данных в бодах: 9600, без паритета, без управления потоком, тайм-аут 10 секунд
- Формат данных - RTU
- Адрес подчиненного устройства - 0

Если ваше устройство использует другие параметры, убедитесь, что подключены входы ко всем необходимым VI.

Этот пример показывает использование Modbus с последовательным портом, однако его легко адаптировать для работы с TCP/Ethernet путем замены VI их Ethernet-эквивалентами в соответствующей палитре.

В этом примере после запуска из устройства Modbus извлекаются последние значение температуры, масштабируются и помещаются в главную таблицу памяти. Затем считывается последнее выданное значение, масштабируется обратно в целое число и новое значения записывается в подчиненное устройство. Для интеграции этого примера в вашу управляющую архитектуру запустите этот цикл как еще одну параллельную задачу "driver". Входы и выходы устройства Modbus хранятся в локальной таблице памяти используемых переменных общего доступа типа Single Process с FIFO реального времени.

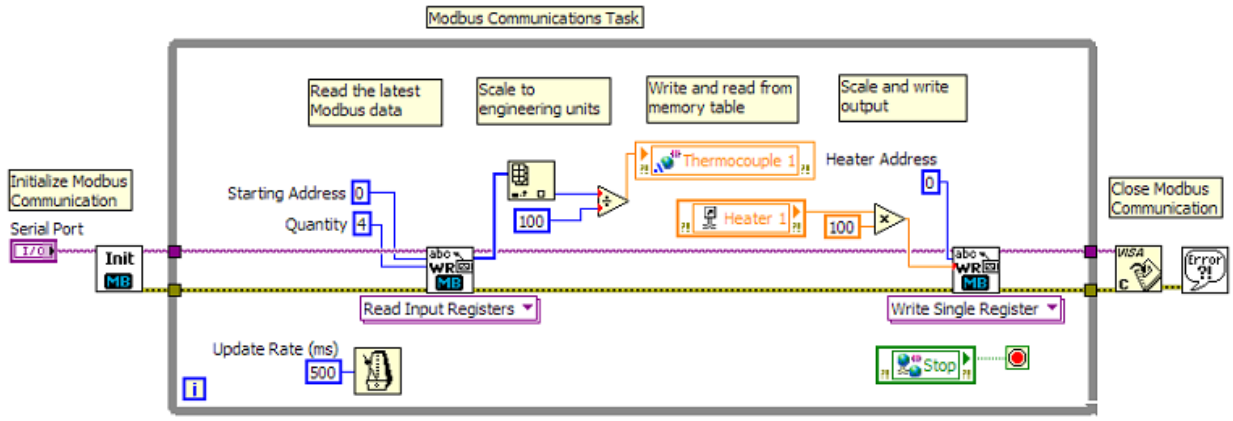


Рисунок 4.59 Пример обмена данными по протоколу Modbus в LabVIEW

Сначала вы должны выбрать последовательный порт. В CompactRIO это ASRL1::INSTR. Если целевым устройством является контроллер CompactRIO, это поле должно заполниться автоматически. Далее начинается главный цикл Modbus. В этом примере показано простое чтение и запись входных регистров. Регистры используются для многозначных данных в формате 16-разрядных целых чисел. Как правило, входные регистры читаются группами, так что нужно задать начальный адрес и количество последовательно расположенных адресов, которые нужно прочитать. Таким образом вы можете прочитать много входов за один вызов Read VI.

Например, первое возвращаемое значение индексируется из массива, масштабируется в инженерные единицы и помещается в таблицу памяти. В зависимости от типа датчика, калибровки и других факторов, параметры масштабирования определяются конечным пользователем. Как правило, пользователи помещают функции масштабирования в subVI.

Для записи самых последних выходных данных в сеть Modbus вызывается VI Modbus Write. Еще раз данные должны быть подвергнуты обратному масштабированию в формат целых чисел,

используемых в Modbus, а далее опубликованы по соответствующему адресу хоста. В примере показана запись в один регистр, однако существуют функции для обновления многих значений одновременно.

В примере скорость обновления цикла составляет 500 мс, однако она может быть изменена для достижения скорости обновлений, необходимых для оконечного устройства. Не забудьте, что максимальная частота обновлений зависит от количества читаемых/записываемых регистров/обмоток реле, скорости передачи данных в бодах и возможностей оконечного устройства. Вообще устройства Modbus/TCP могут обновляться быстрее, чем устройства Modbus с последовательным портом. Обратите внимание, что излишне малые времена обновления (менее 10 мс) могут чрезмерно расходовать время процессора CompactRIO и оставлять меньше процессорного времени на другие задачи.

EtherNet/IP

Еще один очень распространенный промышленный протокол - EtherNet/IP. EtherNet/IP - это протокол, часто используемых в современных PLC компании Rockwell (Allen-Bradley) и использующий стандартное подключение Ethernet для связи с промышленными устройствами ввода- вывода. У NI есть библиотека, доступная по адресу ni.com/labs, позволяющая CompactRIO непосредственно читать и записывать тэги из PLC, используя сообщения в явном виде (Explicit Messaging), или же быть полным адаптером EtherNet/IP класса 1 (подчиненным устройством). Хотя библиотека – лабораторный продукт, она создана на основе верифицированного стека протоколов EtherNet/IP и была должным образом протестирована. Инсталлируйте драйвер, запустив программу установки.

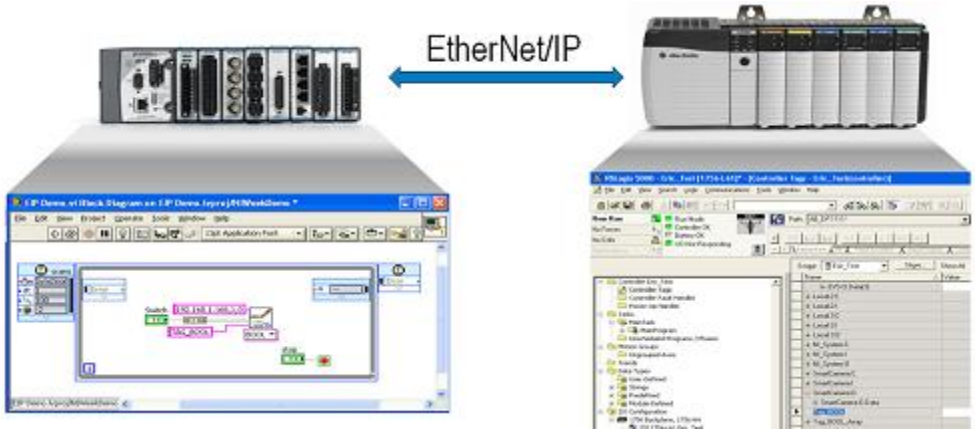


Рисунок 4.60. С EtherNet/IP вы можете напрямую читать и записывать тэги в PLC Rockwell из CompactRIO

Драйвер EtherNet/IP интегрируется в управляющую архитектуру с помощью той же технологии, что и библиотека Modbus. Создайте другой параллельный цикл в качестве задачи "driver". Передавайте данные из сети EtherNet/IP в таблицу памяти с помощью переменных общего доступа типа Single Process с FIFO реального времени.

OPC

OPC, или OLE for Process Control – распространенный протокол, используемый во многих приложениях с большим количеством каналов и низкими скоростями обновления, применяемый, например, в нефтегазовой и фармакологической промышленности. OPC разработан для подключения HMI и SCADA-систем к контроллерам – обычно он не используется для связи между контроллерами. Как правило, OPC используется в приложениях HMI и SCADA при интеграции контроллера CompactRIO в систему сторонней фирмы.

Спецификация OPC - обширное собрание стандартов, которые охватывают многие приложения. Это обсуждение фокусируется на OPC Data Access (Доступ к данным), основной спецификации, определяющей, как универсально перемещать данные, используя распространенные технологии Windows.

Сетевая переменная общего доступа обеспечивает шлюз LabVIEW к OPC.

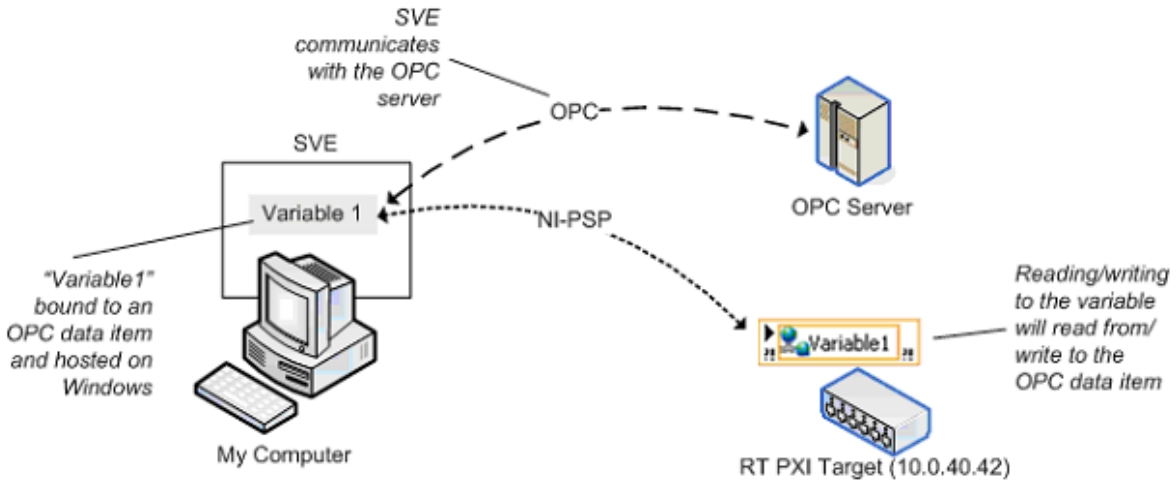


Рисунок 4.61. Публикуемая в сети переменная общего доступа предоставляет шлюз к OPC.
Variable 1 – переменная 1, My Computer – мой компьютер, RT PXI Target – целевой объект реального времени

Переменная 1 привязана к элементу данных OPC и обслуживается в Windows.

SVE связывается с сервером OPC. Чтение/запись в переменную эквивалентно чтению/записи в элемент OPC

OPC - технология только для Windows, и CompactRIO не может напрямую связываться по протоколу OPC. Вместо этого Windows PC может связываться с CompactRIO, используя протокол PSP, и может транслировать и переопубликовывать информацию по OPC. Механизм продвижения переменных общего доступа, реализованный на компьютере, выполняет функции OPC-сервера, автоматически транслируя и повторно публикуя переменные.

Это означает, что другие OPC-клиенты, подобные SCADA-пакетам сторонних фирм или программы управления процессом могут получать доступ и извлекать данные из устройства CompactRIO.

Публикация данных из системы CompactRIO через OPC

В следующем примере показано, как использовать окна механизм продвижения переменных общего доступа для того, чтобы сделать значения доступными на OPC-сервере. В этом примере предполагается, что вы уже публикуете данные в сети по протоколу PSP (Network-Published Shared Variables), хостом которого является система CompactRIO. В этом примере сетевая переменная общего доступа называется "SV_PID_SetPoint."

1. Windows PC также является хостом для публикуемых в сети переменных общего доступа. Любая основанная на Windows сетевая переменная общего доступа по умолчанию публикуется и как элемент OPC.
2. Переменные Windows привязаны к данным системы CompactRIO, поскольку механизм продвижения переменных Windows является подписчиком, получает данные из системы CompactRIO и обновляет элементы OPC.
3. Другие OPC-клиенты подключены к Windows PC и используют эти данные для отображения, HMI и т.п.
4. 4. CompactRIO публикует данные в сети, используя публикуемые в сети переменные общего доступа, хостом которых является контроллер CompactRIO.

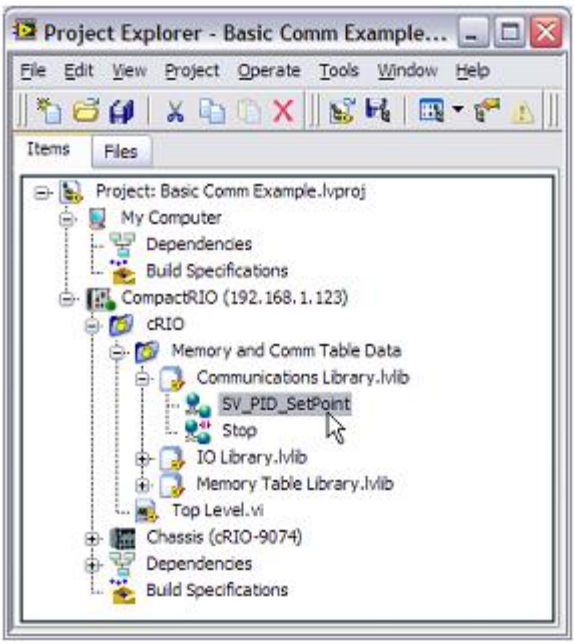


Рисунок 4.62. Публикуемые в сети переменные общего доступа обслуживаются системой CompactRIO

5. Windows PC также является хостом для публикуемых в сети переменных общего доступа. Любая основанная на Windows сетевая переменная общего доступа по умолчанию публикуется и как элемент OPC. В проекте LabVIEW, переменные общего доступа Windows находятся в папке "My computer". Создайте переменную общего доступа, щелкнув правой кнопке мыши по строке "My computer " и выбрав контекстном меню команду New»Variable. (Новая»Переменная)

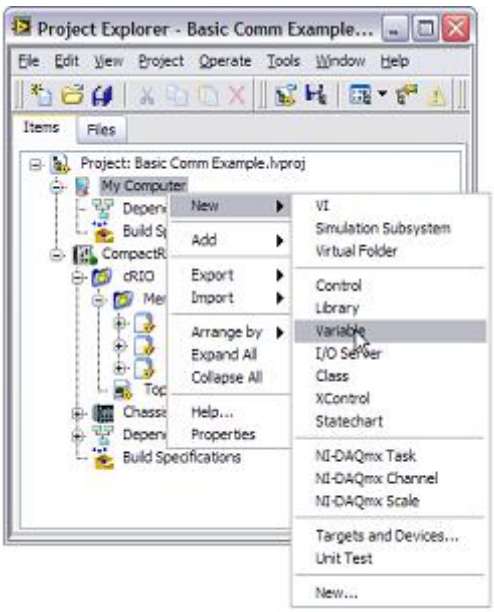


Рисунок 4.63. Публикуемые в сети переменная общего доступа, обслуживаемые ПК с Windows, автоматически публикуются через OPC

6. Переменные Windows являются "псевдонимами" для данных CompactRIO, поскольку механизм продвижения переменных Windows является подписчиком, получает данные из системы CompactRIO и обновляет элементы OPC. В окне свойств переменной общего доступа назначьте переменной осмысленное имя, например, "SV_PID_SetPoint_OPC". Установите флажок в окошке Enable Aliasing и нажмите кнопку Browse (Просмотр) для наблюдения переменной SV_PID_SetPoint в CompactRIO.

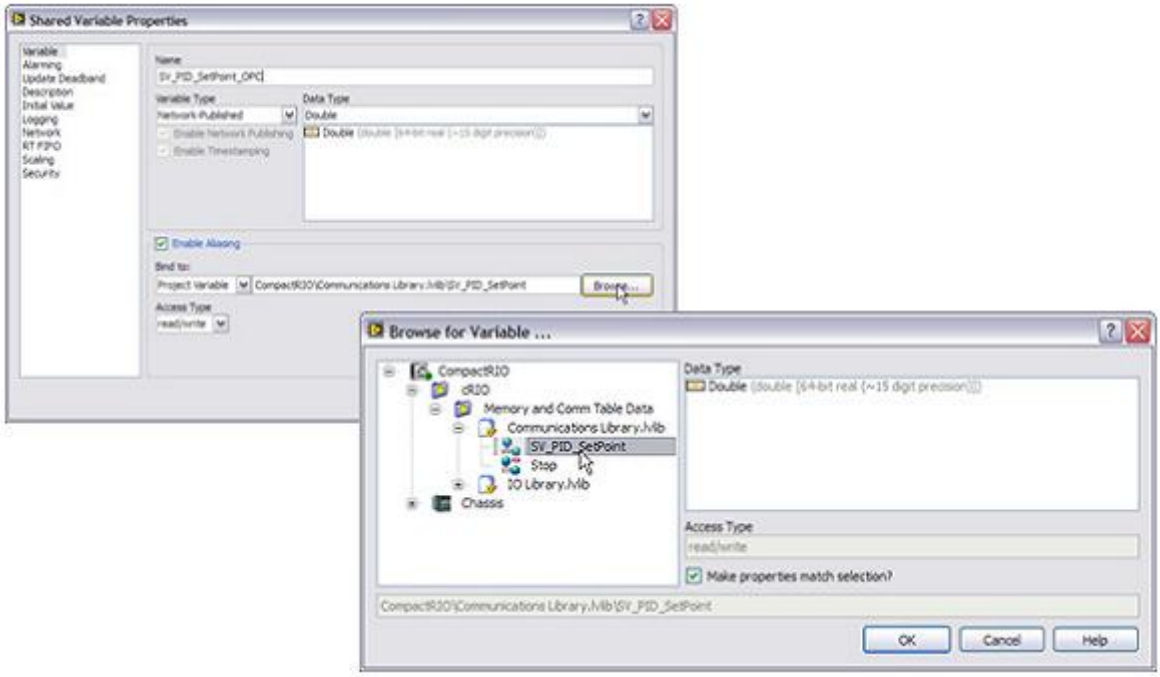


Рисунок 4.64. Именованная и публикуемая в сети переменная общего доступа предоставляет шлюз между CompactRIO и OPC

7. Сохраните вновь созданную библиотеку под именем "OPC Library.lvlib."
8. Разверните переменные в механизме продвижения переменных общего доступа, щелкнув по имени библиотеки правой кнопкой мыши и выбрав команду Deploy.
9. Переменная общего доступа теперь опубликована как элемент OPC. Вы можете убедиться, что переменная работает верно, запустив OPC-клиент, например, менеджер распределенных систем NI (меню **Start»Programs»National Instruments»NI Distributed System Manager**).

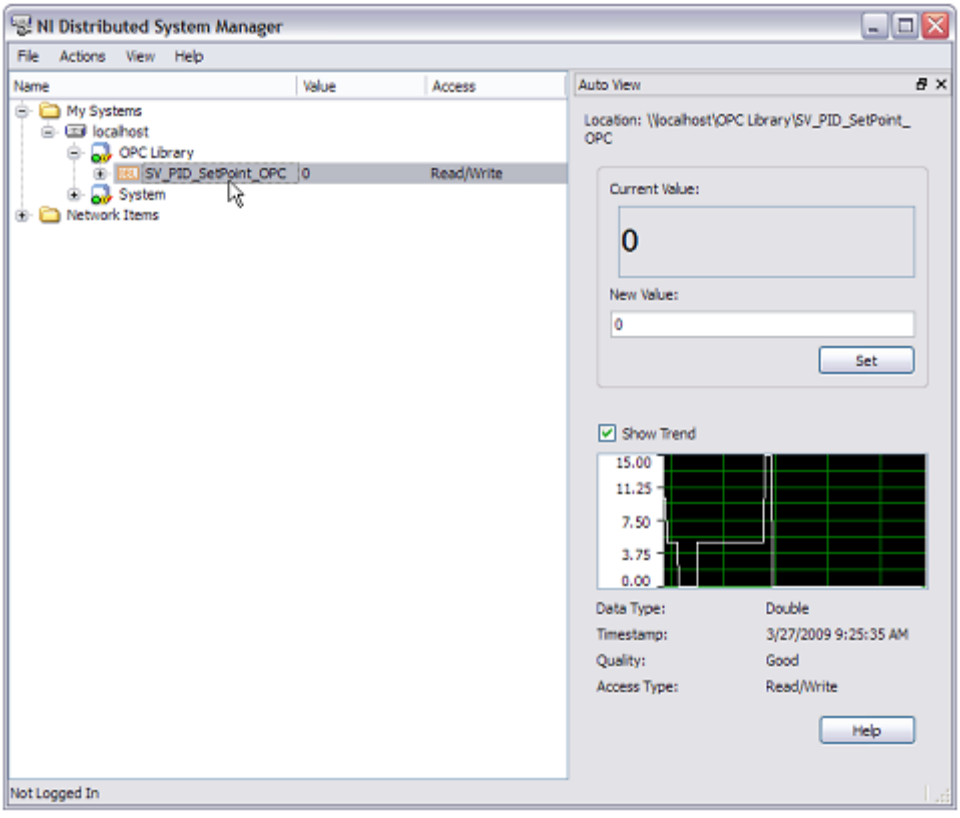


Рисунок 4.65. OPC-клиенты, подобные NI Distributed System Manager, могут читать и записывать в элемент OPC

10. Если у вас есть доступный OPC-клиент, например, модуль LabVIEW DSC, вы можете еще раз убедиться, что переменная работает, и из этого клиента. Элементы OPC, ассоциируемые с переменными общего доступа, публикуются сервером National Instruments.Variable Engine.1.
11. Как только переменные развернуты в механизме продвижения переменных общего доступа, они останутся развернутыми и при последующих перезагрузках сервера OPC.

Дополнительная информация о механизме продвижения переменных общего доступа и OPC находится на сайте ni.com.

РАЗДЕЛ 5
Расширение системы ввода-вывода CompactRIO

Добавление ввода-вывода в CompactRIO

Типичная система CompactRIO может содержать в корпусе до 8 модулей C-серии, но в некоторых управляющих приложениях требуется большее количество каналов ввода-вывода, или же нужны рассредоточенные относительно главного контроллера каналы ввода-вывода. Вы можете использовать два простых метода расширения вашей системы:

- Средства ввода-вывода через Ethernet
- Средства детерминированного ввода-вывода через Ethernet

Ввод-вывод через Ethernet

Расширение системы CompactRIO с использованием средств ввода-вывода через Ethernet заключается в добавлении одной или более систем CompactRIO в ту же сеть. Главный контроллер отвечает за выполнение регулирования в цикле реального времени, используя каналы ввода-вывода двух различных систем. Контроллер расширения, который либо не выполняет никаких логических функций, либо реализует только простую логику сетевого сторожевого таймера, обеспечивает расширенный или рассредоточенный ввод-вывод для главного контроллера.

Программисты могут использовать существующую сетевую инфраструктуру, а именно коммутаторы и маршрутизаторы. Хотя полнодуплексные сетевые коммутаторы исключают коллизии пакетов, однако они вызывают джиттер, потому обычный Ethernet должен использоваться только в приложениях, где не требуется детерминированных коммуникаций. Если же требуется синхронизация между локальным и расширенным вводом-выводом, обратитесь к разделу "Детерминированный ввод-вывод через Ethernet" для получения дополнительной информации.

Использование другой системы CompactRIO в качестве подсистемы ввода-вывода через Ethernet возможно с применением нескольких различных архитектур, среди которых:

1. Код LabVIEW выполняется на хост-компьютере, а две системы CompactRIO обеспечивают ввод-вывод и/или дополнительную обработку
2. Оба контроллера - главный и контроллер расширения выполняют встроенную логику реального времени и логику FPGA и делятся информацией через псевдонимы ввода-вывода или переменные общего доступа

Шаг 1. Конфигурирование системы расширения

Простейший способ добавления еще одной системы CompactRIO в качестве подсистемы ввода-вывода через Ethernet предполагает подключение двух систем CompactRIO к одной сети. Подробные инструкции по конфигурированию каждого контроллера и его IP-адреса приведены ниже в этом документе в разделе "Введение в CompactRIO".

Рисунок 5.1. Простейший способ добавления еще одной системы CompactRIO в качестве подсистемы ввода-вывода через Ethernet предполагает подключение двух систем CompactRIO к одной сети

Как только у обоих контроллеров будут IP-адреса, добавьте оба контроллера в проект LabVIEW. Выберите для обоих режим программирования Scan Interface (интерфейс сканирования). В окне LabVIEW Project Explorer (Обозреватель проекта LabVIEW) автоматически появятся все модули и каналы ввода-вывода главной системы и системы расширения.

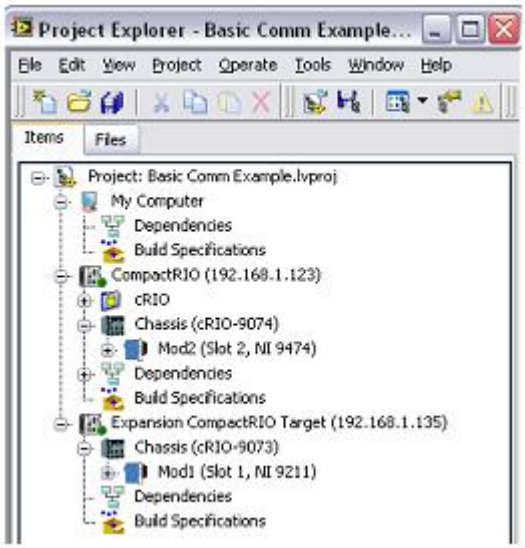


Рисунок 5.2. В окне Обозревателя проекта LabVIEW показаны две системы CompactRIO, соединенные по TCP/IP

Данные ввода-вывода из контроллера расширения автоматически публикуется по протоколу PSP. Вы можете непосредственно перетащить элементы ввода-вывода на блок-диаграмму главного контроллера для чтения и записи из контроллера расширения. Вы можете настроить частоту передачи обновленных данных из шасси расширения в сеть, изменив период публикации данных сети. Щелкните правой кнопкой мыши по строке контроллера и выберите Properties (Свойства). В окне свойств выберите **Scan Engine** (механизм сканирования) слева в меню для настройки периодов.

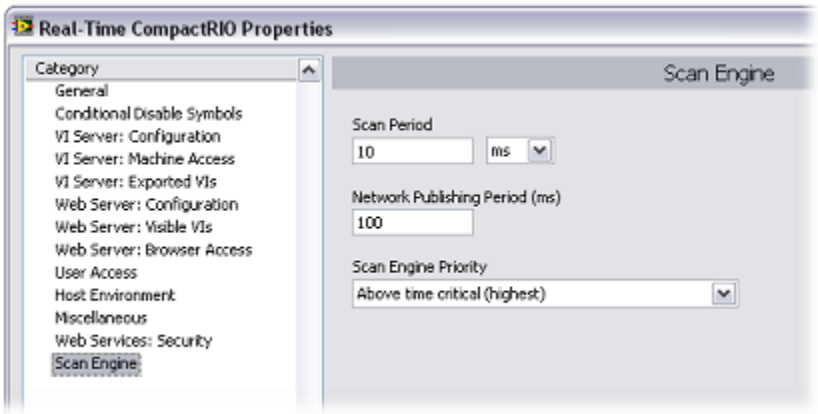


Рисунок 5.3. Вы можете настроить период сканирования и период опубликования данных в сети в свойствах контролера CompactRIO

Соображения по поводу отключения от сети Ethernet

Поскольку контроллер расширения разработан для выполнения встроенного программного приложения реального времени, сетевой сторожевой таймера в нем не используется. Это значит, что если соединение между главным контроллером и контроллером расширения разорвется, выходы останутся в последнем состоянии. Если это неприемлемо, вам нужно написать небольшое приложение реального времени для контроллера расширения, которое будет наблюдать за тактовыми импульсами главного контроллера и устанавливать выходы в безопасное состояние при разрыве соединения.

Шаг 2. Добавление ввода-вывода в процесс сканирования главного контроллера

В программе, выполняемой на главном контроллере, считывайте каналы ввода-вывода и добавьте их в процесс сканирования ввода-вывода. Ранее обсуждалось, как добавлять данные в процесс сканирования ввода-вывода из источников, подобных Ethernet. Однако ниже приведен краткий обзор.

Асинхронный недетерминированный ввод-вывод

Поскольку вы считываете данные по стандартному Ethernet, этот ввод-вывод не является детерминированным. В этом случае вы должны создать обычный цикл while или синхронизируемый цикл с низким приоритетом. Это позволяет задаче управления выполняться с высоким детерминизмом и надежностью, поскольку на нее не влияет потенциально высокий джиттер устройства ввода-вывода в пользовательской задаче сканирования ввода-вывода. Установите время выполнения задачи, исходя из требуемой частоты обновлений.

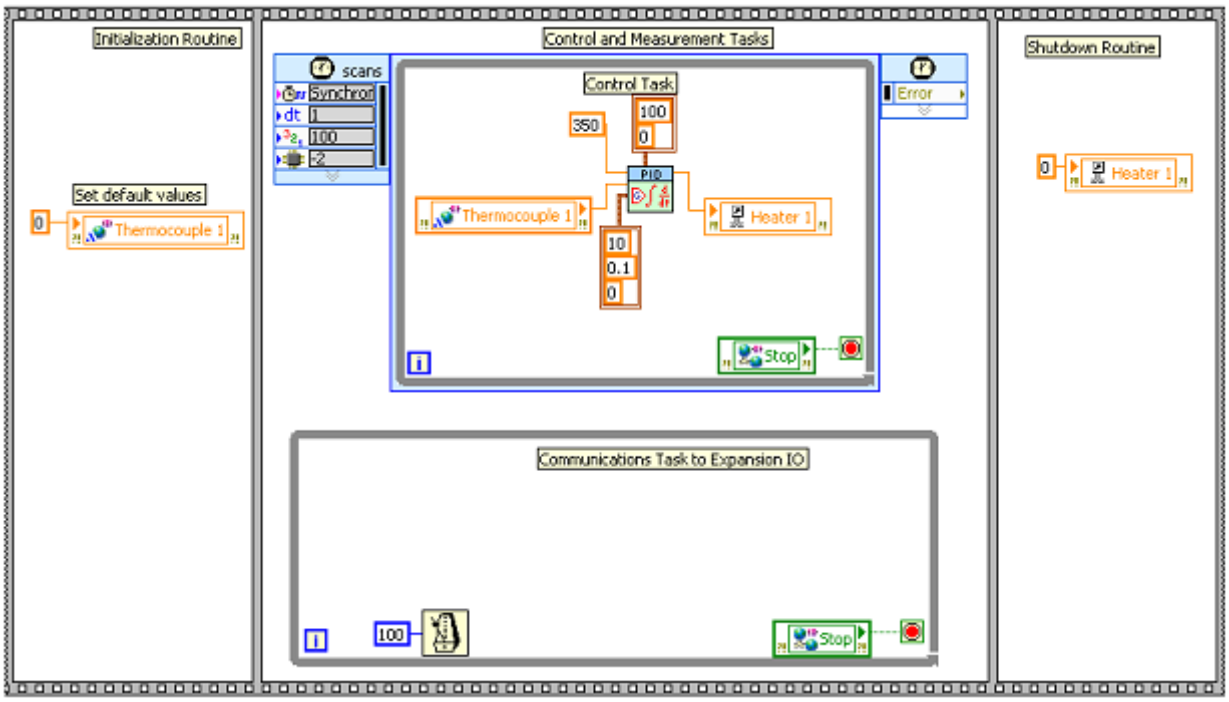


Рисунок 5.4. Добавьте цикл или синхронизируемый цикл для задачи сканирования расширенного ввода-вывода

Шаг 3. Копирование данных в табличную память

Чтобы обеспечить в приложении доступ к данным чтения и записи из пользовательской задачи сканирования ввода-вывода, создайте переменные общего доступа типа Single Process с FIFO реального времени, как обсуждалось в разделе "Обмен данными между задачами".

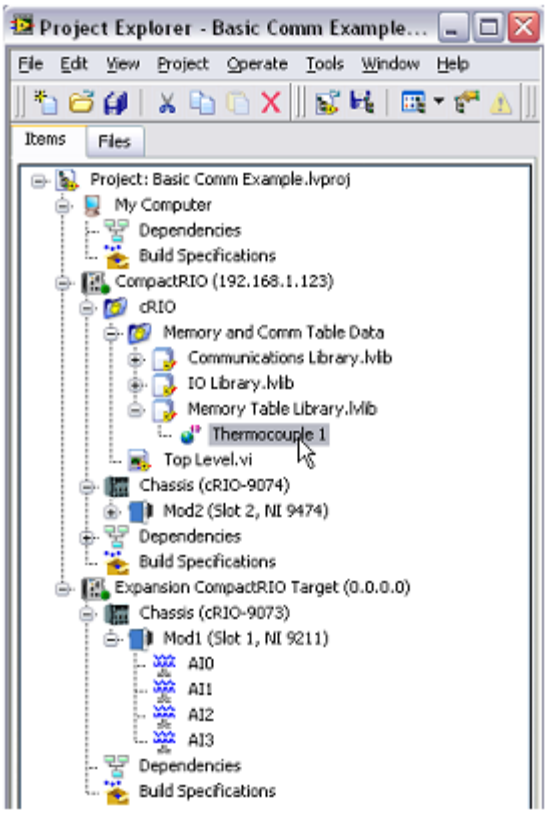


Рисунок 5.5. Использование переменной общего доступа типа Single Process для разделения данных пользовательского ввода-вывода с другими задачами

Совет: Для систем с большим количеством каналов вы можете ускорить создание множества псевдонимов ввода-вывода, экспортировав таблицу.csv с использованием редактора множества переменных (Multiple Variable Editor).

В цикле считывайте переменные из сети, проверяйте на предупреждения или ошибки, и записывайте данные в переменные общего доступа типа Single Process.

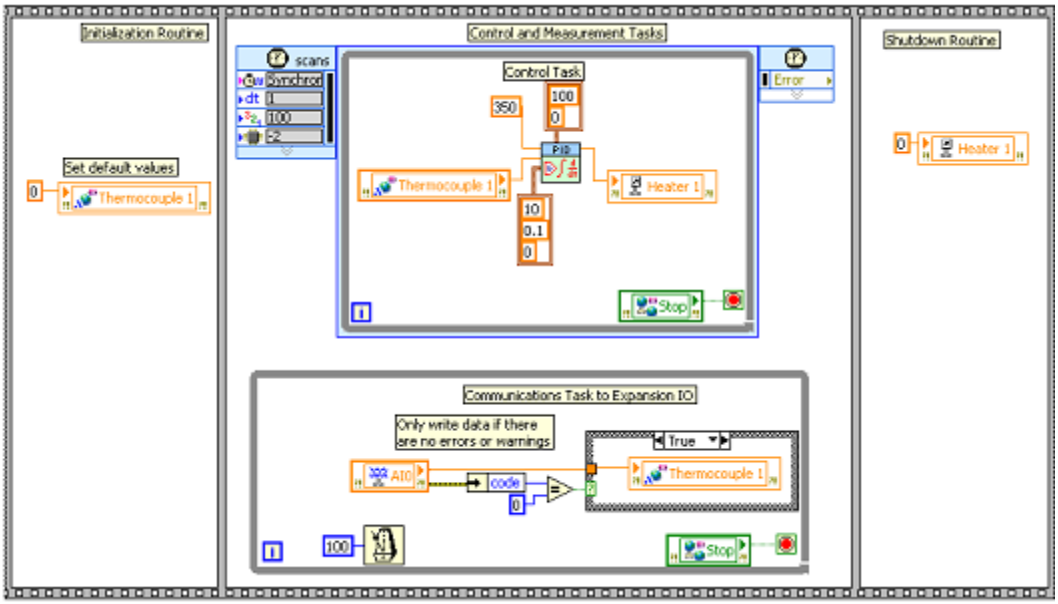


Рисунок 5.6. В задаче коммуникаций считывайте данные из шасси расширения и записывайте их в табличную память, используя переменные общего доступа типа Single Process

Детерминированный ввод-вывод через Ethernet

В данном разделе приведен пример кода LabVIEW

В некоторых приложениях системы главного и расширенного ввода-вывода требуют жесткой синхронизации, например, чтобы все входы и выходы обновлялись одновременно. Использование детерминированной шины позволяет главному контроллеру не только знать, когда обновляется расширенный ввод-вывод, но и сколько времени потребуется на получение данных. Вы можете легко сделать распределенную систему CompactRIO с технологией детерминированного Ethernet, используя шасси расширения NI 9144.

Краткие сведения о NI 9144 – шасси детерминированного Ethernet

NI 9144 - шасси расширения систем CompactRIO для работы в жестких условиях с использованием протокола детерминированного Ethernet, называемого EtherCAT. Архитектура главный-подчиненный позволяет использовать любой контроллер CompactRIO с двумя портами Ethernet как главное устройство, а NI 9144 – как подчиненное. NI 9144 также имеет два порта, с помощью которых можно реализовать цепочечное подключение контроллера для расширения приложений, критичных к времени.

Рисунок 5.7. Архитектура аппаратных средств CompactRIO расширяет критические по времени системы с использованием детерминированного Ethernet и шасси NI 9144

Это шасси с восемью слотами поддерживает все модули C-серии для аналоговых, цифровых, и специальных измерений, а также управления движением. Вы можете программировать шасси расширения NI 9144 при помощи модулей LabVIEW Real-Time и LabVIEW FPGA.

Программирование в режиме сканирования с LabVIEW Real-Time обеспечивает возможность простого и рационального расширения ввода-вывода. Переменные ввода-вывода в режиме сканирования дают вам мгновенный доступ к физическому вводу-выводу при помощи простой технологии drag-and-drop с возможностями контроля производительности системы и улучшенной диагностики неисправностей с помощью тестовых панелей. В шасси имеется также встроенный сетевой сторожевой таймер, который в сканирующем режиме устанавливает выходы в значение по умолчанию (0 В или off) при отключении от главного устройства.

Программирование распределенного ввода-ввода в режиме FPGA открывает совершенно новый уровень адаптации и гибкости в соответствии с требованиями вашего приложения. Благодаря встроенным возможностям принятия решений уменьшается время реакции, ускоряется взаимодействие с внешней средой без вмешательства хоста. Интеллектуальный расширенный ввод-вывод может также разгружать контроллер, выполняя текущую обработку, пользовательский запуск и манипуляции с сигналами в узле. Кроме того, использование LabVIEW FPGA дает возможность экспертам в определенных областях быстро создавать прототипы и реализовывать свои идеи без необходимости быть программистами VHDL. Для получения дополнительных сведений о программировании в LabVIEW FPGA обратитесь к разделу 6, "Разработка специализированных аппаратных средств с помощью LabVIEW FPGA".

Рисунок 5.8. Шасси расширения NI 9144

Шаг 1. Конфигурирование шасси для детерминированного расширения

Для разрешения поддержки EtherCAT в контроллере CompactRIO (NI cRIO-9074 в этом примере), вы должны установить драйвер NI-Industrial Communications for EtherCAT на хост-компьютер. Драйвер NI-Industrial Communications for EtherCAT доступен на CD, поставляемом с шасси 9144, его можно также свободно скачать с сайта ni.com

Рисунок 5.9. Для конфигурирования системы с детерминированным Ethernet подключите порт 1 контроллера cRIO-9074 к хост-компьютеру, а порт 2 – к шасси NI 9144

Конфигурирование контроллера реального времени CompactRIO как Deterministic Bus Master (Главный на детерминированной шине)

1. Запустите утилиту Measurement & Automation Explorer (MAX) и подключитесь к контроллеру CompactRIO.
2. В MAX раскройте контроллер в разделе **Remote Systems** (Удаленные системы) на панели **Configuration** (Конфигурация). Щелкните правой кнопкой по строке **Software** (Программное обеспечение) и выберите пункт **Add/Remove Software** (Добавить/удалить ПО).

Рисунок 5.10. Инсталляция необходимого ПО в Measurement & Automation Explorer

3. Если в контроллере уже установлены LabVIEW Real-Time и NI-RIO, выберите **Custom software installation** (Установка ПО пользователем) и нажмите **Next** (Далее). Нажмите **Yes** (Да), если появится окно предупреждения. Щелкните в окошке рядом с **IndCom for EtherCAT Scan Engine Support**. Требуемые дополнительные средства проверяются

автоматически. Щелкните **Next** (Далее) для продолжения инсталляции программного обеспечения на контроллер.

4. Когда установка программного обеспечения завершится, выберите контролер в разделе Remote Systems (Удаленные системы) на панели Configuration (Настройка). Щелкните по кнопке **Advanced Ethernet Settings** (Расширенные настройки Ethernet). В окне Ethernet Devices (Устройства Ethernet) выберите вторичный MAC-адрес (тот, про который не сказано primary). В разделе Ethernet Device Settings (Настройки устройств Ethernet) выберите **EtherCAT** в выпадающем списке **Mod**e (Режим) и щелкните **OK.**

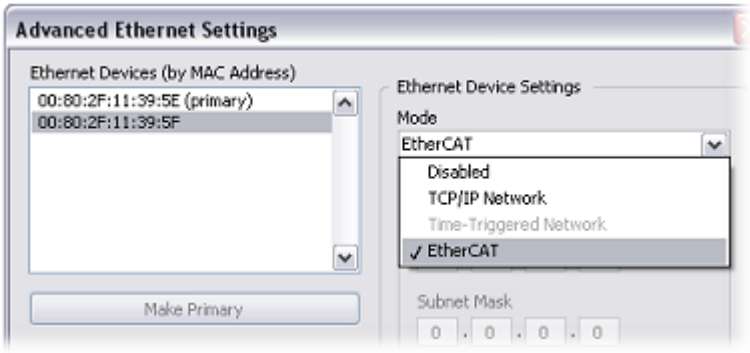


Рисунок 5.11. Выберите режим "EtherCAT" для второго порта Ethernet контроллера CompactRIO

Шаг 2. Добавление детерминированного ввода-вывода в процесс сканирования

1. Подключите хост-компьютер и порт 1 контролера CompactRIO к одной и той же сети Ethernet. Используйте стандартный кабель Ethernet для прямого подключения порта 2 контролера CompactRIO к порту IN шасси расширения NI 9144. Для добавления дополнительных шасси NI 9144 используйте кабель Ethernet для подключения порта OUT предыдущего NI 9144 к порту IN следующего NI 9144.
2. В окне обозревателя проекта LabVIEW щелкните правой кнопкой мыши по контроллеру CompactRIO и выберите **New»Targets and Devices** (Новый»Целевые объекты и устройства)
3. В диалоговом окне Add Targets and Devices (Добавить целевые объекты и устройства) раскройте категорию **EtherCAT Master Devic**e (Главное устройство EtherCAT) для автоматического обнаружения порта EtherCAT на главном контроллере. В появившемся диалоговом окне Scan Slaves (Сканирование подчиненных) выберите первый вариант для автоматического обнаружения любых подчиненных устройств, подключенных к контроллеру. Щелкните по кнопке OK. Проект LabVIEW теперь включает список, перечисляющий главный контролер, шасси NI 9144, их модули ввода-вывода и физический ввод-вывод каждого модуля.

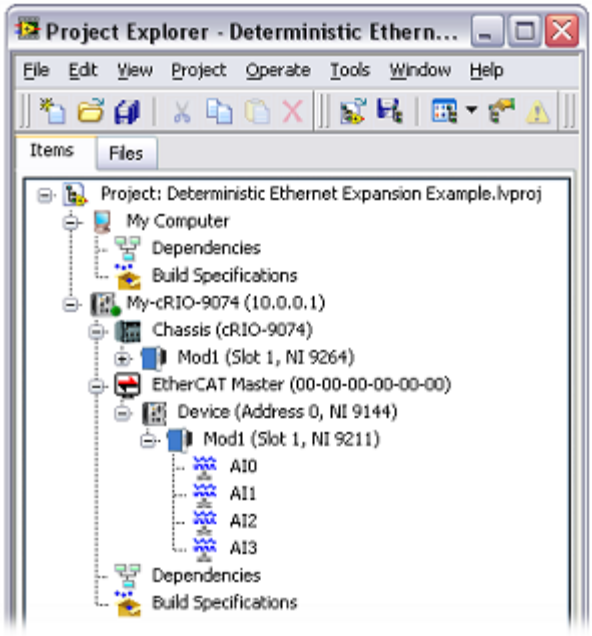


Рисунок 5.12. В проект LabVIEW включены главный контроллер, шасси NI 9144 и модули ввода-вывода

4. Наконец, создайте псевдонимы ввода-вывода, соответствующие физическим каналам, и используйте их для доступа к вводу-выводу расширения. Механизм сканирования автоматически управляет синхронизацией ввода-вывода так, чтобы все модули читались и обновлялись в одно и то же время в каждом цикле сканирования.

Совет: Для систем с большим количеством каналов вы можете ускорить создание множества псевдонимов ввода-вывода, используя редактор множества переменных (Multiple Variable Editor) и экспортируя таблицу .csv.

Шаг 3. Добавление интеллекта FPGA в шасси детерминированного расширения

1. В окне обозревателя проекта LabVIEW щелкните правой кнопкой мыши по устройству (NI 9144) и выберите New»FPGA Target (Новый»Целевое устройство FPGA) для создания нового целевого объекта FPGA в вашем NI 9144.

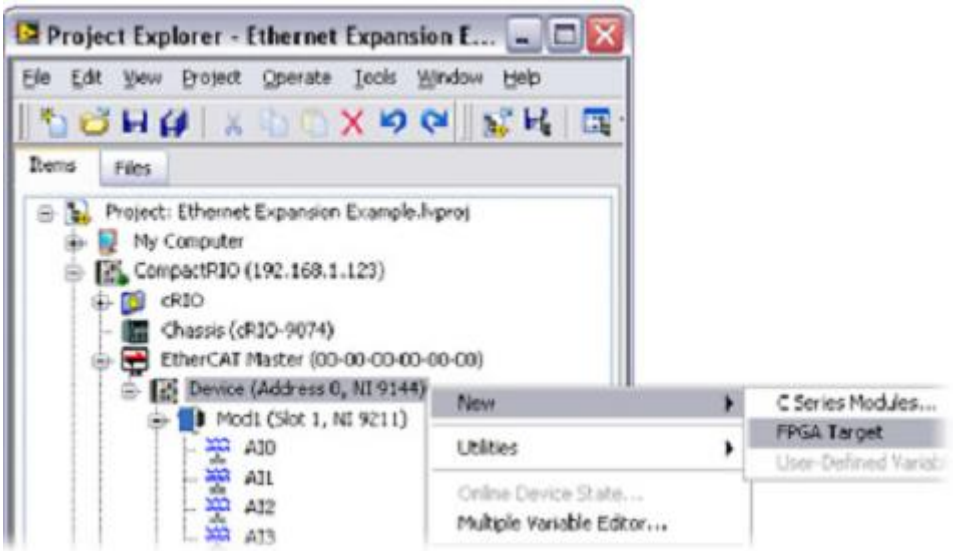


Рисунок 5.13. Добавьте целевой объект FPGA в шасси NI 9144

2. Для использования модуля в режиме LabVIEW FPGA перетащите модуль из NI 9144 в FPGA Target. В окне обозревателя проекта вы можете перетаскивать модули между Device (Устройством) и FPGA Target (Целевое устройство FPGA) для переключения между режимом сканирования и режимом FPGA.

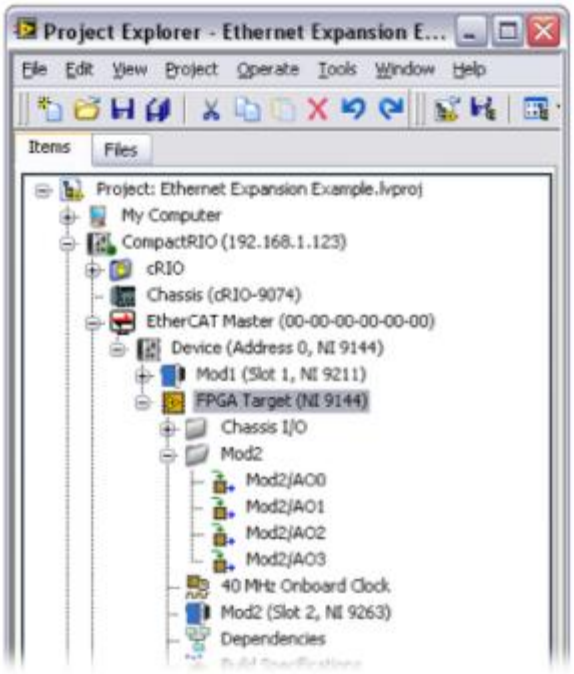


Рисунок 5.14. Перетащите модули в FPGA Target для программирования их в режиме LabVIEW FPGA

3. В окне обозревателя проекта LabVIEW щелкните правой кнопкой мыши по объекту FPGA Target (NI 9144) и выберите New»VI (Новый»VI) для создания нового VI FPGA на NI 9144. Вы можете перетащить каналы ввода-вывода FPGA из модулей, перечисленных под FPGA Target (NI 9144), на блок-диаграмму этого нового VI FPGA.

Различие в программировании с LabVIEW FPGA API

Если вы знакомы с программированием в LabVIEW FPGA для CompactRIO и других платформ NI RIO, вы должны знать о некоторых отличиях между программированием локального FPGA и FPGA для расширенного ввода-вывода. Начиная с LabVIEW 2009, определяемые пользователем переменные ввода-вывода используются для синхронизации данных FPGA с механизмом сканирования NI. Эти определяемые пользователем переменные ввода-вывода предназначены только для передачи данных между VI контроллера реального времени и VI FPGA шасси расширения NI 9144. Для получения дополнительной информации об определяемых пользователем переменных ввода-вывода обратитесь к разделу 6.

Методы обмена данными с FPGA	**Локальное шасси**	**Шасси расширения**
Определяемые пользователем переменные ввода-вывода	√	√
Хост-интерфейс FPGA	√	-
Функции прямого доступа к памяти (DMA)	√	-
Отладка FPGA с лицевой панели	√	-

Таблица 5.1. Методы обмена данными с FPGA для локальных приложений и шасси расширения

1. Как только вы добавили целевое устройство FPGA к NI 9144 в проекте LabVIEW, вы можете создать определяемую пользователем переменную ввода-вывода, щелкнув правой кнопкой мыши по NI 9144 и выбрав **Add»User-Defined Variable** (Добавить»Определяемая

пользователем переменная). Имя переменной, тип данных и направление данных (от хоста к FPGA или наоборот) может быть задано в окне свойств.

2. Для запуска FPGA VI щелкните правой кнопкой по cRIO-9074 в проекте LabVIEW и выберите **Deploy All** для развертывания кода в шасси NI 9144. Переключите механизм сканирования в режим конфигурирования, щелкнув правой кнопкой мыши по контроллеру CompactRIO и выбрав **Utilities»Scan Engine Mode»Switch to Configuration** (Утилиты»Режим механизма сканирования»Переключение в режим конфигурирования) и щелкните по кнопке Run (Запуск) в FPGA VI. При этом начнется процесс компиляции и загрузка битового файла FPGA в EtherCAT. Вам может понадобиться обновить встроенную программу вашего шасси для запуска пользовательского кода FPGA. Если вы получите сообщение об ошибке, заключающееся в том, что вам нужно обновить вашу встроенную программу, следуйте инструкциям в руководстве пользователя NI 9144 в разделе "Обновление встроенной программы". Вы можете найти руководство пользователя, запустив поиск на сайте ni.com.
3. Когда процессы компиляции и загрузки завершатся, вам нужно вернуть механизм сканирования в активный режим для выполнения FPGA VI. Щелкните правой кнопкой мыши по контролеру CompactRIO и выберите **Utilities»Scan Engine Mode»Switch to Active** (Утилиты»Режим механизма сканирования»Переключение в активный режим).

Обратите внимание, что существует предельное количество определяемых пользователем переменных ввода-вывода, которые могут быть созданы в режиме FPGA. NI 9144 может содержать всего 512 байт входных данных и всего 512 байт выходных данных одновременно для переменных ввода-вывода в режиме сканирования и переменных ввода-вывода определяемых пользователем в режиме FPGA. Например, если вы используете четыре 32-канальных модуля в режиме сканирования, и каждый канал занимает 32 бита данных, то 256 байт входных данных используется переменными ввода-вывода режима сканирования. С оставшимися 256 байтами входных данных вы можете создать 64 входных определяемых пользователем переменных ввода-вывода (также длиной 32 бита) для режима FPGA.

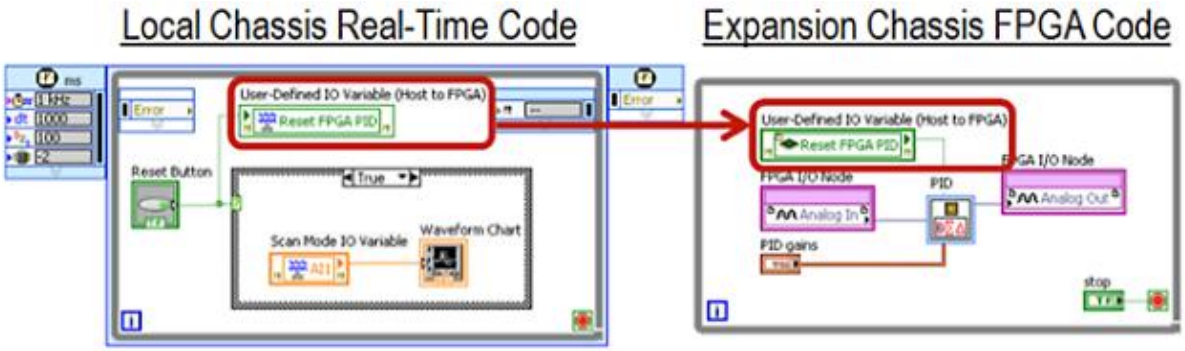


Рисунок 5.15. Создайте определяемые пользователем переменные ввода-вывода для обмена данными между VI реального времени и FPGA VI

Local Chassis Real-Time Code – код реального времени локального шасси, Expansion Chassis FPGA Code – FPGA-код шасси расширения

Помимо требований к реализации переменных ввода-вывода, определяемых пользователем для шасси расширения, вы можете программировать FPGA шасси расширения с любыми функциями алгоритмов LabVIEW FPGA. Кроме того, шасси NI 9144 предоставляет несколько дополнительных сигналов ввода-вывода шасси для настройки и синхронизации вашего кода. Например, вы может выполнять некую логику FPGA во время одного из семи состояний EtherCAT. Таким образом, если связь с главным контроллером нарушается, шасси NI 9144 может перейти в другое рабочее состояние, используя ваш код FPGA. Примеры LabVIEW включены в драйвер NI-Industrial Communications for EtherCAT для демонстрации расширенных возможностей FPGA, таких, как синхронизация механизма сканирования, специальных цифровых сигналов и асинхронная передискретизация.

Контроль средствами машинного зрения

Машинное зрение – комбинация сбора изображений из одной или более промышленных камер и обработки полученных изображений. Эти изображения обычно обрабатываются с использованием библиотеки функций обработки изображений, возможности которых варьируются от простого обнаружения границ объекта до чтения различного типа текста или сложных кодов.

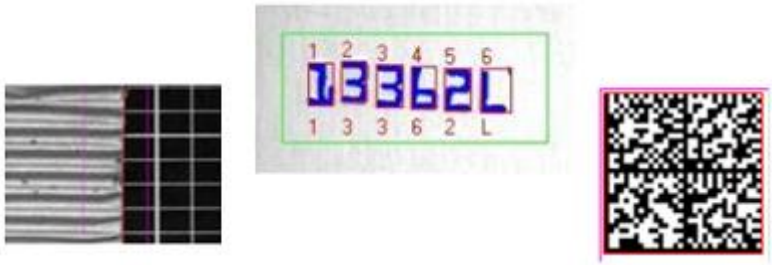


Рисунок 5.13. Задачи машинного зрения: обнаружение границ, оптическое распознавание символов и чтение двумерного кода

Зачастую в одной системе машинного (технического) зрения выполняется различные виды этих измерений для одного или более изображений. Вы можете это использовать для многих приложений, например, чтобы удостовериться, что содержимое контейнера соответствует этикетке на бутылке, или что код напечатан в нужном месте стикера.

Информация об обработанных изображениях поступает в систему управления для регистрации данных, обнаружения дефектов, управления движением, управления процессом и т.п.

Для получения информации об алгоритмах, предоставляемых инструментальными средствами технического зрения NI, обратитесь к руководству **Vision Concepts (Концепции технического зрения).**

Архитектура систем машинного зрения

Типичные системы машинного зрения состоят из промышленной камеры, подключенной к системе технического зрения реального времени, обычно через стандартную для камер шину IEEE 1394, гигабитный Ethernet или Camera Link. Система реального времени обрабатывает изображения и управляет вводом-выводом для связи с управляющей системой.

Несколько компаний объединили камеру с системой технического зрения, создав нечто под названием "интеллектуальная камера" (smart camera). Интеллектуальные камеры - промышленные камеры, которые содержат встроенную обработку изображений, и как правило, включающие базовые средства ввода-вывода.

National Instruments предлагает оба типа встраиваемых систем машинного зрения. NI Compact Vision System (рисунок 5.14) - встраиваемая система технического зрения реального времени, позволяющая напрямую подключать вплоть до трех камер IEEE 1394, а также использовать 29 линий ввода-вывода общего назначения для синхронизации и запуска. Эта система совместима с промышленной сетью Ethernet, предоставляя связь с аппаратными средствами NI CompactRIO.

NI Smart Cameras (рисунок 5.14) - промышленные датчики изображений, объединенные с программируемыми процессорами, предназначенные для создания промышленных функционально законченных приложений машинного зрения. Эти камеры с разрешением VGA (640x480 пиксел) или SXGA (1280x1024 пиксел) с возможностью подключения дополнительного цифрового сигнального процессора (DSP) с целью увеличения производительности при выполнении специфических алгоритмов. Эти камеры имеют сдвоенный порт гигабитного Ethernet, цифровые входы и выходы и встроенный контролер освещения.

Рисунок 5.14. NI Compact Vision System и NI Smart Camera

National Instruments предлагает также встраиваемые платы захвата изображений (frame grabber), которые обеспечивают подключение промышленных камер к слотам PCI, PCI Express, PXI и PXI Express. Эти платы широко используются для научных приложений и автоматизации тестирования, но вы можете их использовать и для создания систем технического зрения на базе ПК до приобретения промышленных систем технического зрения или интеллектуальных камер.

Все аппаратные средства для ввода изображений NI используют одни и те же драйвера и программное обеспечение NI Vision Acquisition. С помощью этого ПО вы можете разрабатывать и создавать прототипы ваших приложений на выбранной вами аппаратной платформе, а затем, с минимальными изменениями кода, развертывать на промышленной платформе, которая лучше удовлетворяет требованиям вашего приложения.

Освещение и оптика

В этом руководстве вопросы подсветки и оптики освещены неглубоко, но они критичны для успешной работы приложения. В документах "A Practical Guide to Machine Vision Lighting - Part I, II, III" (Практическое руководство по подсветке в системах машинного зрения: части 1, 2 и 3) рассматриваются базовые, а также некоторые более глубокие концепции освещения в системах машинного зрения.

Линзы, используемые в приложениях машинного зрения, изменяют поле обзора. Поле обзора – это инспектируемая область, изображение с которой передает камера. Важно убедиться, что поле обзора вашей системы включают объект, который вы хотите контролировать. Для расчета горизонтального и вертикального поля обзора (FOV) системы, используйте уравнение 1 и спецификации датчика изображений вашей камеры.

$$FOV = \frac{Pixel\ Pitch \times Active\ Pixels \times Working\ Distance}{Focal\ Length}$$

Уравнение 1. Расчет поля обзора

Где:

- *FOV (Поле обзора) - поле обзора в горизонтальном или вертикальном направлении*
- *Pixel Pitch (Шаг пикселя) - измеряет расстояние между центрами соседних пикселей в горизонтальном или вертикальном направлении*
- *Active Pixels (Активные пиксели) - количество пикселей в горизонтальном или вертикальном направлении*
- *Working Distance (Рабочее расстояние) – расстояние от переднего элемента (внешней оптики) линз до наблюдаемого объекта*
- *Focal Length (Фокусное расстояние) измеряет, насколько сильно линза фокусирует или рассеивает свет*.

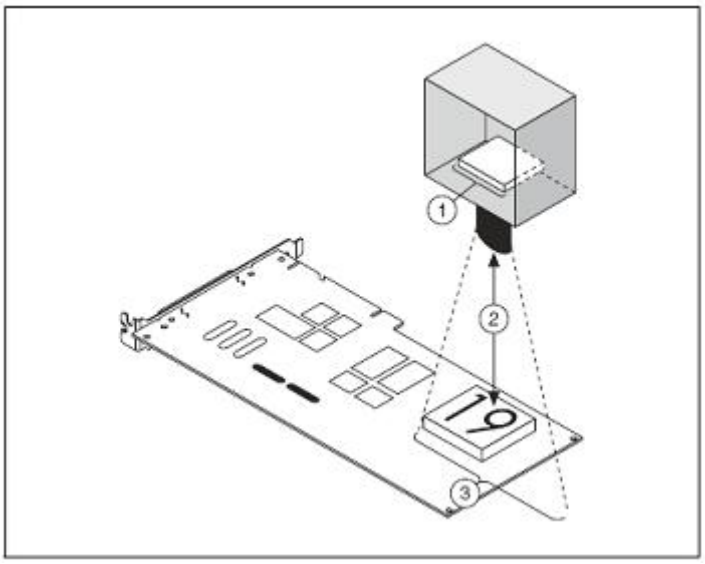


Рисунок 5.15. Выбор линзы определяет поле обзора

1 – Ширина изображения по горизонтали
2 – Рабочее расстояние
3 – Горизонтальное поле зрения

Например, если рабочее расстояние до изображения - 100 мм, фокусное расстояние линзы – 8 мм, тогда поле обзора в горизонтальном направлении NI Smart Camera с использованием датчика VGA в режиме полного сканирования составляет:

$$FOV_{horizontal} = \frac{0.0074 \text{ mm} \times 640 \times 100 \text{ mm}}{8 \text{ mm}} = 59.2 \text{ mm}$$

Аналогично, поле обзора в вертикальном направлении составляет:

$$FOV_{vertical} = \frac{0.0074 \text{ mm} \times 480 \times 100 \text{ mm}}{8 \text{ mm}} = 44.4 \text{ mm}$$

На основании этих результатов видно, что вам может понадобиться настроить различные параметры в уравнении FOV, пока не достигнете нужной комбинации компонентов, удовлетворяющих требованиям визуального контроля. Это может состоять в увеличения рабочего расстояния, выборе линзы с более коротким фокусным расстоянием или камеры с более высоким разрешением.

Варианты программного обеспечения

Как только вы выбрали аппаратную платформу для вашей системы машинного зрения, вам необходимо выбрать программную платформу. National Instruments предлагает две среды разработки приложений машинного зрения. И NI Compact Vision Systems, и NI Smart Cameras - целевые объекты LabVIEW Real-Time, так что вы можете разрабатывать приложение машинного зрения с использованием модулей LabVIEW Real-Time Module и NI Vision Development Module.

Модуль NI Vision Development - библиотека функций машинного зрения, назначение которых варьируется от простой фильтрации до сопоставления образов и оптического распознавания символов. Эта библиотека также включает VI NI Vision Assistant и экспресс-VI Vision Assistant. Vision Assistant (Помощник в создании систем машинного зрения) – инструмент быстрого макетирования приложений машинного зрения. С помощью этого инструмента вы можете использовать технику "щелчка и перетаскивания", и конфигурируемые меню для настройки большей части приложения. С экспресс-VI Vision Assistant Express VI вы можете использовать тот же инструмент разработки прототипов непосредственно в LabVIEW Real-Time.

Другая программная платформа - NI Vision Builder for Automated Inspection (AI) (Построитель приложений для контроля средствами машинного зрения). Vision Builder AI – конфигурируемая среда разработки систем машинного зрения, основанная на модели диаграмм состояний, в которых организация циклов и принятие решений очень просты. Vision Builder AI включает много инструментов высокого уровня, имеющихся в модуле Vision Development. Оба аппаратные целевые устройства работают и с Vision Builder AI, давая возможность выбора наиболее удобной среды разработки ПО, и аппаратных средств, лучше всего подходящих для вашего приложения.

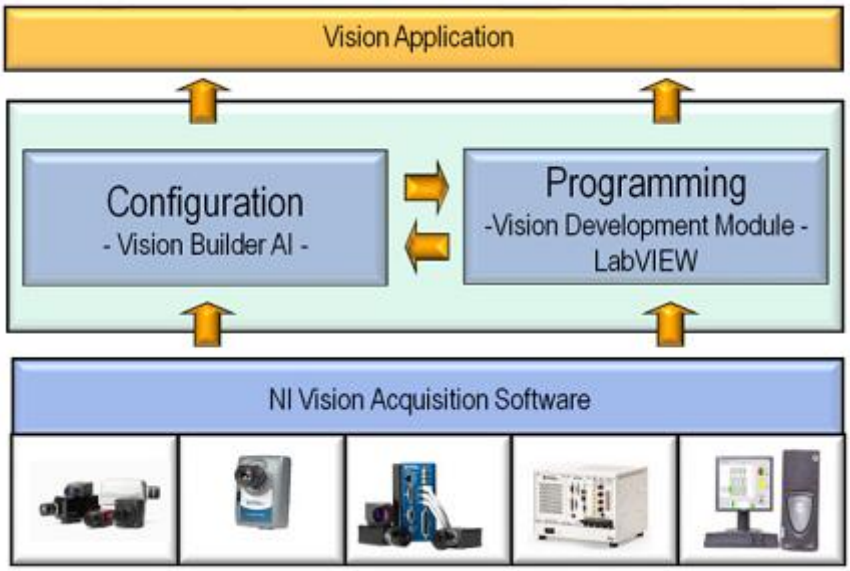


Рисунок 5.16. National Instruments предлагает конфигурируемое ПО и полнофункциональную программную среду для разработки приложений машинного зрения
Vision Application – приложение технического зрения, Configuration - настройка, Programming - программирование, NI Vision Acquisition Software – ПО сбора данных NI Vision

Интерфейс средств машинного зрения и системы управления

Большинство приложений на основе интеллектуальных камер или встроенных систем технического зрения обеспечивают оперативную обработку данных в реальном времени и содержат выходы, которые вы можете использовать как другой вход в систему управления. Система управления обычно запускает сбор и обработку изображения, посылая сигнал запуска системе технического зрения. Сигнал запуска может также прийти от аппаратных датчиков, например, бесконтактных или квадратурных энкодеров.

Изображение перерабатывается в набор используемых данных, например, положение ящика на конвейере или в значение и качество двумерного кода, напечатанного на детали автомобиля. Эти результаты передаются обратно в систему управления и/или посылаются по промышленной сети для регистрации. Вы может выбирать из нескольких методов передачи этих результатов, от простого цифрового ввода-вывода до переменных общего доступа или прямой связи по TCP/IP, как обсуждалось ранее.

Машинное зрение с использованием LabVIEW Real-Time

В следующем примере показано, как создать систему машинного зрения для NI Smart Camera, используя модули LabVIEW Real-Time и Vision Development.

Шаг 1. Добавление NI Smart Camera в проект LabVIEW

Вы можете добавить NI Smart Camera в тот же проект LabVIEW, что и систему CompactRIO. Если хотите создать макет без подключения интеллектуальной камеры, вы можете ее симулировать.

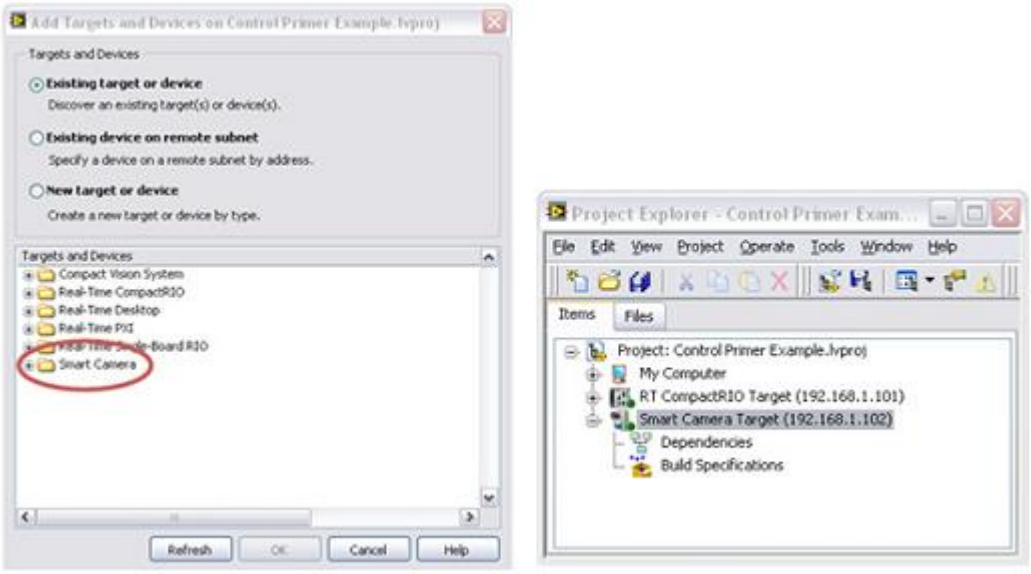


Рисунок 5.17. Вы можете добавить в систему NI Smart Camera в тот же проект LabVIEW, что и системы CompactRIO

Шаг 2. Использование LabVIEW для программирования NI Smart Camera

Создание приложения для интеллектуальной камеры очень напоминает создание приложения для контроллера реального времени CompactRIO. Основным отличием является использование драйверов NI Vision Acquisition для сбора изображений и выполнения алгоритмов обработки в Vision Development Module.Вы можете создать новый VI и запустить его на интеллектуальной камере так же, как вы это проделывали с системами CompactRIO.

Рисунок 5.18. Добавление VI для NI Smart Camera в проекте LabVIEW Real-Time

После развертывания этот VI сохраняется в памяти интеллектуальной камеры и запускается там на выполнение.

Для упрощения процесса ввода и обработки изображений можно использовать экспресс-VI, включенные в палитру Vision. Используйте экспресс-VI в этом примере для ввода изображений с интеллектуальной камеры (или системы технического зрения) и их обработки. Для доступа к экспресс-VI щелкните правой кнопкой по блок-диаграмме и выберите **Vision»Vision Express**.

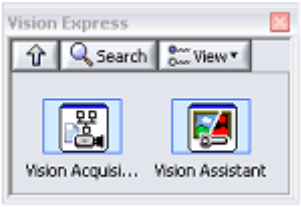


Рисунок 5.19 Палитра Vision Express

Первым шагом является настройка сбора данных с вашей камеры. Или, если камера недоступна или не установлена корректно, вы может настроить симулируемый ввод изображений, открывая изображения, хранимые на вашем жестком диске. Начните с размещения на блок-диаграмме экспресс-VI Vision Acquisition. Интерфейс этого VI, позволяет выполнить настройку с помощью меню, так что вы можете быстро получить первое изображение. Если у вас есть подключенные и опознанные аппаратные средства ввода изображений, они появятся как возможные варианты выбора. Если их нет, в первом меню вы можете выбрать вариант открытия изображения с диска.

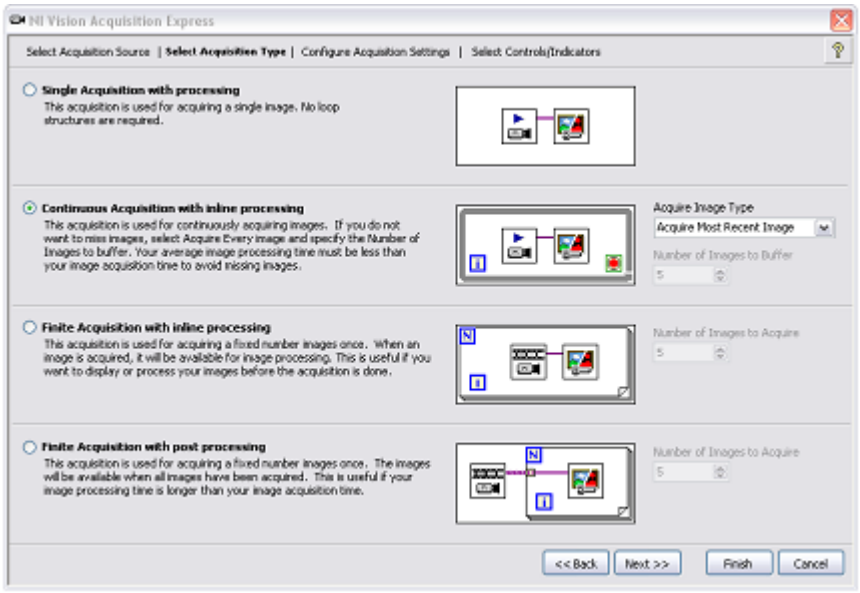


Рисунок 5.20. Экспресс-VI Vision Acquisition ведет вас в процессе создания приложения машинного зрения в LabVIEW

Далее выберите реализуемый вами тип сбора данных. Для этого примера выберите Continuous Acquisition with inline processing (Непрерывный сбор с оперативной обработкой данных). Это позволит вам обрабатывать изображения в цикле. Далее проверьте источник вводимых данных и убедитесь, что все выглядит правильно. Если все нормально, щелкните по кнопке finish внизу.

Как только этот экспресс-VI сгенерирует код LabVIEW, опять появится блок-диаграмма. Теперь поместите экспресс-VI Vision Assistant справа от Vision Acquisition экспресс-VI.

С помощью Vision Assistant вы можете быстро реализовать прототип программы обработки видеоданных. Вы можете развернуть эту программу в системе реального времени, хотя традиционные приборы LabVIEW, как правило, обеспечивают более высокую эффективность в системах реального времени.

Для обзора инструментов, доступных в этом помощнике, вызовите учебник **NI Vision Assistant Tutorial.**

Шаг 3. Связь с системой CompactRIO

После установки системы машинного зрения, которой собираетесь управлять при помощи экспресс-VI Vision Assistant, вам остается организовать обмен данными с CompactRIO. Вы можете

использовать публикуемые в сети переменные общего доступа для передачи данных межу двумя системами. Детали сетевого соединения между системами LabVIEW рассмотрены в предыдущих разделах этого документа. В этом примере рассматривается зажим аккумуляторной батареи – выполняется проверка, правильно ли просверлено отверстие, и каков текущий зазор зажима.

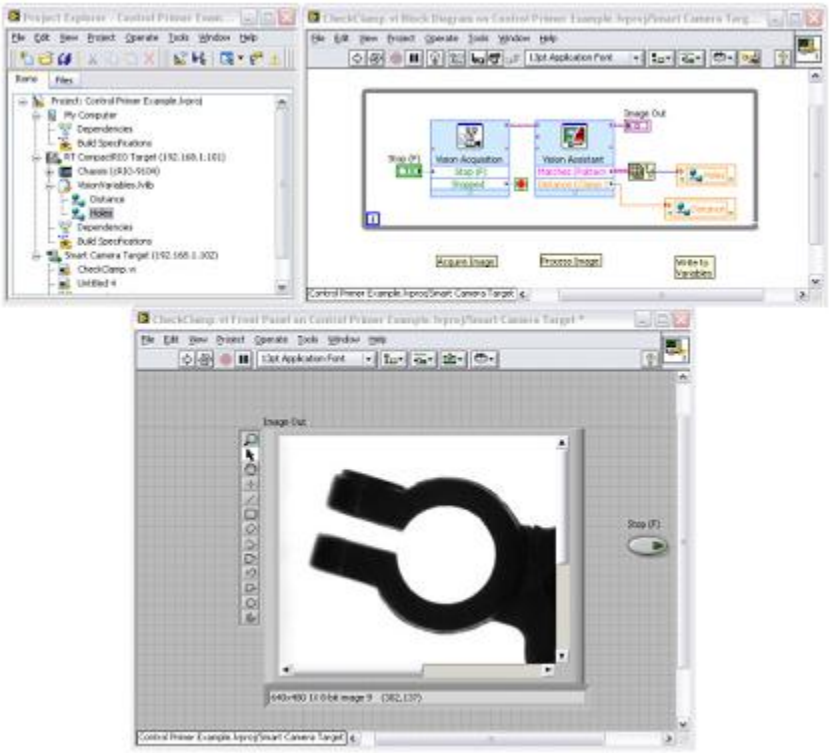


Рисунок 5.21. Полный контроль в LabVIEW.

Как видно из рисунка 5.21, результаты контроля передаются как текущие значения в CompactRIO через переменные общего доступа, обслуживаемые в CompactRIO. Данные можно также передавать в CompactRIO и как команды.

Машинное зрение с использованием Vision Builder AI

Как упоминалось выше, Vision Builder AI – конфигурируемая среда для создания машинного зрения. Вы реализуете все операции по вводу и обработке изображений, а также обработке данных с помощью меню настроек.

Шаг 1. Настройка NI Smart Camera с Vision Builder AI

Когда вы откроете среду проектирования, то увидите стартовую панель, где можете выбрать целевое устройство. Если интеллектуальная камера не подключена, вы можете ее симулировать. Для примера выберите этот вариант.

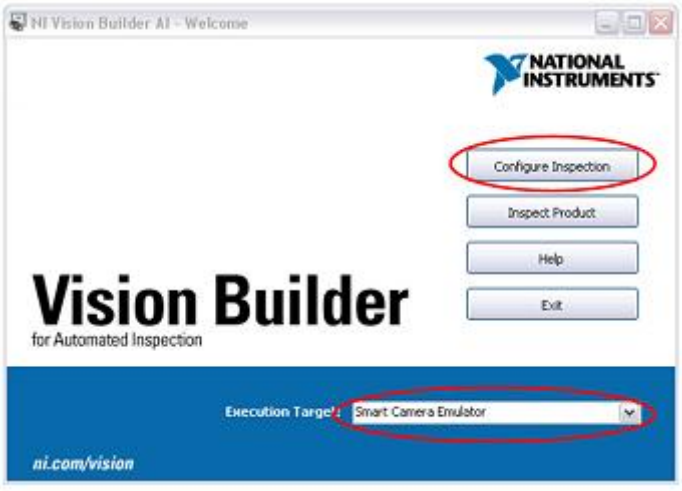


Рисунок 5.22. Выбор целевого устройства на стартовой панели Vision Builder AI

При помощи этого эмулятора можно задать варианты подсветки и запуска, настроить ввод-вывод для связи с аппаратными средствами CompactRIO и выполнить много других действий, требуемых для настройки интеллектуальной камеры, даже если ее еще нет в системе. Как только вы выбрали целевое устройство, щелкните по кнопке Configure Inspection (Настройка контроля). Откроется окно среды разработки с четырьмя главными окнами. Используйте самое большое окно Image Display (Показ изображения), чтобы увидеть полученное вами изображение и любые слои, которые вы на него наложили. Используйте правое верхнее окно State Diagram (Диаграмма состояний) для просмотра результатов контроля (текущее состояние подсвечено). Нижнее правое окно, Inspection Step Palette (Палитра стадий контроля), отображает все стадии контроля, выполняемые приложением. Наконец узкая панель внизу – окно Step Display (Отображение стадий), где вы видите все стадии в текущем состоянии.

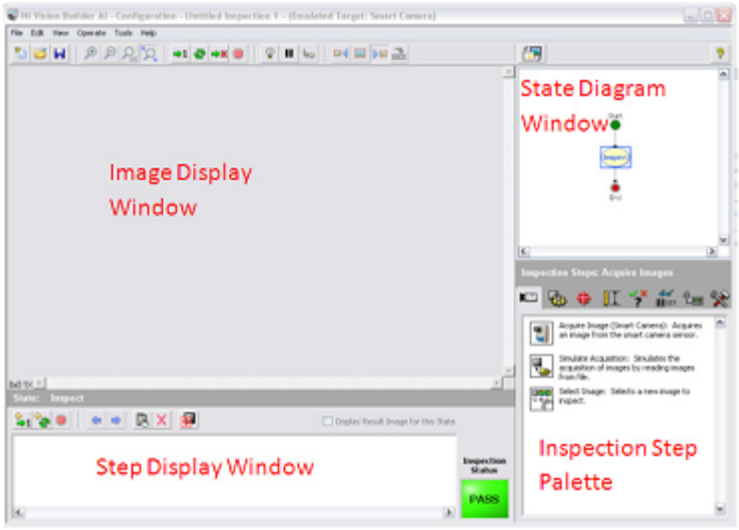


Рисунок 5.23. Среда разработки Vision Builder AI

Этот пример реализует тот же процесс контроля, что и пример LabVIEW, который исследует зажим аккумуляторной батареи и возвращает количество отверстий и размер зазора зажима в CompactRIO через две переменные общего доступа, обслуживаемые в аппаратуре CompactRIO.

Шаг 2. Конфигурирование контроля

Первая стадия контроля, как и в LabVIEW – ввод изображения. Реализуйте это процессом **Acquire Image** (Smart Camera). Выберите эту стадию и настройте эмуляцию, щелкнув по кнопке **Configure**

Simulation Settings (Настройка параметров симуляции). Например, выберите ввод изображение по следующему пути:

C:\Program Files\National Instruments\Vision Builder AI 3.6\DemoImg\Battery\BAT0000.PNG.

Эта библиотека изображений устанавливается с каждой копией Vision Builder AI (с различными номерами версий). Выбрав указанный выше файл, установите флажок **Cycle through folder images.**

Вы видите также, что в этом окне можете настраивать время экспозиции, масштаб и другие настройки камеры. Это неважно в эмуляторе, но в реальном мире также важно, как и выбор освещения и оптики.

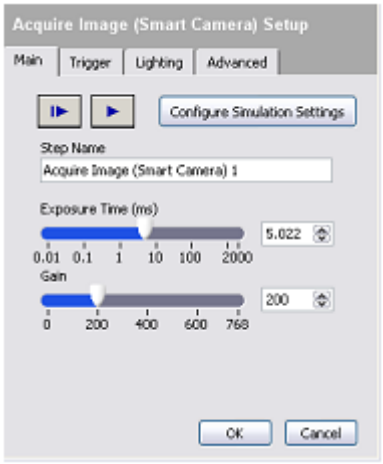


Рисунок 5.24. Конфигурирование стадии ввода изображения

Здесь же установите сравнение с эталоном, выделение границ, обнаружение объектов, чтение кода или любые другие требуемые вам алгоритмы.

Например, реализуйте сравнение с эталоном для настройки вращения объекта, обнаружение объектов для определения, на месте ли оба отверстия, и инструмент штангенциркуль, чтобы измерить расстояние между щечками зажима. При этом будут формироваться несколько результатов типа pass/fail (годен/брак) которые вы можете использовать для настройки общего статуса контроля, который можете показать пользователю на дополнительном слое.

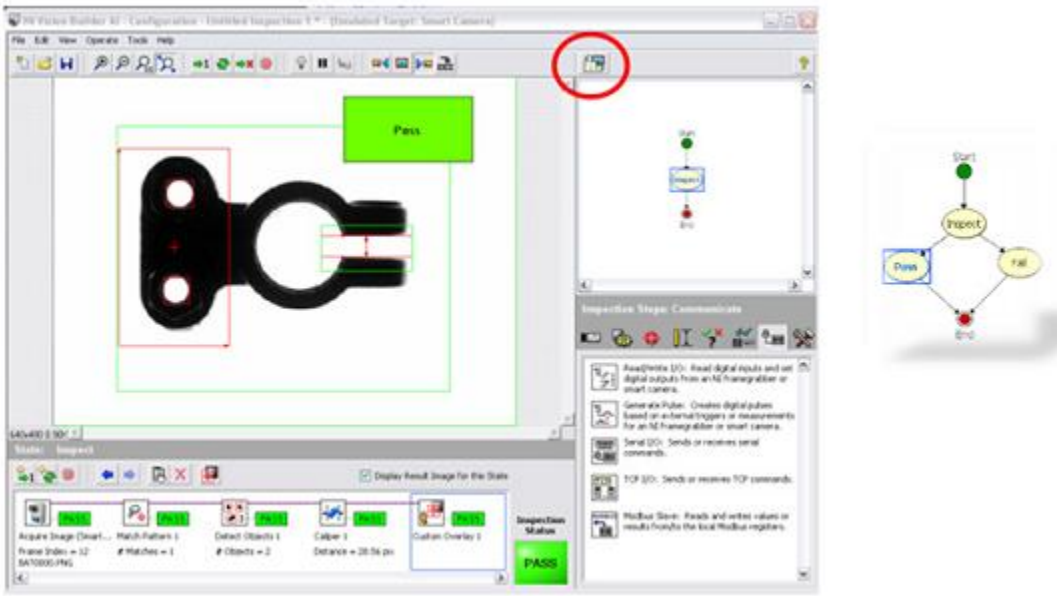


Рисунок 5.25. Полный контроль в Vision Builder AI

Теперь создайте два новых состояния, щелкнув на переключателе в главном окне (обведенном красным кружком). Создайте состояние "годен" и состояние "брак". В обоих состояниях передайте значения обратно в CompactRIO, но для состояния "брак" отбросьте деталь, используя цифровой ввод-вывод.

Шаг 3. Связь с системой CompactRIO

Vision Builder AI обеспечивает доступ к различным типам средств ввода-вывода, которые рассматривались ранее, включая публикуемые в сети переменные общего доступа, RS232, Modbus, Modbus/TCP и необработанные посылки TCP/IP. В этом примере используйте менеджер переменных для доступа к переменным общего доступа, обслуживаемым в CompactRIO. Открывается менеджер переменных из меню **Tools»Variable Manager**. Вы можете увидеть переменные CompactRIO, перейдя на закладку **Network Variables** (сетевые переменные). Здесь можно выбирать и добавлять переменные и привязывать их к переменным контроля.

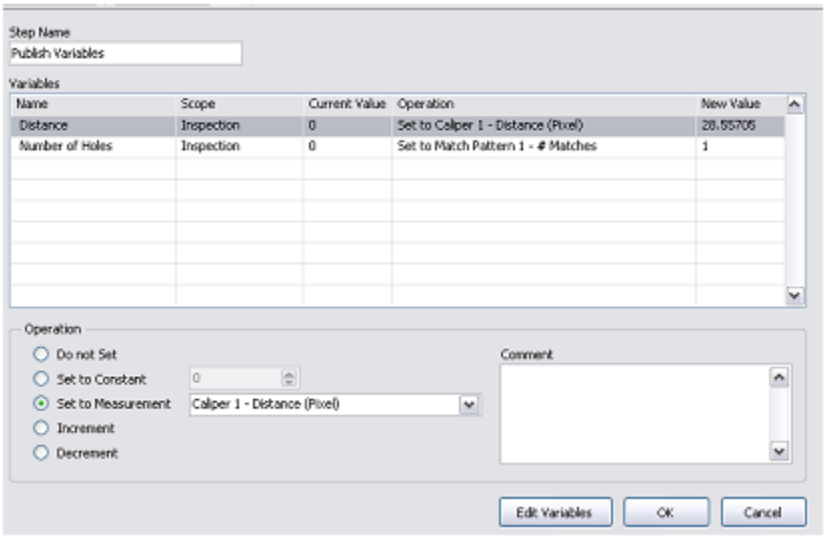


Рисунок 5.26. Настройка переменных для обмена данными в Vision Builder AI

Теперь эти результаты возвращаются в CompactRIO, а процесс контроля ожидает следующего запуска камеры.

Как вы можете видеть, оба метода (программирование и конфигурирование) предлагают пользователю возможности ввода изображений, их обработки и использования релевантной информации для передачи результатов обратно в систему управления CompactRIO, или непосредственно управлять вводом-выводом из системы технического зрения реального времени.

Управление движением

В этом документе рассматривается прецизионное управление перемещениями. Управление двигателем – это двухпозиционное регулирование или простое управление скоростью вращения таких объектов, как вентиляторы и насосы. Вы можете реализовать управление двигателем с помощью стандартных модулей цифрового вывода для пускателя электродвигателя или с помощью аналогового вывода для частотно-регулируемого электропривода, используя ранее рассмотренные архитектуры. Со специализированными сенсорами, актюаторами и быстрыми контурами регулирования вы можете выполнять прецизионное управление положением или скоростью, часто по нескольким координатам. В этом разделе рассматривается более сложная задача высокоточного управления движением на основе аппаратных средств CompactRIO.

Полномасштабная реализация управления движением представляет собой сложную систему с несколькими вложенными контурами регулирования, из которых часть работает на очень высоких скоростях, вместе с прецизионными механическими компонентами.

Надежная высокопроизводительная система управления движением состоит из следующих устройств:

1. Контроллер перемещений. Этот контроллер – управляющий элемент, запускающий программные алгоритмы и замкнутые контуры регулирования для генерации команд профиля движения на основании ограничений, определенных пользователем в прикладной программе, и обратной связи по вводу-выводу.
2. Модуль коммуникаций: этот модуль ввода-вывода осуществляет интерфейс с приводом двигателя и устройствами обратной связи. Он преобразует управляющие сигналы от контроллера перемещений в цифровые или аналоговые значения, которые привод может интерпретировать.
3. Привод/усилитель: Привод/усилитель состоит из силовой электроники, преобразующей аналоговые и цифровые управляющие сигналы от модуля коммуникаций в электрическую силу, необходимую для вращения двигателя. Во многих случаях он также имеет обрабатывающий элемент, которым замыкаются высокоскоростные контуры регулирования тока.
4. Двигатель: двигатель преобразует электрическую энергию, подаваемую от привода/усилителя в механическую энергию. Константа вращающего момента (kt) и эффективность привода определяют отношение между током электродвигателя и механическим моментом вращения.
5. Механическая передача: передача состоит из компонентов, соединенных с двигателем, которые управляют вращательным движением мотора. Она обычно включает механические устройства, такие, как коробки передачи и шкивы, и ходовой винт, преобразующий вращательное движение вала электродвигателя в линейное перемещение полезного груза с некоторым коэффициентом передачи. Распространенные примеры – конвейерные ленты, платформы и т.п.
6. Устройства обратной связи: это датчики, такие, как, например, энкодеры и концевые выключатели, обеспечивающие привод/усилитель и контроллер перемещений мгновенной информацией о положении и скорости.

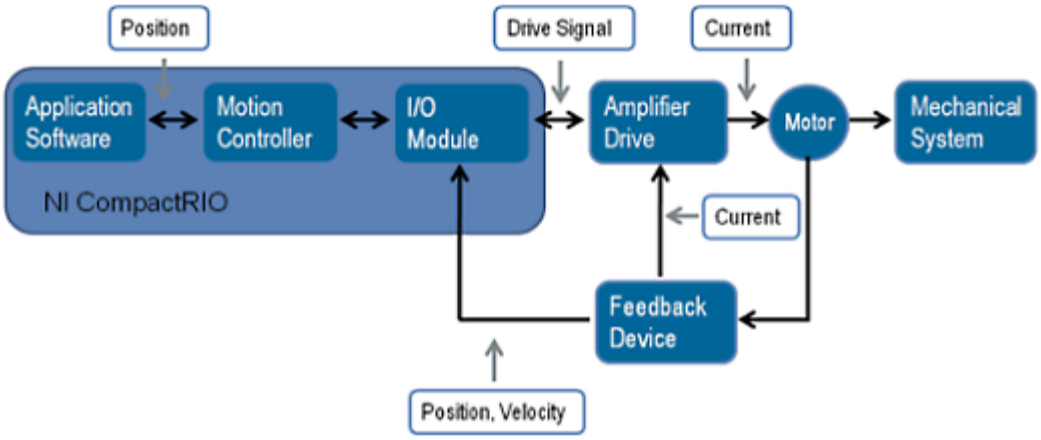


Рисунок 5.27. Упрощенная блок схема управления движением в CompactRIO

Контроллер перемещений

Контроллер перемещений – сердце системы управления движением. Он содержит ПО управления движением, которое определяет гибкость создаваемых сложных многокоординатных приложений управления движением. Контролер перемещений состоит из трех каскадно соединенных контуров регулирования.

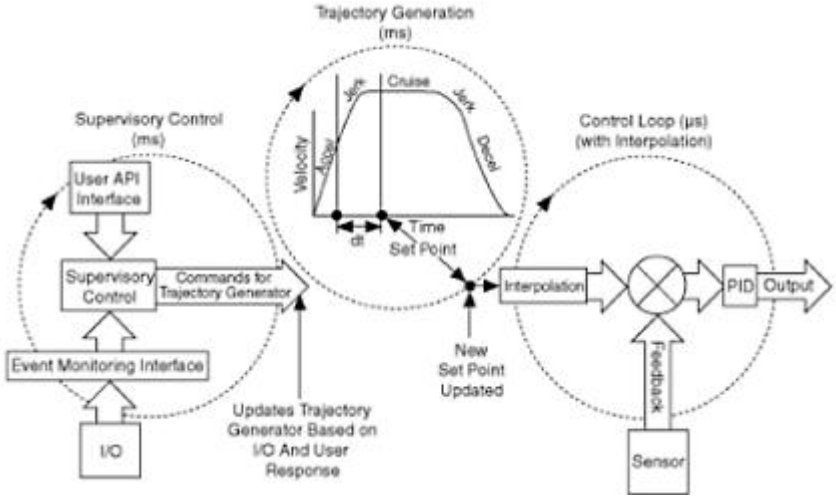


Рисунок 5.28. Функциональная архитектура контроллеров управления перемещениями NI

1. **Супервизорное управление:** главный управляющий контур выполняет последовательность команд и передает команды контурам формирования траектории. Это цикл выполняет:
   - Инициализацию системы, включая установку в исходную позицию
   - Обработку событий, включая запуск выходов на основании позиции или датчика обратной связи, и обновление профилей на основании определенных пользователем событий.
   - Обнаружение ошибок, включая останов движения при срабатывании концевого выключателя, безопасная реакция системы на аварийный останов или сбои двигателя и другие действия по сторожевому таймеру.
2. **Генератор траекторий**. Этот цикл получает команды из контура супервизорного управления и генерирует траекторию движения на основании заданного пользователем профиля. Генератор в детерминированном режиме предоставляет для контура управления новые координаты положений. На практике этот цикл обычно должен выполняться за время 5 мс или менее.
3. **Цикл управления:** Это очень быстрый контур регулирования, выполняющийся каждые 50 мкс. Он использует датчики положения и скорости и заданную координату из генератора траекторий для формирования команд приводу. Поскольку этот цикл имеет более высокую скорость, чем генератор траекторий, он также генерирует промежуточные координаты с учетом времени с помощью программы, называемой сплайн-интерполяцией. Для систем с шаговыми двигателями контур регулирования заменяется компонентом формирования шагов.

LabVIEW NI SoftMotion и модули NI 951x интерфейса с приводом

Вы можете создать все компоненты управления движением с нуля, используя модули LabVIEW Real-Time и LabVIEW FPGA, но National Instruments предоставляет программный модуль и модули ввода-вывода C-серии с предварительно собранными и протестированными компонентами.

Модуль LabVIEW NI SoftMotion предлагает высокоуровневый набор API для программирования управления движением и базовую архитектуру систем управления движением, описанную выше. Контур супервизорного управления и генератор траекторий работают на процессоре реального времени контроллера CompactRIO. NI также предлагает модули C-серии (NI 951x) интерфейса с приводом, работающие в контуре регулирования и обеспечивающие ввод-вывод для подключения приводов и датчиков перемещения. Все контуры и модули ввода-вывода конфигурируются в проекте LabVIEW.

Для приложений, где требуется большие возможности настройки, например, генератор произвольных траекторий или другой управляющий алгоритм, LabVIEW NI SoftMotion предлагает

программные средства для открытия кода и настройки этих компонентов в LabVIEW Real-Time и LabVIEW FPGA. Для приложений, требующих характеристик и функциональности, недоступных в модулях NI 951x, LabVIEW NI SoftMotion предлагает узлы связи с координатными осями, так что вы можете использовать другие модули ввода-вывода или коммуникационные интерфейсы для приводов сторонних фирм. Кроме того, LabVIEW NI SoftMotion позволяет разрабатывать виртуальные прототипы приложений управления движением и механизмами с использованием интерфейса системы автоматизированного проектирования трехмерных объектов SolidWorks Premium 3D CAD. С помощью NI SoftMotion for SolidWorks вы можете симулировать разработки, созданные в SolidWorks, используя реальные профили движения, спроектированные на основе функциональных блоков LabVIEW NI SoftMotion, прежде чем закупать физические прототипы.

Для получения дополнительной информации посетите сайт ni.com/virtualprototyping.

Начало работы по управлению движением в CompactRIO

Вы можете построить приложение управления движением на основе CompactRIO в четыре основных этапа:

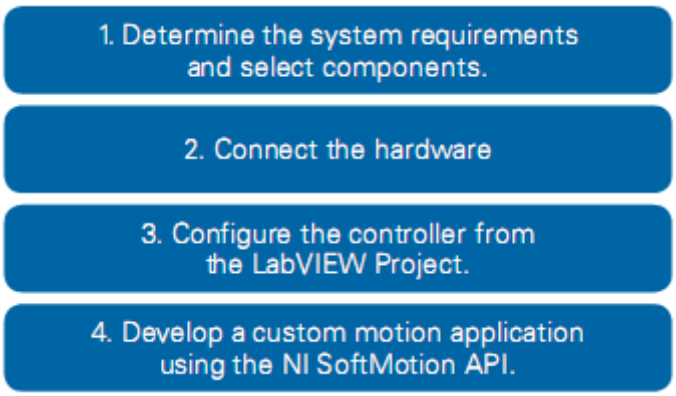


Рисунок 5.29. Четыре этапа построения системы управления движением

1. Определение требований к системе и выбор компонентов
2. Подключение аппаратных средств
3. Конфигурирование контроллера из проекта LabVIEW
4. Создание пользовательского приложения управления движением с помощью NI SoftMotion API

Определение требований к системе и выбор компонентов

Начните с выбора правильных механических компонентов и двигателей для вашей системы. Наиболее распространенные приложения включают перемещение объекта из одной точки в другую. Типичный способ превратить вращательное движение двигателя в полезное линейное перемещение – соединение двигателя с механическим преобразователем и использование механизма ходового винта для перемещения полезного груза.

Механические преобразователи (Stages) - устройства, которые обеспечивают линейное или вращательное движение, используемое для позиционирования и перемещения объектов. Они выпускаются различных типов и форм, так что вы может использовать их в различных приложениях. Чтобы выбрать правильные преобразователи для вашего приложения вы должны владеть базовой терминологией, применяемой для их описания. Ключевые элементы, на которые стоит обратить внимание при выборе механических преобразователей:

- Передаточное отношение: определяет отношение длины линейного перемещения к круговому перемещению двигателя.
- Точность: Насколько близко длина заданного перемещения совпадает с эталонной длиной.
- Разрешение: самое маленькое перемещение, которое способно выполнить система – может быть несколько нанометров.
- Длина перемещения: максимальное расстояние, на которое система способна переместиться в одном направлении.

- Повторяемость: повторяемое перемещение в заданную позицию при одинаковых условиях. Часто она задается как однонаправленная повторяемость, определяющая способность возвращаться в ту же точку из одного направления, и двунаправленная повторяемость, определяющая способность возвращаться из любого направления.
- Максимальная нагрузка: максимальный вес, который преобразователь физически способен перемещать, при заданной точности и повторяемости.

Выбор механического преобразователя

Вы можете выбирать из множества механических преобразователей в соответствии с требованиями вашего приложения. Вы можете сузить выбор до двух основных типов - линейного и вращательного. Линейные преобразователи движутся по прямой линии и часто располагаются пакетом для обеспечения движения в нескольких направлениях. Трехкоординатная система с компонентами x, y и z - широко распространенная архитектура для позиционирования объекта в трехмерном пространстве. Вращательная ступень преобразователя вращается вокруг оси (обычно относительно центра). Линейные ступени преобразователей используются для позиционирования объектов в пространстве, а вращательные - для ориентации объектов в пространства и настройки вращения, тангаж и рыскания объекта. Многие приложения, например, высокоточной юстировки, требуют точного и позиционирования, и ориентации. Разрешение вращательных ступеней часто измеряется в градусах или угловых минутах (1 градус равен 60 минутам). Существуют также особые типы механических преобразователей, например, гониометр, который похож на линейный механический преобразователь, движущийся по дуге, а не по прямой линии, или гексапод, который является механизмом параллельной кинематики, позволяющий передвигать объект в пространстве шести координат - x, y, z, вращение, тангаж и рыскание. С помощью гексапода вы можете определить виртуальную точку в пространстве, вокруг которой будет вращаться преобразователь. В этом преимущество гексаподов, а недостаток их в том, что гексаподы параллельны, и кинематика их работы значительно сложнее, чем простых пакетных преобразователей.

Люфт

Еще одним фактором при выборе механического преобразователя для прецизионной системы управления движением является люфт. Люфт – это зазор, образуемый, когда одна шестерня в системе меняет направление и перемещается на небольшое расстояние, прежде чем войдет в контакт с ведомой шестерней. Люфт может привести к значительный погрешностям, особенно в системе с большим количеством шестерней в кинематической цепи. При перемещении с нанометровой точностью, даже небольшой люфт может вызвать значительную погрешность системы. Эффективное проектирование механизма минимизирует люфт, но не могут устранить его полностью. Вы можете компенсировать эту проблему, реализовав дуальную петлю обратной связи в программном обеспечении. Модуль интерфейса сервопривода NI 9516 C-серии поддерживает обратную связь от сдвоенного энкодера и принимает отклики от двух различных источников для одной оси. Чтобы понять условия, при которых необходимо использовать эту возможность, рассмотрите механический преобразователь. Если вы контролируете положение преобразователя непосредственно (а не положению двигателя, управляющего преобразователем), то можете определить, достигло ли заданное вами перемещение цели. Однако, поскольку контроллер движения предоставляет входные сигналы двигателю, а не преобразователю, который является основным источником сигнала обратной связи, разница в ожидаемом выходе относительно входного сигнала может привести к нестабильности системы. Для задания настроек, необходимых для сохранения стабильности системы, можно снимать сигнал обратной связи непосредственно с энкодера двигателя, как источника вторичной петли обратной связи. Используя этот метод, вы можете контролировать реальное положение преобразователя и рассчитать погрешности кинематической цепи.

Выбор двигателя

Двигатель обеспечивает движение преобразователя. Некоторые высокоточные механические преобразователи и компоненты обладают встроенным двигателем для минимизации люфта и улучшения повторяемости, но большинство преобразователей и компонентов используют механические соединения для соединения со стандартным двигателем вращения. Для облегчения соединений Национальная ассоциация изготовителей электрооборудования (National Electrical Manufacturers Association - NEMA) стандартизировала размеры двигателя. Для маломощных электродвигателей (менее 1 л.с.) габаритные размеры имеют двухзначное обозначение, например, NEMA 17 или NEMA 23. Для этих двигателей габаритные размеры определяют высоту вала, диаметр вала и шаблон монтажных отверстий. В обозначении не определяется вращающий момент и скорости, так что существует набор различных комбинаций вращающего момента и скорости при одних и тех же габаритных размерах.

Мотор должен быть подобран в паре с механической системой для обеспечения требуемой производительности. Вы можете выбирать из следующих четырех основных технологий:

1. Шаговый двигатель: шаговый двигатель менее дорогой, чем серводвигатель того же размера и, как правило. проще в использовании. Эти устройства называются шаговыми двигателями, потому что они перемещаются дискретными шагами. Управление шаговым двигателем требует шагового привода, который получает сигналы о шагах и направлении от контроллера. Вы можете запускать шаговые двигатели, которые эффективны для недорогих приложений, в режиме без обратной связи (без отклика энкодера). Вообще шаговый двигатель имеет высокий вращающий момент на низких скоростях и хорошо его удерживает, но низкий – на высоких скоростях, а также более низкую максимальную скорость. Движение на низких скоростях также могут быть прерывистыми, но большинство шаговых двигателей обладают способностью к мелким шагам для устранения этой проблемы.
2. Щеточный серводвигатель: это простой двигатель, где электрические контакты передают энергию ротору через механический поворотный выключатель, называемый коммутатором. Эти двигатели обеспечивают двухпроводное подключение и управляются изменением тока через обмотку двигателя, часто с помощью широтно-импульсной модуляции. Привод двигателя преобразует управляющий сигнал, обычно аналоговый сигнал ±10 В, в ток, подаваемый на двигатель, и может требовать настройки. Такие двигатели довольно просты в управлении и обеспечивают хороший вращающий момент во всем диапазоне скоростей. Однако они требуют периодического обслуживания щеток и, по сравнению с бесщеточными серводвигателями, обладают ограниченным диапазоном скоростей и менее эффективны ввиду механических ограничений срока службы щеток.
3. Бесщеточный серводвигатель: бесщеточные серводвигатели используют ротор с постоянными магнитами, трехфазные управляющие обмотки и датчики Холла для определения положения ротора. Специализированный привод преобразует аналоговый сигнал ±10 В от контроллера в трехфазные напряжения для двигателя. Привод обладает интеллектом для электронной коммутации и требует настройки. Этим двигателям свойственны более высокий к.п.д., высокие вращательный момент и скорость, меньший объем работ по обслуживания. Однако они более сложны для монтажа и настройки, а двигатель и привод более дорогие.
4. Пьезодвигатель: пьезодвигатели используют пьезоэлектрический материал для генерации ультразвуковых вибраций или шагов и создания линейного или вращательного перемещений при помощи движений, похожих на движение гусеницы. Эти двигатели, которые могут формировать экстремально точное движение, часто используются в приложениях нанопозиционирования, например, для юстировка лазеров. Для большей точности пьезодвигатели часто интегрируются в механический преобразователь или актюатор, но вы также можете использовать вращательный пьезодвигатель.

Тип привода	Скорость	Макси-мальная нагрузка	Длина переме-щения	Повторя-емость	Относи-тельная сложность	Относи-тельная цена
Шаговый	Средняя/ низкая	Средняя/ низкая	Высокая	Средняя/ низкая	Низкая	Низкая
Щеточный серводвигатель	Высокая	Высокая	Высокая	Средняя	Средняя	Низкая
Бесщеточный серводвигатель	Очень высокая	Высокая	Высокая	Высокая	Высокая/ средняя	Высокая
Пьезо	Средняя	Низкая	Низкая	Очень высокая	Высокая	Высокая

Рисунок 5.30. Сравнение вариантов технологий движения

Поскольку двигатель и привод тесно связаны, вы должны использовать их от одного производителя. Хотя это не обязательное требование, оно упростит выбор и наладку.

Подключение аппаратных средств

Как только вы выбрали требуемые механические компоненты, двигатель и привод для вашего приложения, вам нужно подключить систему CompactRIO к аппаратным средствам. При помощи Axis Interface Node (узла интерфейса осей) и LabVIEW FPGA вы можете использовать любой модуль C-серии для подключения к системе CompactRIO; однако для простоты National Instruments рекомендует использование модулей интерфейса привода C-серии NI 951x.

Модули интерфейса привода NI 951x

Модули NI 951x формируют сигналы интерфейса шагового или сервопривода для одной координаты и могут подключаться к сотням типов шаговых и сервоприводов.

Помимо подводки питания к соответствующим линиям ввода-вывода для управления движением, модули снабжены процессором для сплайн-интерполяции с запатентованным NI алгоритмом генерации шагов или контуром управления сервоприводом.

- NI 9512 - модуль интерфейса одноосного шагового или командно-позиционного привода с обратной связью через инкрементальный энкодер
- NI 9514 - модуль интерфейса одноосного сервопривода с обратной связью через инкрементальный энкодер
- NI 9516 - одноосный сервопривод с обратной связью через сдвоенный энкодер

При запуске в режиме сканирования эти модули должны вставляться в любой из первых четырех слотов шасси CompactRIO. С LabVIEW FPGA вы можете использовать любой слот.

Модули NI 951x были разработаны для обеспечения гибкости и простоты подключения двигателя к линиям ввода-вывода одного модуля управления движением. Подключение еще большее упрощается, благодаря возможности программного конфигурирования линий цифрового ввода-вывода в качестве источников или приемников тока нагрузки. В модулях реализованы:

- Аналоговые или цифровые управляющие сигналы для привода
- Сигналы включения привода; программно настраиваемые как источники или приемники тока нагрузки (24 В)
- Сигналы цифрового ввода для исходного положения, ограничений, и общего цифрового ввода-вывода; все программно настраиваемые как источники или приемники тока нагрузки (24 В)
- Входы энкодера и питания (5В); настраиваемые как несимметричные или дифференциальные

- Цифровой ввод/цифровой вывод для высокоскоростных функций фиксации/сравнения положения (5 В)
- Светодиодные индикаторы для быстрой отладки состояний энкодера, пределов и сбоев осей.

Для дальнейшего упрощения подключения National Instruments предлагает несколько вариантов подключения модулей интерфейса привода NI 951x к внешним приводам шаговых двигателей или сервоусилителям:

- NI 9512-to-P7000 Stepper Drives Connectivity Bundle – комплект для подключения модуля NI 9512 к приводам шаговых двигателей P70530 или P70360
- NI 951x Cable and Terminal Block Bundle – комплект из кабеля и коннекторного блока для подключения модуля NI 951x с помощью 37 пружинными или винтовыми клеммами
- D-Sub and MDR solder cup connectors – разъемы для упрощения изготовления пользовательского кабеля
- D-Sub to pigtails cable и MDR to pigtails cable – кабели с разъемом на одном конце, упрощают монтаж пользовательского кабеля

Рисунок 5.31. Выбор из нескольких вариантов для простого подключения модулей NI 951x непосредственно к приводу

Конфигурирование контроллера из проекта LabVIEW

Для использования LabVIEW NI SoftMotion в LabVIEW вы должны сначала создать оси, координаты и таблицы в проекте LabVIEW. Когда вы создадите эти элементы, физические ресурсами устройства движения нужно привязать к логическим каналам и определить фоновые контуры регулирования для выполнения на контроллере. Вы используете логические каналы, созданные при разработке приложений управления движением с использованием функций API NI SoftMotion LabVIEW.

- Ось – логический канал, ассоциируемый с отдельным двигателем. Каждый контролируемый вами двигатель должен быть представлен осью.
- Координата - группировка одной или более осей. Планируя несколько осей для координаты, вы создаете многокоординатное движение. Например, если у вас есть механический преобразователь XY, и вы хотите создать овальную траекторию движения, сложно управлять двумя двигателями по отдельности. Но если вы сгруппируете их, как ось, генератор траекторий LabVIEW NI SoftMotion автоматически создаст командные точки для каждой оси, чтобы результирующее движение было по овалу.

- Таблица используется для задания более сложных профилей движения, например, движения по контуру или по кривой. Вы можете импортировать профиль движения из внешнего текстового файла с табулятором в качестве разделителя.

Добавление осей, координат и таблиц в проект

Вы можете добавить оси, координаты и таблицы, щелкнув правой кнопкой мыши по контроллеру CompactRIO в проекте LabVIEW и выбрав New (Новый).

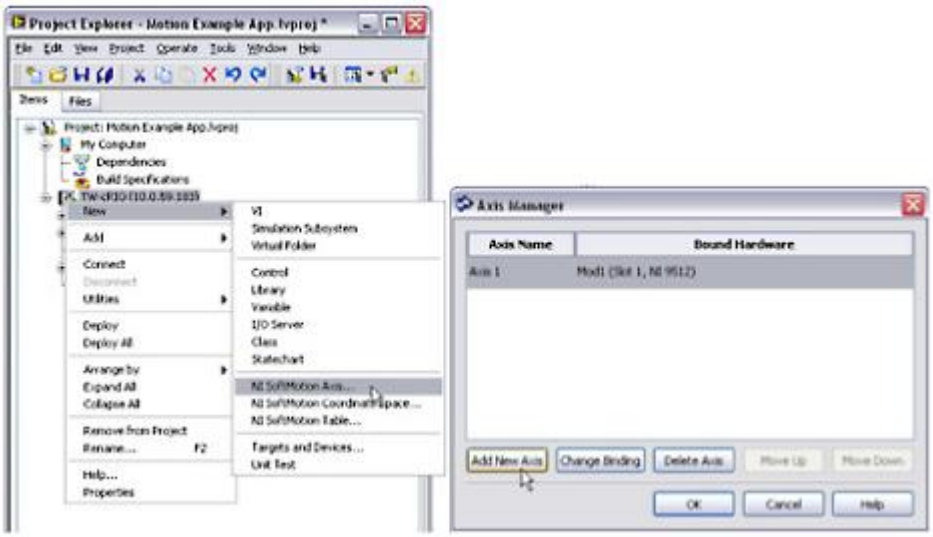


Рисунок 5.32. Добавление оси в систему CompactRIO

Ось состоит из генератора траекторий, контура ПИД-регулирования или выхода сигналов для шагового двигателя, и супервизорного управления. Вы можете связать ось LabVIEW NI SoftMotion с симулируемым или реальным оборудованием. Оси серводвигателя требует обратной связи с энкодером. Оси шаговые двигателей не требуют обратной связи.

Как только вы создали оси, вы можете генерировать координаты и добавлять оси в координаты. При использовании координатных ресурсов в LabVIEW вы получаете положение цели и другую информацию о координатах в одномерном массиве. Информация об осях расположена в том же порядке, в каком оси добавлялись в диалоговое окно.

Конфигурирование осей

После добавления оси в проект, нужно их сконфигурировать, щелкнув правой кнопкой мыши и выбрав properties (Свойства).

Для конфигурирования привода шагового двигателя, подключенного к приводу шагового двигателя серии NI P7000, выполните следующие действия:

1. Щелкните правой кнопкой мыши по оси в окне обозревателя проекта и выберите Properties (Свойства) из контекстного меню для открытия диалогового окна Axis Configuration (Конфигурирование осей).

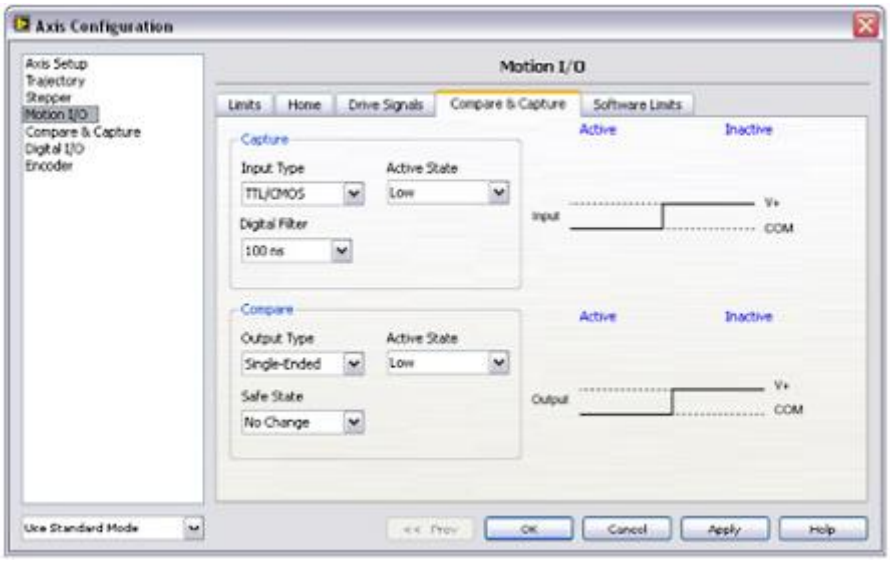


Рисунок 5.33. Вы можете использовать окно Axis Configuration (Конфигурирование осей) для настройки всех параметров ввода-вывода системы управления движением

2. На странице Axis Setup (Установка осей) подтвердите, что Loop Mode (Режим контура) установлен на Open-Loop (Разомкнутый). Оси в этом режиме формируют выходные сигналы для выполнения шага, но не требуют обратной связи от двигателя для контроля положения.
3. На странице Axis Setup убедитесь также, что установлены флажки Axis Enabled (Оси разрешены) и Enable Drive on Transition to Active Mode (Разрешить привод при переходе в активный режим). Таким образом оси активируются автоматически при переключении механизма сканирования NI в активный режим.
4. Если у модулей нет входов для физических ограничений и ввода исходного положения, вы должны отключить эти входные сигналы для правильной работы системы. Для этого перейдите на страницу Motion I/O (ввод-вывод управления движением) и снимите флажки в разделах Forward Limit (передний предел), Reverse Limit (обратный предел) и Home (исходное положение).
5. Настройте дополнительные параметры ввода-вывода в соответствии с требованиями приложения.
6. Щелкните по кнопке OK для закрытия диалогового окна Axis Configuration.
7. Щелкните правой кнопкой мыши по элементу контроллера в окне обозревателя проекта и выберите из контекстного меню команду Deploy All (Развернуть все).

ВНИМАНИЕ: Прежде чем развертывать проект, убедитесь, что все технические средства смонтированы, а питание включено. Развертывание переключает механизма сканирования NI в активный режим и включает оси и привод, так что движение может начаться немедленно.

Тестирование системы управления движением

Чтобы убедиться, что ваша система сконфигурирована и подключена верно, вы можете использовать интерактивные тестовые панели (Interactive Test Panel) для тестирования и отладки вашей системы. При помощи интерактивных тестовых панелей вы можете выполнять и контролировать простое прямолинейное движение, проверять состояние ввода-вывода, изменять ограничения движения, получать информацию об ошибках и сбоях в системе, просматривать графики перемещения или скорости. Если в системе подключено устройство обратной связи, вы можете также получать информацию о состоянии обратной связи и об ошибках позиционирования.

Для запуска интерактивного окна щелкните правой кнопкой мыши по оси в окне обозревателя проекта и выберите пункт Interactive Test Panel из контекстного меню. На соответствующих закладках установите желаемую позицию, режим перемещения и ограничения.

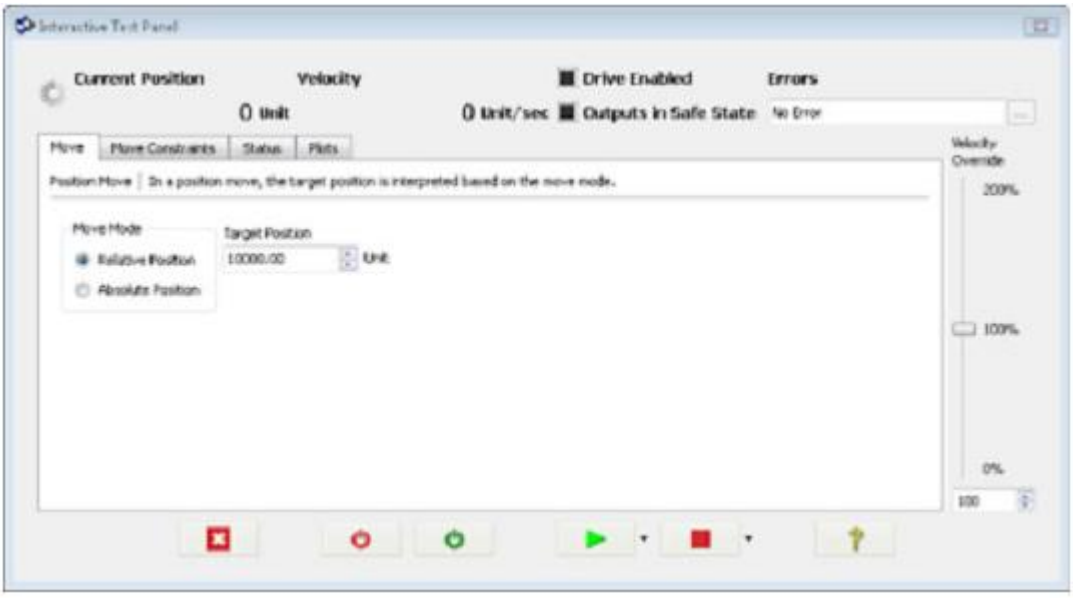


Рисунок 5.34. С помощью интерактивных тестовых панелей вы можете проверить правильность конфигурирования системы управления движением до разработки кода

Щелкните по кнопке Start (Старт) внизу диалогового окна для начала движения с заданными параметрами. Используйте закладки Status (Состояние) and Plots (Графики) для наблюдения за движением.

Разработка пользовательского приложения управления движением с использованием LabVIEW NI SoftMotion API

Модуль LabVIEW NI SoftMotion предоставляет API в виде функциональных блоков для создания детерминированных приложений управления движением.

Эти функциональные блоки, разработка которых инспирирована функциональными блоками управления движением PLCopen, используют ту же терминологию и парадигму выполнения, что и функциональные блоки IEC 61131-3. Несмотря на то. что программирование с использованием функциональных блоков похоже на программирование на основе потока данных в LabVIEW, существуют некоторые различия при выполнении, которые вы должны понимать, прежде чем разрабатывать приложения с использованием функциональными блоками. Сами функциональные блоки не выполняют алгоритмов управления движением. Они представляют собой API (программный интерфейс приложения), который посылает команды менеджеру движения, которые выполняется как служебный драйвер контроллера CompactRIO. Менеджер движения выполняется со скоростью сканирования, но вы можете запустить API функциональных блоков на любой скорости и даже вызвать блоки в недетерминированном коде.

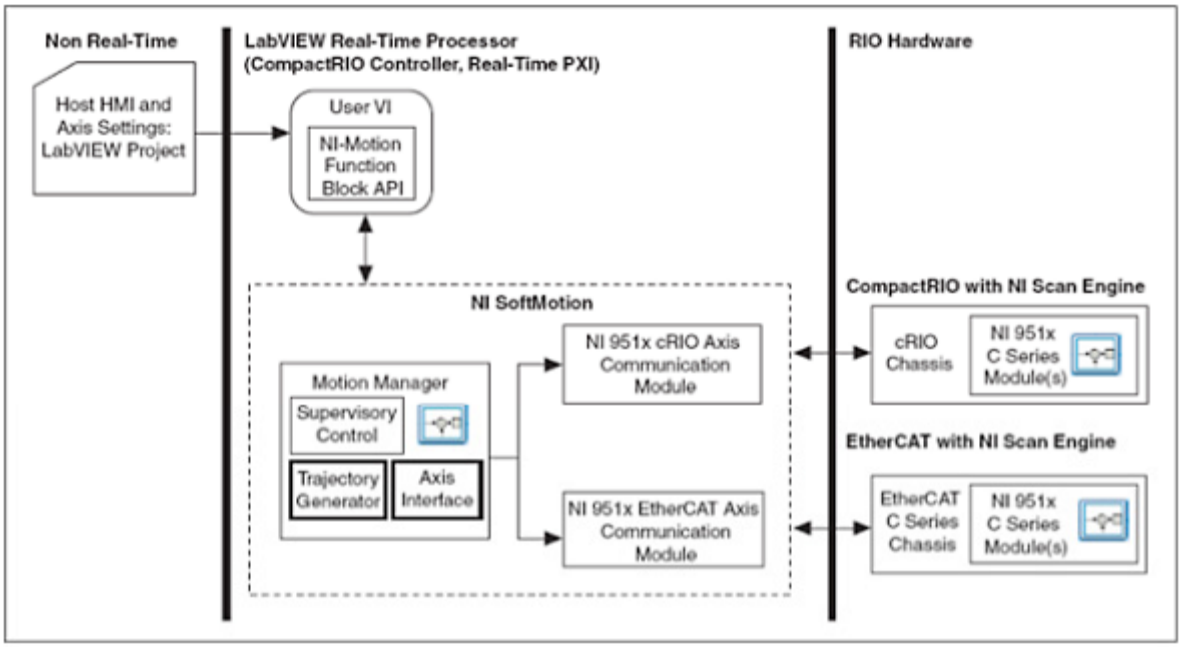


Рисунок 5.35. Блок-схема компонентов LabVIEW NI SoftMotion в системе CompactRIO
Non Real-Time – не реальное время, LabVIEW Real-Time Processor – процессор реального времени LabVIEW (контроллер CompactRIO, PXI реального времени), RIO Hardware – аппаратные средства RIO

Host HMI and Axis Settings: LabVIEW Project – интерфейс оператора и настройки осей, User VI – пользовательский VI, NI-Motion Function Block API – API функциональных блоков NI-Motion, Motion Manager – менеджер движений, Supervisory control –супервизорное управление , Trajectory Generator – генератор траекторий, Axis Interface – интерфейс осей, NI 951x cRIO Axis Communication Module – модуль NI 951x cRIO интерфейса осей, NI 951x EtherCAT Axis Communication Module – модуль NI 951x EtherCAT интерфейса осей, CompactRIO with NI Scan Engine – CompactRIO с механизмом сканирования, EtherCAT with NI Scan Engine – EtherCAT с механизмом сканирования, cRIO Chassis – шасси cRIO, EtherCAT C-series Chassis – шасси EtherCAT для модулей C-серии

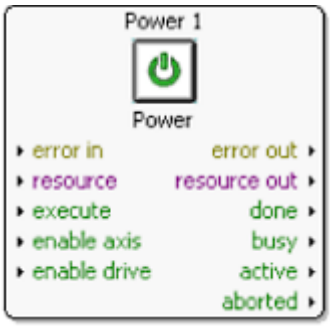


Рисунок 5.36. Функциональный блок LabVIEW NI SoftMotion

Функциональные блоки для работы с движением являются API функциями менеджера движений, выполняющиеся в системе CompactRIO. Они выполняют две операции:

1. **Посылают команду менеджеру движений.** Функциональный блок посылает команду менеджеру движения, когда на вход "execute" поступает переход от низкого уровня сигнала к высокому (положительный фронт). Значение по умолчанию – "false", так что если к блоку подключена константа "true", на первой итерации считается. как положительный фронт.
2. **Опрос менеджера движений**. Каждую итерацию функциональный блок опрашивает менеджер, чтобы узнать, была ли команда выполнена успешно. Результаты опроса возвращаются через выходы "done" (выполнено), "busy" (выполняется), "active" (активно), "aborted" (прервано) и "error out" (выход ошибки).

Эти выходы ведут себя согласно следующей таблице:

Output exclusivity (Взаимоисключаемость выходов)	"Busy", "done", "error out" и "aborted" - взаимно исключающие друг друга выводы. Только один из этих выводов одного функционального блока в определенный момент может находиться в состоянии "true". Если вход "execute" - "true", то один из этих выходов должен быть истинен.
Output status (Статус выходов)	Выходы "done" and "aborted" сбрасываются по отрицательному фронту execute (состояние фиксируется, пока на входе "execute" высокий уровень). Однако отрицательный фронт сигнала execute не останавливает и не влияет на работу менеджера движений, если он уже получил команду.
Behavior of done output (поведение выхода "выполнено")	Выход "done" принимает истинное значение, если команда успешно выполнена. Значение по умолчанию – "false". Как только движение завершится, он сбрасывается и снова принимает значение "false", если execute - "false".
Behavior of aborted output (поведение выхода "прервано")	"Aborted" устанавливается, когда операция прерывается другой командой. Поведение этого выхода при сбросе такое же, как у выхода "done".
Behavior of busy output (Поведение выхода "занят")	У каждого функционального блока есть выход "busy", указывающий, что операция этого функционального блока незавершена. "Busy" устанавливается по положительному фронту "execute" и сбрасывается при установке любого из выходов "done", "error out" и "aborted". Рекомендуется, чтобы приложение, контролирующее выполнение этого функционального блока не закрывалось, по крайней мере, пока выход "busy" истинен, потому что программа может остаться в неопределенном состоянии.
Output active (Выход "активно")	Выход "active" принимает значение "true" в момент, когда функциональный блок выполняет управление заданным ресурсом, все свойства зафиксированы и функциональный блок выполнился.

Использование функциональных блоков LabVIEW NI SoftMotion

Следующие советы могут вам пригодиться при программировании с функциональными блоками LabVIEW NI SoftMotion в LabVIEW:

Функциональные блоки:

- Не блокируются и всегда выполняются за определенный период времени
- Начинают передачу команд менеджеру движений по положительному фронту сигнала на входе "execute" (выполнить).
- Обеспечивают обратную связь через выход "done" (выполнено), если менеджер движений успешно завершил задачу.

- Копируемы (имеют уникальное пространство памяти) и реентерабельны, но каждая копия может быть вызвана только из одного места программы.
- Должны выполняться из VI, входящего в проект LabVIEW.
- Двойным щелчком мыши позволяют вызвать диалог, который можно использовать для настройки значений по умолчанию, автоматической привязки данных к переменным и конфигурирования в качестве источников данных терминала, переменной или значения по умолчанию.
- Имеют методы, доступные из контекстного меню. Например, функциональный блок "Stop Move" (остановить движение) может уменьшить скорость привода, немедленно остановить или отключить привод.
- Должны запускаться только в цикле. В зависимости от требований приложения используйте или цикл while с функцией задержки Wait Until Next ms Multiple или синхронизируемый цикл.

Вы должны использовать входы и выходы функциональных блоков (execute, done и т.п.), а не стандартные методы программирования LabVIEW, для определения порядка выполнения функциональных блоков. Например, не помещайте функциональные блоки в структуру Case, если только структура не управляется выходами статуса. Рассмотрим приведенную ниже блок-диаграмму:

Рисунок 5.37. Пример неправильного программирования движения с использованием функциональных блоков

Если вы не знакомы с функциональными блоками, вы будете ожидать, что код переместит ось 1 в положение 1000, а затем обратно в положение 0. Однако функциональные блоки на самом деле не совершают движения – они просто посылают команды менеджеру движений, когда появляется положительный фронт на входе "execute". Приведенный выше код включит привод и оси, но движения не произойдет.

- Функция Power (Питание) обнаружит положительный фронт на входе "execute" (по умолчанию – "false", так что если вы подключите "true", это произойдет при первой итерации).
- При получении положительного фронта, она посылает команду менеджеру движений включения оси и привода.
- Эта неблокируемая функция не ожидает, пока менеджер выполнит команду, прежде чем продолжить код LabVIEW, и на выход "done" - "false" при первой итерации.
- Все остальные блоки в цепи кода не обнаруживают положительный фронт на входах "execute" и не посылают никаких команд менеджеру.
- Движения нет.

Рисунок 5.38. Пример работающей программы с использованием функциональных блоков

В этом примере привод включается, перемещается в положение 1000 и возвращается в положение 0, но не выполняет этих движений повторно. причина этого в том, что каждый функциональный блок получает положительный фронт на входе "execute" только один раз. Чтобы эта ось непрерывно повторяла движение между положениями 1000 и 0, вам нужно написать код, подобный показанному на рисунке 5.39.

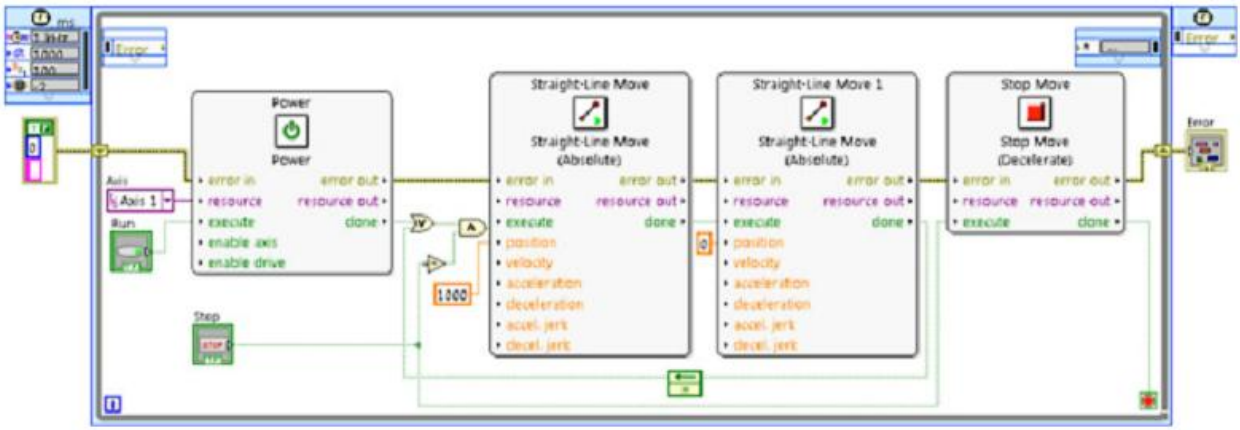


Рисунок 5.39. Пример правильного программирования повторяющихся движений с помощью функциональных блоков

Этот код вызывает повторяющиеся движения между положениями 1000 и 0.

- В начале цикла функциональный блок Power (Питание) посылает команду менеджеру движений.
- Далее опрашивается менеджер для проверки статуса команды. При получении подтверждения, что команда выполнена, он записывает "true" в бит "done" и выходит из первого цикла.
- Первый функциональный блок Straight-Line Move (Движение по прямой) в синхронизируемом цикле при сигнале "true" на терминале "execute" выдает команду менеджеру переместиться в положение 1000.

- На выходе "done" удерживается уровень "false", пока движение не завершится, и второй блок Straight-Line Move ничего не делает, потому что на его входе "execute" сигнал "false".
- На каждой итерации цикла первый блок Straight-Line Move опрашивает менеджер, завершено ли движение. Когда первое движение завершено, на выходе "done" появляется уровень "true".
- Второй блок Straight-Line Move обнаруживает переход от низкого уровня к высокому на входе "execute" и посылает команду менеджеру.
- Когда этот блок получает подтверждение, что команда завершена, он переводит свой терминал "done" в состояние "true".
- Благодаря сдвиговому регистру на на терминал "execute" первого функционального блока поступает переход от низкого уровня к высокому, и перемещение повторяется.
- При выполнении команды "стоп", код очистки в этой команде останавливает движение. Поскольку менеджер движений – отдельный процесс, без команды стоп, посылаемой менеджеру, движение будет продолжаться до завершения, даже если код LabVIEW больше не выполняется.

В следующей таблице приведено краткое описание функциональных блоков:

Объект палитры	Символ палитры	Описание
Line (Линия)		Выполняет движение по прямой, используя осевые или координатные ресурсы. Движение по прямой соединяет две точки с использованием одной или более осей. Характеристики движения зависят от режима движения по прямой (Straight-Line Move Mode).
Arc (Дуга)		Выполняет движение по круговой, сферической или спиральной дуге. Движение по дуге осуществляется по заданному вами радиусу. Тип дуги зависит от режима движения (Arc Move Mode).
Contour (Контур)		Выполняет движение по контуру, используя осевые или координатные ресурсы. Движение по контуру задается набором точек, которые ПО использует для экстраполяции гладкой кривой. Эти точки хранятся в таблице. Каждая точка движения интерпретируется как абсолютное положение, а начальная точка движения – как временное положение "zero". Тип движения по контуру зависит от режима движения по контуру (Contour Mode).
Reference (Ссылка)		Выполняет движение по ссылке, например определение исходной или предельной позиции, используя осевые ресурсы. Движения по ссылке используются для инициализации системы управления движением и создания повторяемого положения по ссылке. Характеристики движения зависят от режима движения по ссылке (Reference Move Mode).
Capture (Захват)		Записывает положение энкодера на основании внешнего входного сигнала, например, состояния датчика. Вы можете использовать эту функцию для выполнения движения относительно зафиксированного положения, или простой записи положения энкодера.
Compare (Сравнение)		Синхронизирует двигатель с внешним процессом и задает положение энкодера. Когда достигнуто заданное положение, формируется определяемый пользователем импульс. Характеристики операции сравнения положений зависят от режима сравнения (Compare Mode).
Gearing (зубчатая передача)		Настраивает указанные оси для операции зубчатой передачи. Это синхронизирует движения подчиненной оси с движением главного устройства, которое может быть энкодером или траекторией другой оси. Движение подчиненной оси может выполняться с более высоким или более низким передаточным числом, чем главной. Например, каждый поворот главной оси может вызвать двойной поворот подчиненной. Тип операции зависит от режима зубчатой передачи (Gearing Mode).
Camming (эксцентрическая		Настраивает указанную ось для операций с эксцентричными объектами. Эти отношения автоматически обрабатываются LabVIEW NI SoftMotion, обеспечивая точное переключение

связь)		передаточных чисел. Подобные операции используются в приложениях, где подчиненная ось следует нелинейному профилю главного устройства. Тип операции зависит от режима (Camming Mode).
Read (Чтение)		Читает статусную информацию и данные с осей, координат, устройств обратной связи и других ресурсов. Используйте методы чтения для получения информации о ресурсах.
Write (Запись)		Записывает информацию в оси, координаты или устройства обратной связи. Используйте методы записи для записи информации в разные ресурсы.
Reset Position (Установка в нулевое положение)		Устанавливает в нулевое положение указанную ось или координаты.
Stop (Стоп)		Останавливает текущее движение по оси или координате. Характеристики останова зависит от его режима (Stop Mode).
Power (Питание)		Включает и выключает оси и/или приводы для указанных осевых или координатных ресурсов.
Clear Faults (Очистить сбои)		Очищает сбои LabVIEW NI SoftMotion.

Пример кода LabVIEW

В данном разделе приведен пример кода LabVIEW

Конечный автомат или диаграмма состояний – широко распространенная архитектура для приложений управления движением. Однако, хотя конечный автомат увеличивает гибкость приложения, появляются дополнительные сложности при использовании функциональных блоков управления движением в конечном автомате или диаграмме состояний.

При последовательном программировании переход между командами движения реализуется на основе выходов функциональных блоков "error", "aborted и "done". (Выход "busy" - логическая комбинация упомянутых выходов, так что для упрощения программирования вы можете также наблюдать за выходом "busy"). При программировании конечного автомата у вас могут быть параллельные состояния или события, которые влияют на движения и могут быть причиной выхода из состояния до завершения движения.

Например, вы можете построить логику, когда можно послать в систему команду "стоп". Если состояние останова выполняется параллельно состоянию движения, менеджер движений останавливает движение, и состояние движение возвращает сигнал "true" на выходе "aborted".

Вы можете также структурировать ваш код для завершения без ожидания возвращения команды "abort" функциональным блоком, например, создав в программе состояние ESTOP. В большинстве разработок приложений ESTOP немедленно закрывает все другие состояния и переходит в состояние безопасного отключения. В этом случае, поскольку функциональнее блоки не завершили работу чисто, перед запуском вам придется установить их в исходное состояние. Функциональный блок устанавливается в исходное состояние, когда на вход "execute" поступает сигнал "false".

Для решения этой проблемы вам необходимо убедиться, что на вход "execute" функционального блока поступает сигнал "false". Вы можете сделать это несколькими методами в LabVIEW, включая добавление логики для наблюдения данных о состоянии для каждого состояния движения, чтобы убедиться, что функциональный блок всегда очищен перед выполнением. Или вы можете просто всегда подавать сигнал "false" на вход "execute" в первой итерации состояния. Заметьте, что это отложит выполнение вашего движения на один цикл.

Рассмотрим пример простой машины состояний с четырьмя состояниями - простой, инициализация, движение, останов. В каждом состояние выход "busy" используется для определения, завершил ли работу функциональный блок.

Вы можете переходить в состояние останова движения из любого состояния. Это означает, что состояние останова может разрешать движение во время выполнения и заставить функциональный блок требовать сброса, прежде чем вы сможете запустить его снова. Чтобы добиться этого, машина состояний запрограммирована так, что на вход "execute" каждого функционального блока поступает сигнал "false" при первой итерации и сигнал "true" – в каждой последующей итерации. Это автоматически очищает функциональный блок, выполнение которого прервано.

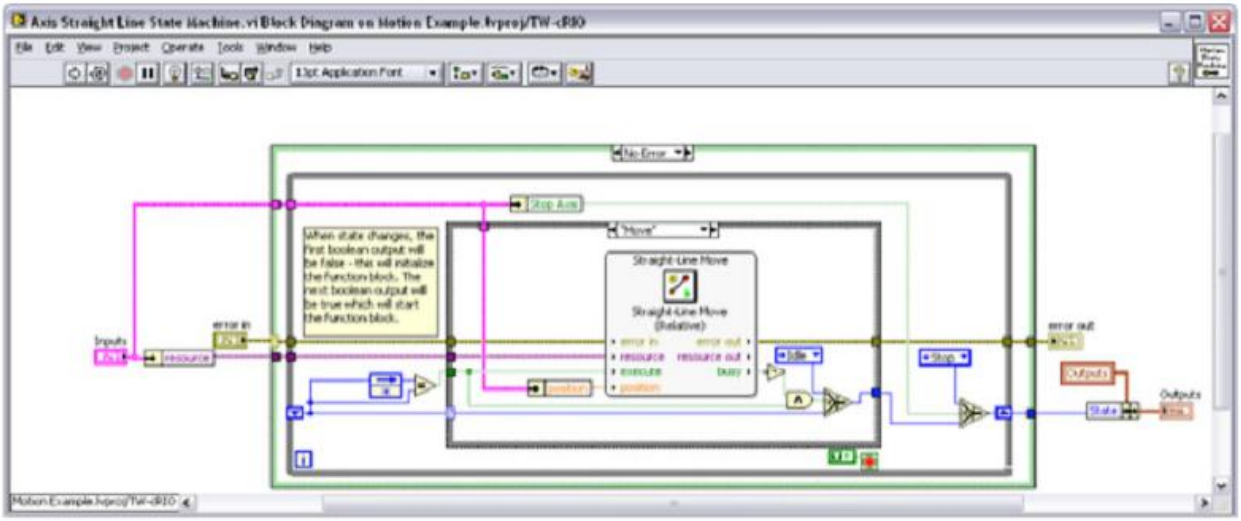


Рисунок 5.40. Конечный автомат с функциональными блоками управления движением

Вы можете поместить этот конечный автомат в стандартную архитектуру контроллера CompactRIO с состояниями инициализации и выключения.

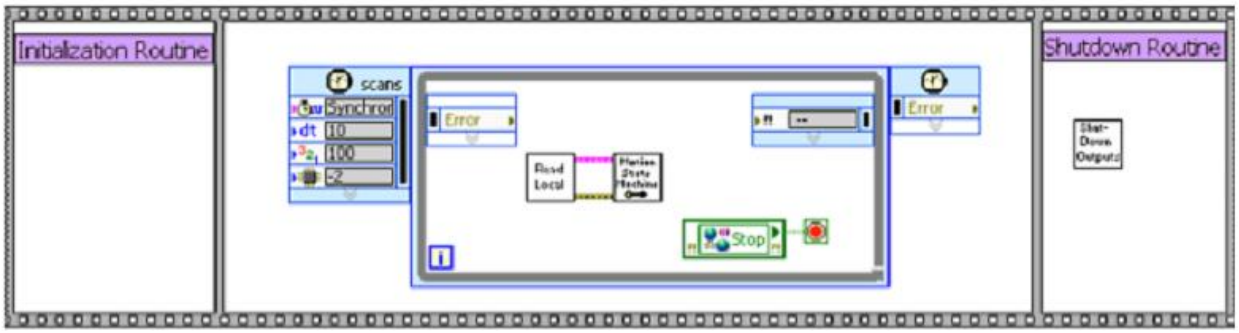


Рисунок 5.41. Конечный автомат с командами движения может быть помещена в стандартную архитектуру контроллера

РАЗДЕЛ 6
Разработка специализированных аппаратных средств с помощью LabVIEW FPGA

Расширение CompactRIO с помощью LabVIEW FPGA

До сих пор обсуждение архитектур в данном документе было сфокусировано на использовании встроенного в CompactRIO FPGA с зафиксированными свойствами и функционировании CompactRIO в режиме сканирования. В этом разделе рассказывается, когда, почему и как можно расширить способности FPGA. Режим сканирования поддерживает большинство модулей C-серии для CompactRIO и множество высокоуровневых функций, например, измерения с помощью квадратурного энкодера и генерацию ШИМ-сигналов, но FPGA способны на гораздо большее при проектировании с помощью LabVIEW FPGA.

С помощью LabVIEW FPGA вы можете:

- Собирать аналоговые сигналы с частотой в сотни килогерц
- Формировать необходимые цифровые импульсы с частотой до 40 МГц
- Реализовывать пользовательские протоколы цифровых коммуникаций
- Запускать контуры регулирования со скоростью в сотни килогерц
- Использовать модули, не поддерживающие режим сканирования, включая модули обслуживания протоколов CAN и PROFIBUS
- Реализовывать пользовательские режимы синхронизации, запуска и фильтрации

Поскольку структур FPGA в CompactRIO создается пользователем заново, вы можете одновременно запускать интерфейс сканирования и пользовательский код LabVIEW FPGA.

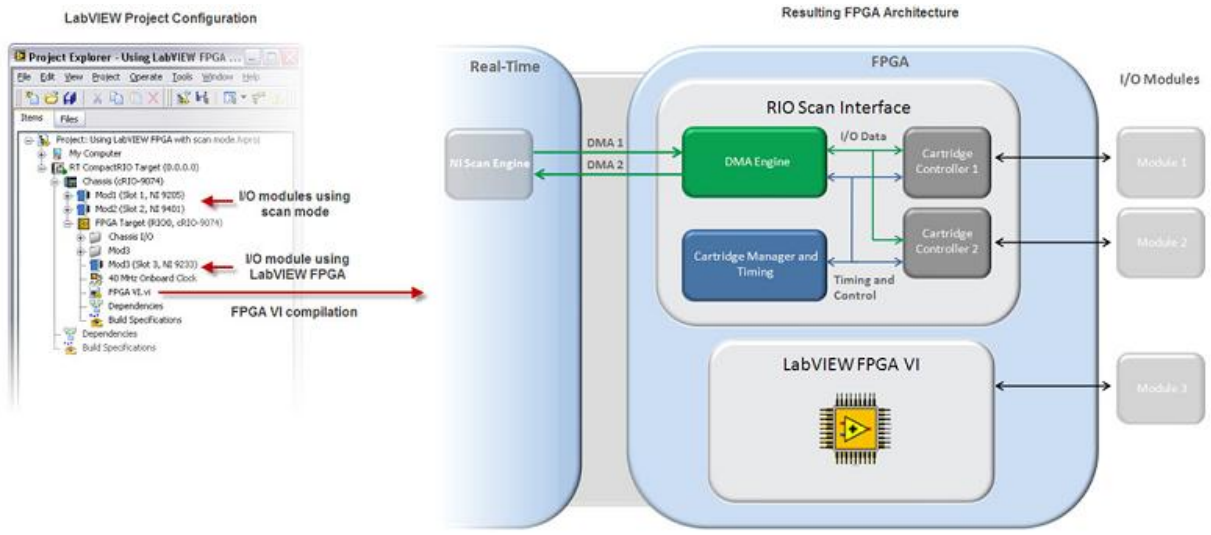


Рисунок 6.1. Вы можете дополнять возможности режима сканирования пользовательским кодом LabVIEW FPGA

LabVIEW Project Configuration – конфигурация проекта LabVIEW, I/O module using scan mode – модули ввода-вывода, использующие интерфейс сканирования, I/O module using LabVIEW FPGA - модули ввода-вывода, использующие LabVIEW FPGA, FPGA VI compilation – компиляция VI FPGA, Real Time – реальное время, NI Scan Engine – механизм сканирования, DMA – прямой доступ к памяти, Resulting FPGA Architecture – результирующая архитектура FPGA, RIO Scan Interface – интерфейс сканирования RIO, DMA Engine – механизм прямого доступа к памяти, I/O Data – данные ввода-вывода, Cartridge Manager and Timing – картридж управления и синхронизация, Timing and Control – синхронизация и управление, Cartridge Controller – картридж контроллера, I/O modules - модули ввода-вывода

Когда использовать LabVIEW FPGA

Вы должны дополнить сканирующий режим модулем LabVIEW FPGA по нескольким причинам. Комбинация в CompactRIO процессора реального времени и программируемых FPGA позволяет создать систему, использующую сильные стороны каждой из платформ. Процессор реального времени лучшего всего выполняет вычисления с плавающей точкой, анализ и периферийные коммуникации, например, работу с публикуемыми в сети переменными общего доступа и веб-службами. FPGA лучше для более мелких задач, которые требуют очень быстродействующей логики и точной синхронизации. Ниже приведен список сценариев, для которых нужно непосредственное программирование FPGA .

Высокоскоростной сбор и генерация сигналов (выше 1 кГц)

Режим сканирования CompactRIO оптимизирован для контуров регулирования с частотой до 1 кГц, но многие модули C-серии способны на сбор и генерацию данных с намного большими скоростями. Если вы хотите воспользоваться всеми возможностями этих модулей по сбору и генерации сигналов с частотой больше 1 кГц, вы можете использовать LabVIEW FPGA для сбора данных со скоростью, определяемой пользователем в зависимости от требований приложения.

Пользовательский запуск/временные диаграммы/синхронизация

С реконфигурируемыми FPGA вы можете создавать простые, улучшенные и прочие пользовательские схемы запуска, формирования временных диаграмм и синхронизации шасси или ввода-вывода. Это может быть так же сложно, как запуск пользовательского сообщения CAN при превышении порога сигналом аналогового сбора, или так же просто, как сбор вводимых значений по положительному фронту внешнего тактового генератора.

Аппаратная реализация анализа/генерации и совместной обработки

Многие датчики выдают больше данных, чем может быть обработано одним процессором реального времени. Вы можете использовать FPGA как сопроцессор для обработки или генерации сложных сигналов, освобождая процессор для других критических задач. Сопроцессор на базе FPGA часто используется в таких приложениях, как:

- Кодирующие/декодирующие датчики
  - – Тахометры
  - – Стандартные и/или пользовательские цифровые протоколы
- Обработка и анализ сигналов
  - – Спектральный анализ (быстрое преобразование Фурье и работа с окнами)
  - – Фильтрация, усреднение и т.п.
  - – Сжатие данных
  - – Интеграция IP сторонних фирм
- Симуляция датчиков
  - – Кулачковых и кривошипно-шатунных механизмов
  - – Измерительных преобразователей линейных перемещений (LVDT)
- Аппаратно-программное моделирование

Управление с наивысшей производительностью

Вы можете не только использовать FPGA для высокоскоростного сбора и генерации данных, но и реализовывать множество управляющих алгоритмов на FPGA. Вы можете использовать одноточечный ввод-вывод с многоканальным, настраиваемым ПИД-регулятором иди другим управляющим алгоритмом для реализации детерминированного управления с частотой до сотен килогерц.

Неподдерживаемые модули

Несколько модулей C-серии не поддерживают режим сканирования. Для таких модулей вы должны использовать LabVIEW FPGA при создании интерфейса между вводом-выводом и вашим приложением реального времени. Список модулей, которые поддерживают режим сканирования, приведен в документе "C Series Modules Supported by CompactRIO Scan Mode" (Модули C-серии, поддерживающие режим сканирования CompactRIO).

Неподдерживаемые целевые устройства

Целевые объекты CompactRIO с FPGA, содержащим 1M вентилей, не могут полностью поддерживать режим сканирования. Реализовать некоторые возможности режима сканирования на таких целевых устройствах можно с помощью LabVIEW FPGA. В статье базы знаний "Using CompactRIO Scan Mode with Unsupported Backplanes" (Применение режима сканирования CompactRIO в шасси, неподдерживающими такой режим) описывается, как использовать LabVIEW FPGA для создания пользовательского интерфейса сканирования при работе с подобными целевыми устройствами FPGA.

Обзор FPGA

Реконфигурируемый программируемый массив вентилей, или FPGA, - программируемый кристалл, состоящий из трех основных компонентов: логических блоков, программируемых соединений и блоков ввода-вывода.

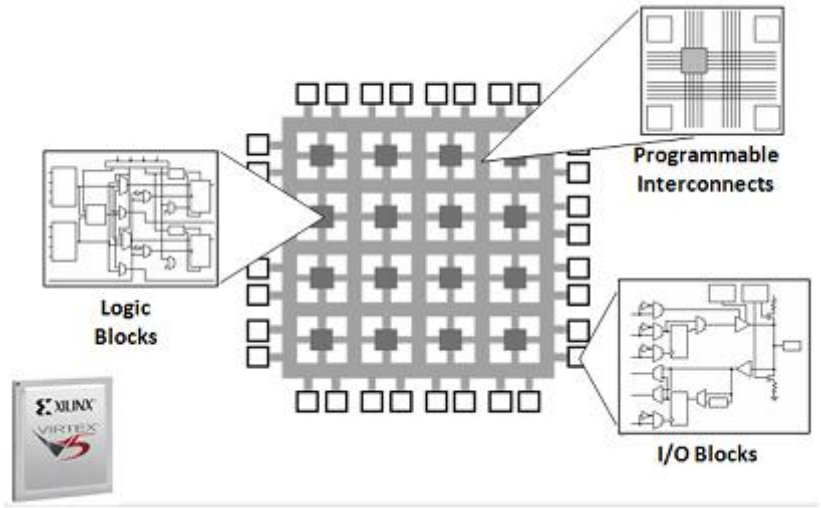


Рисунок 6.2. FPGA состоит из конфигурируемой логики и блоков ввода-вывода, связанных программируемыми межсоединениями
Logic blocks – логические блоки, Programmable interconnects – программируемые межсоединения, I/O blocks – блоки ввода-вывода

Логические блоки – это набор цифровых компонентов, таких, как таблицы преобразования, умножители и мультиплексоров, с помощью которых преобразуются и обрабатываются биты данных с целью получения программируемых результатов. Эти логические блоки взаимодействуют через программируемые межсоединения, которые представляют собой миниатюрную разновидность макета для перенаправления сигналов от одного логического блока соседнему. Программируемые соединения могут также маршрутизировать сигнал к блокам ввода-вывода, которые подключены к выводам кристалла для двусторонней коммуникации с окружающими схемами. По сути, FPGA - пустые кремниевые холсты, которые вы можете запрограммировать на выполнение функций любых цифровых аппаратных средств. Традиционно программирование кристаллов FPGA было сложным, и поэтому доступным только опытным разработчикам и инженерам по аппаратным средствам. National Instruments абстрагировала программирование таких устройств с помощью графического интерфейса LabVIEW FPGA, так что практически кто угодно может воспользоваться преимуществами и мощью этих реконфигурируемых чипов.

Поскольку в VI LabVIEW FPGA синтезируются физические аппаратные вентили, соединяемые программируемыми связями, процесс компиляции FPGA отличается от процесса компиляции в традиционном LabVIEW для Windows или LabVIEW Real-Time. При разработке кода FPGA вы пишете тот же код LabVIEW, что и для другого любого целевого устройства, но когда вы нажимаете кнопку Run, внутри LabVIEW выполняется другой процесс. Сначала LabVIEW FPGA генерирует код VHDL и передает его компилятору Xilinx. Затем компилятор Xilinx синтезирует VHDL, помещает и подключает все синтезированные компоненты к битовому файлу. Наконец, битовый файл загружается в FPGA, и FPGA принимает заданный вами при программировании облик. Этот процесс более сложен, чем другие компиляции LabVIEW, и может занять от 5 до 10 минут, вплоть до нескольких часов для сложных приложений. Из-за относительно большого времени компиляции вам стоит потратить больше времени на отладку и проектирование кода LabVIEW FPGA, прежде, чем попытаться этот код откомпилировать.

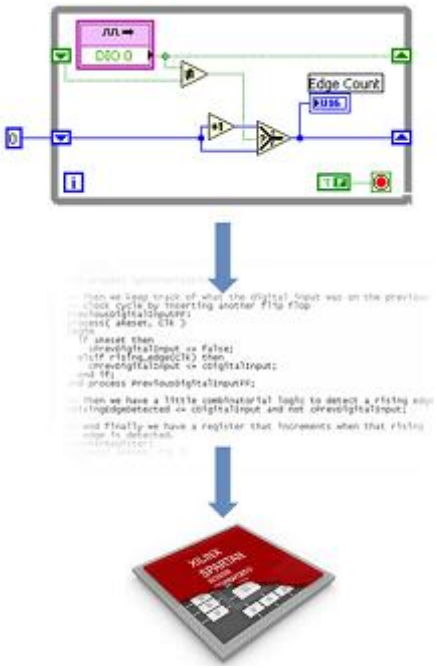


Рисунок 6.3. Компилятор LabVIEW FPGA скрытно преобразует код LabVIEW в VHDL и вызывает средства компиляции Xilinx для генерации битового файла, который выполняется непосредственно в FPGA

Преимущества FPGA

Поскольку код LabVIEW FPGA выполняется непосредственно на аппаратном уровне, вы можете воспользоваться следующими преимуществами проекта, реализуемого на FPGA.

Высокая надежность

Код LabVIEW FPGA, выполняемый на FPGA, обладает высокой надежностью, потому что логика компилируется в аппаратно реализуемое устройство. Когда вы запрограммируете FPGA, он становится кристаллом со всей свойственной микроэлектронному кристаллу надежностью.

Высокий детерминизм

Основанные на процессорах системы часто включают несколько уровней абстрагирования, чтобы помочь распределить задачи и разделить ресурсы между несколькими процессами. Уровень драйверов управляет аппаратными ресурсами, а операционная система управляет памятью и загрузкой процессора. Для любого процессорного ядра в каждый момент времени может выполняться только одна инструкция. В системах реального времени при программировании с хорошей иерархией приоритетов, джиттер можно уменьшить до конечного максимума. FPGA не используют операционных систем. Это повышает надежность при действительно параллельном выполнении, а детерминировано функционирующие аппаратные средства выделяются для каждой задачи. Для критических компонентов вашей разработки, основанной на NI FPGA, вы можете добиться точности аппаратных средств в 25 нс.

Истинный параллелизм

Многопоточные приложения разделяются на параллельные фрагменты кода, которые выполняются циклически, создавая видимость параллельного выполнения.

Многоядерные процессоры расширяют эту идею, позволяя многопоточным приложениям одновременно выполнять действительно параллельный код. Количество параллельных фрагментов ограничено количеством ядер процессора. Поскольку FPGA реализует параллельный код как параллельные схемы аппаратуры, вы не ограничены количеством ядер процессора, и все

параллельные фрагменты кода во всем приложении FPGA могут выполняться одновременно. При реализации принципов конвейерной обработки в FPGA может улучшиться даже выполнение традиционных последовательных операций.

Реконфигурируемость

Реконфигурируемые кристаллы FPGA способны выдержать все будущие модификации, которые вам могут потребоваться. По мере развития вашего продукта вы можете вводить функциональные дополнения, не тратя времени на перестройку аппаратных средств или изменение топологии печатной платы. Это особенно важно для протоколов промышленных коммуникаций. Поскольку промышленные протоколы развиваются и совершенствуются, вы можете изменить реализацию этого протокола в FPGA для поддержки самой последних технологий и изменений.

Мгновенная загрузка

Поскольку код LabVIEW FPGA выполняется напрямую в FPGA без участия операционной системы, код, который вы загрузили во флэш-память FPGA, начинает работу через миллисекунды после подачи питания на шасси CompactRIO. Этот код может начать выполнение контура регулирования или установить стартовые значения на выходах.

Программирование в LabVIEW FPGA

Когда вы столкнетесь с необходимостью расширить управляющую архитектуру аппаратных средств CompactRIO, добавив программирование FPGA пользователем, примите во внимание, что аппаратура CompactRIO может быть запрограммирована в трех режимах: режиме сканирования, режим "чистого" интерфейса с FPGA и гибридный режим. Большинство управляющих архитектур этого документа для извлечения данных ввода-вывода были сфокусированы на режиме сканирования. Вы узнали, что эффективная технология для этого режима программирования – фиксированная структура FPGA, которая упорядочивает данные в FPGA и на хосте реального времени. В этом разделе рассматривается, как использовать FPGA в гибридном режиме для прямого доступа к вводу-выводу с возможностями программирования FPGA, оставляя другие средства ввода-вывода в режиме сканирования. В гибридном режиме вы можете использовать модель программирования FPGA для управления буферизированным сбором сигналов, оперативной обработки данных и работы с некоторыми модулями, не поддерживающими режим сканирования.

Рисунок 6.4. В гибридном режима вы можете выбрать сбор данных с физических каналов ввода-вывода с помощью режима сканирования или через интерфейс FPGA

Physical I/O – физический ввод-вывод

Гибридный режим CompactRIO

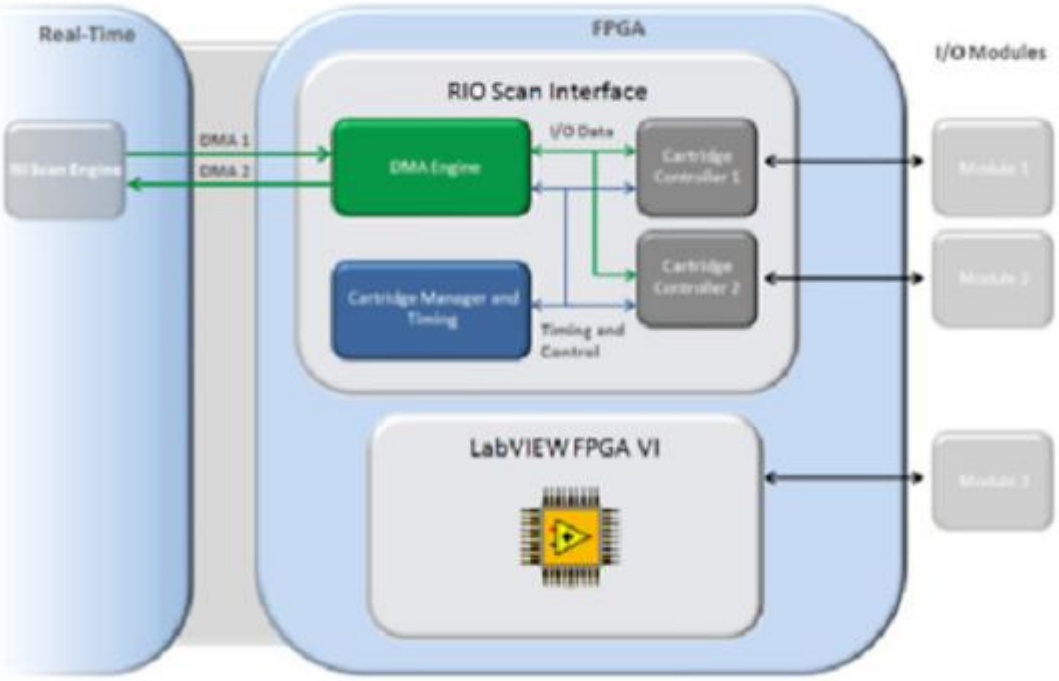


Рисунок 6.5. После активации гибридного режима создайте FPGA VI для интерфейса с модулем и передачи данных хосту реального времени

Real Time – реальное время, NI Scan Engine – механизм сканирования, DMA – прямой доступ к памяти, Resulting FPGA Architecture – результирующая архитектура FPGA, RIO Scan Interface – интерфейс сканирования RIO, DMA Engine – механизм прямого доступа к памяти, I/O Data – данные ввода-вывода, Cartridge Manager and Timing – управление и синхронизация картриджей, Timing and Control – синхронизация и управление, Cartridge Controller – картридж контроллера

В гибридном режим вы можете продолжать использовать интерфейс сканирования NI RIO на некоторых модулях, программируя другие напрямую в FPGA. Активируйте программирование FPGA для конкретного модуля, перетащив из проекта модуль, который расположен в шасси CompactRIO под целевым устройством FPGA. Сделав это, вы можете программировать FPGA для создания пользовательского кода, выполняющегося параллельно с интерфейсом сканирования в других модулях.

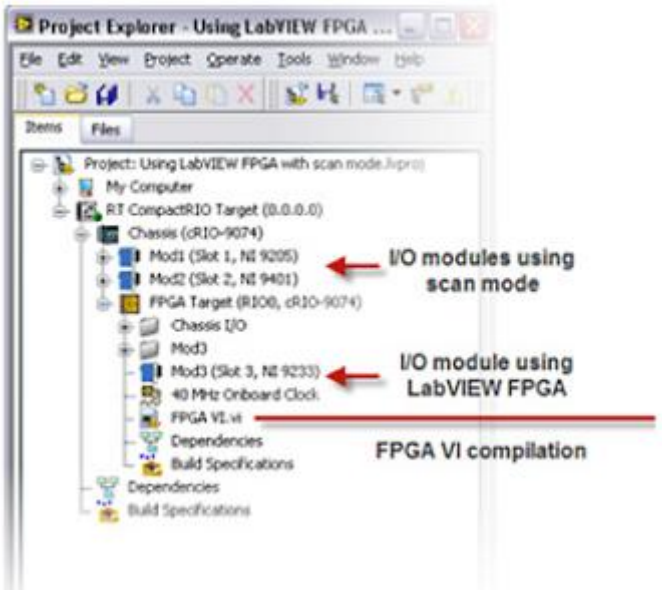


Рисунок 6.6. В этом проекте в гибридном режиме показаны некоторые модули в шасси CompactRIO, а некоторые (модуль 3 на этом рисунке) – в FPGA
I/O module using scan mode – модули ввода-вывода, использующие интерфейс сканирования, I/O module using LabVIEW FPGA - модули ввода-вывода, использующие LabVIEW FPGA, FPGA VI compilation – компиляция VI FPGA.

Использование гибридного режима требует знания программирования FPGA и применения интерфейса FPGA на стороне контроллера реального времени, потому что приложения, требующие гибридного режима, обычно значительно более сложны. Есть два способа передачи данных между FPGA и стороной реального времени: хост-интерфейс и определенные пользователем переменные ввода-вывода.

Хост-интерфейс – низкоуровневый интерфейс, предлагающий большую гибкость, но и более сложный. Определенные пользователем переменные ввода-вывода позволяют FPGA непосредственно читать и записывать одноточечные значения в табличную память реального времени, но пересылка данных возможна для ограниченных типов данных, поддерживаемых только в режиме сканирования.

Далее приведены пять примеров приложений

1. Асинхронная оперативная обработка данных в FPGA, по точкам передающая значения ввода-вывода приложению реального времени с использованием хост-интерфейса
2. Синхронная оперативная обработка данных в FPGA, по точкам передающая значения ввода-вывода приложению реального времени с использованием хост-интерфейса
3. Асинхронная оперативная обработка данных в FPGA, по точкам передающая значения ввода-вывода приложению реального времени с использованием пользовательских переменных ввода-вывода
4. Синхронная оперативная обработка данных в FPGA, по точкам передающая значения ввода-вывода приложению реального времени с использованием пользовательских переменных ввода-вывода
5. Высокоскоростная потоковая передача данных между FPGA и приложением реального времени с использованием хост-интерфейса

Пример – простой обмен данными по точкам между FPGA и системой реального времени с использованием хост-интерфейса

В данном разделе приведен пример кода LabVIEW

Основы программирования FPGA

Режим сканирования – метод сбора данных по точкам, который позволяет работать в контуре с обратной связью на частоте до 1 кГц. Однако большинство модулей обладают гораздо большими скоростями. Для опроса входов модуля с частотой выше 1 кГц выполните следующие действия:

1. Активируйте программирование FPGA модуля, поместив модуль в раздел FPGA проекта (как было описано ранее)
2. Создайте VI FPGA, который:

   a. читает данные из узла ввода-вывода FPGA

   b. использует структуры и логику LabVIEW для определения скорости сбора данных

   c. проверяет ошибочные состояния, чтобы убедиться в целостности данных

   d. обрабатывает данные локально или упаковывает их в FIFO с каналом прямого доступа в память для обработки на хосте
3. Разработайте хост-VI, который читает упакованные данные из FIFO с каналом прямого доступа в память или результаты функций оперативной обработки данных в FPGA

Чтение из узла ввода-вывода

Для создания нового VI FPGA щелкните правой кнопкой по объекту FPGA в проекте и выберите **New»VI**. Для каждого модуля в разделе "FPGA Target" вы должны увидеть соответствующие папки, содержащие все точки ввода-вывода модуля (если их нет, щелкните правой кнопкой по модулю, выберите **New»FPGA I/O** (Новый»Ввод-вывод FPGA) и добавьте все доступные ресурсы). Перетащите точку ввода-вывода на блок-диаграмму, чтобы создать узел ввода-вывода.

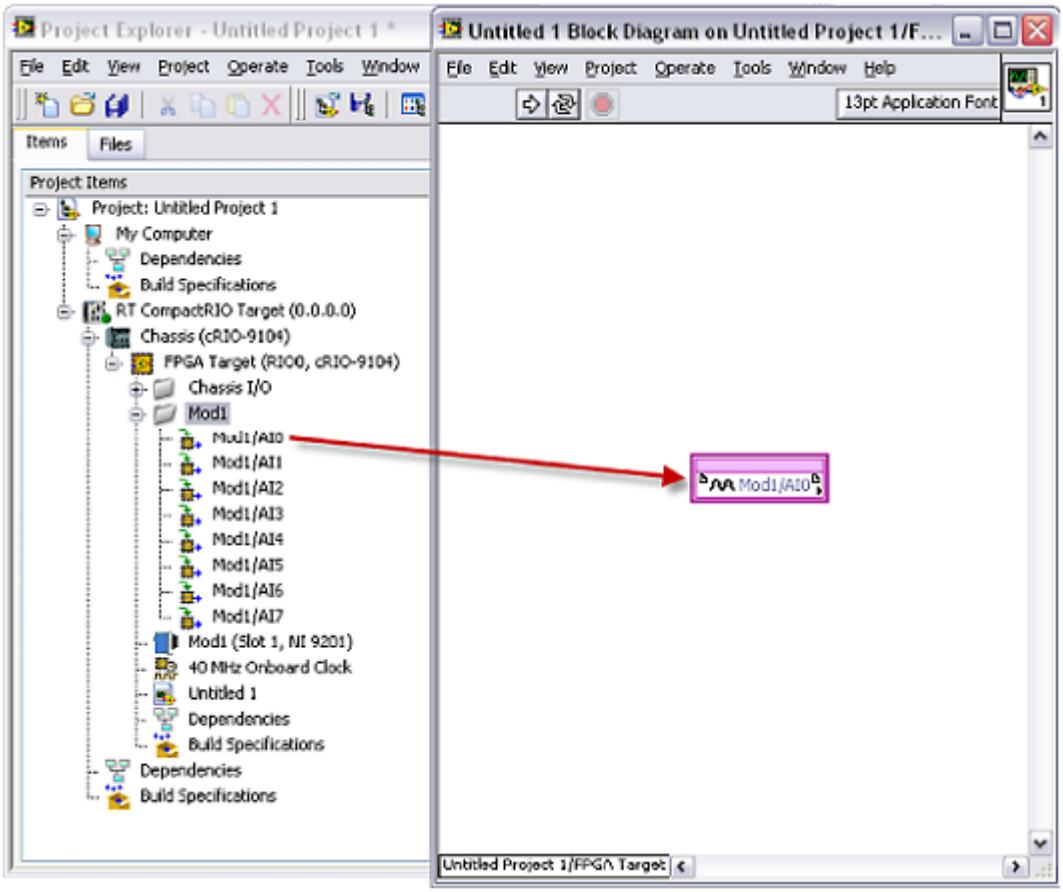


Рисунок 6.7. Перетащите узел ввода-вывода на блок-диаграмму FPGA

Запуск этого VI начнет процесс компиляции. Во время разработки вы можете использовать симуляцию FPGA. Чтобы переключить целевое устройство FPGA в режим симуляции, щелкните правой кнопкой и выберите **Execute VI on»Development Computer**

with Simulated I/O (Выполнение VI на»Компьютере разработчика с симуляцией ввода-вывода).

Использование структур и логики LabVIEW для задания скорости сбора данных и запуска

Узел ввода-вывода FPGA возвращает единственный отсчет данных на канал. Для сбора их с требуемой частотой, поместите узел ввода-вывода FPGA в цикл и используйте таймер цикла для задания периода сбора данных. Хорошей практикой является использование структуры последовательности flat sequence, чтобы убедиться, что таймер цикла вызывается до любого другого кода в цикле, например, до узла ввода-вывода FPGA.

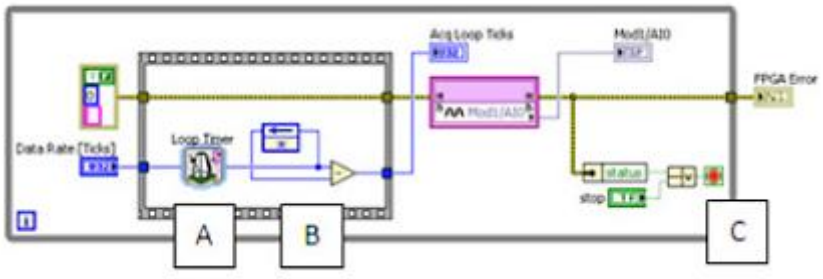


Рисунок 6.8. Сбор данных с помощью модуля аналогового ввода

Далее объясняется, что делает код в этом примере:

A. Когда таймер цикла вызывается в первый раз, он запоминает начальное время и немедленно выполняет последующий код. При следующем вызове он ждет, пока пройдет заданное время, прежде чем выполнять следующий код. Если выполнение кода в цикле превышает заданный период цикла, таймер цикла немедленно сбрасывается и использует текущее время как новую точку отсчета для последующих итераций. Используйте терминал "count" (счет) таймера цикла для задания периода отсчетов цикла. Интервал между отсчетами измеряется в периодах (тиках - ticks) тактового генератора FPGA, который по умолчанию работает с частотой 40 МГц. Например, реализация задержки между отсчетами в 20 000 тиках с частотой 40 МГЦ будет означать частоту отсчетов 2 кГц.

B. Таймер цикла гарантирует, что цикл while не запустится быстрее, чем заданный период цикла; однако он не сообщит, если цикл while работает медленнее, чем требуется. При разработке и тестировании вашего FPGA кода хорошей практикой считается измерение времени выполнения цикла, чтобы убедиться, что данные собираются с правильной скоростью. Простой узел обратной связи (Feedback Node) после таймера цикла используется для вычитания количества тиков последовательных итераций цикла.

Оперативная обработка данных в FPGA

В зависимости от приложения хост реального времени может нуждаться только в обработанных характеристиках собранных данных. Иногда может оказаться полезным передать задачи обработки сигналов, такие, как фильтрацию, усреднение или измерения в FPGA. Например, вы можете использовать FPGA для сбора данных, вычисления среднеквадратичного значения и передачи результата в контролер реального времени. Приведенный ниже код FPGA считывает канал 0 модуля 1, передает данные по конвейеру, вычисляет среднеквадратичное значение и передает результат в программу реального времени через индикатор FPGA "RMS result ch0".

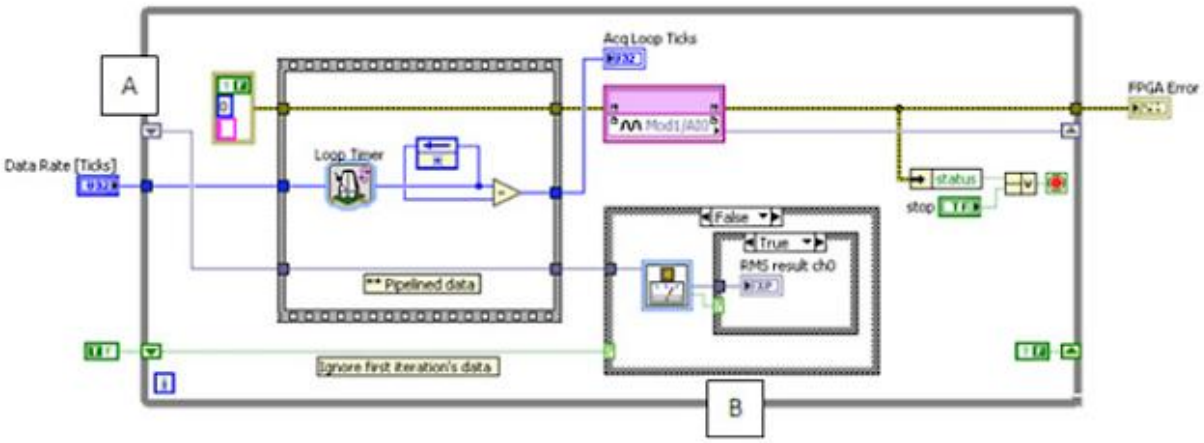


Рисунок 6.9. Конвейерная передача данных программе оперативной обработки

Что делает код в этом примере:

A. Выполнение узла ввода-вывода может занять значительную часть интервала времени между отчетами, особенно при высокой частоте дискретизации. Соответственно, выполнение оперативной обработки данных последовательно с узлом ввода-вывода может привести к тому, что цикл не успеет завершиться за требуемый период времени. Поскольку FPGA могут легко выполнять параллельную логику, полезно передать собранные отсчеты данных в сдвиговый регистр для обработки в следующей итерации цикла. Эта техника, известная как конвейеризация, разрешает параллельную обработку собранных данных и помогает минимизировать риск недозагрузки модулей, когда модуль производит отсчеты медленнее, чем требуется.

B. Поскольку собранные данные не обрабатываются до следующей итерации цикла, первой итерации цикла не достается данных для обработки. Структура case отключает оперативную обработку данных и игнорирует данные первой итерации. Все последующие итерации выполняют оперативную обработку конвейеризованных данных.

Проверка ошибочных состояний: недозагрузка модулей

Как обсуждалось ранее, недозагрузка модуля - состояние, когда один или больше периодов дискретизации длиннее, чем требовалось. Это происходит, если оперативная обработка данных заняла больше времени, чем требуемый интервал времени между отсчетами, или если пользователь задал частоту взятия отсчетов больше, чем модуль поддерживает. Особенно важно при разработке контролировать частоту выполнения цикла и отслеживать состояние недозагрузки.

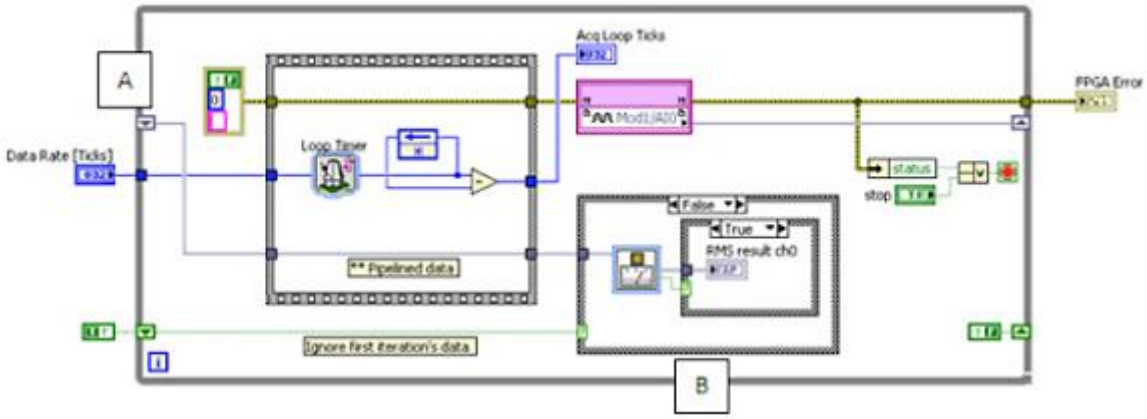


Рисунок 6.10. Проверка недозагрузки модулей в FPGA

Что делает код в этом примере:

A. Добавив несколько тиков к требуемому периоду цикла, вы зададите порог недозагрузки, с которым сравнивается реальная длительность цикла. Реальная длительность цикла также отображается в FPGA VI на соответствующем индикаторе для того, чтобы контролировать в хост-приложении реального времени.

B. Поскольку это связано с итерациями цикла и возможностью опустошения конвейера, организованного на сдвиговых регистрах таймера цикла, не начинайте проверку недозагрузки до второй итерации.

C. Состояние недозагрузки модуля на самом деле не останавливает цикл сбора данных, оно просто защелкивает булевский индикатор. Приложение реального времени проверяет индикатор FPGA и принимает решение о направлении действий. Поскольку FPGA обновляет индикатор “Module Underflow” быстрее, чем приложение реального времени может его проверять, важно защелкивать флаг недозагрузки, используя узел обратной связи и вентиль OR. Защелкнув его значение, вы удостоверитесь, что даже если недогрузка произойдет во время сбора данных, индикатор останется истинным, и приложение реального времени не пропустит сигнал предупреждения.

Использование хост-интерфейса и программе реального времени для связи с LabVIEW FPGA

Программа реального времени должна передавать данные по точкам и потоки данных, а также устанавливать и очищать прерывания с FPGA. Вы можете выполнить все это, используя интерфейс FPGA с хостом.

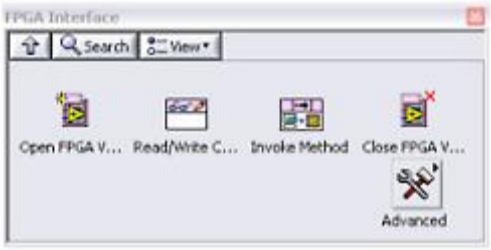


Рисунок 6.11. Палитра интерфейса с FPGA

Open FPGA VI Reference (Открытие ссылки на VI FPGA)

Открывает ссылку на FPGA VI или битовый файл и выбранный целевой объект FPGA. Щелкните правой кнопкой мыши по функции Open FPGA VI Reference и выберите Configure Open FPGA VI Reference (Конфигурирование ссылки открытия на FPGA VI) из контекстного меню для отображения соответствующего диалогового окна. Выберите ваш FPGA VI в этом окне. Вы можете также настроить, будет ли FPGA VI запускаться автоматически.

Read/Write Control (Управление чтением/записью)

Считывает или записывает значение элемента индикации/управления на лицевой панели FPGA VI целевого устройства FPGA. Это может быть условие запуска, частота дискретизации или другие данные или параметры, устанавливаемые элементам управления или индикатором в FPGA VI.

Invoke Method (Вызов метода)

Вызов из хост-VI метода интерфейса FPGA или действия в FPGA VI. Используйте эти методы для выполнения следующих операций: загрузки, прерывания, сброса и запуска VI FPGA на целевом устройстве; ожидания подтверждения запрошенного VI FPGA прерывания; чтения и записи в DMA FIFO (буфера FIFO с прямым доступом в память). Методы, которые вы можете выбрать, зависят от целевого устройства и FPGA VI. Вы должны подключить вход FPGA VI Reference In для просмотра доступных методов в контекстном меню.

Close FPGA VI Reference (Закрытие ссылки на VI FPGA)

Закрывает ссылку на FPGA VI и, возможно, сбрасывает или прерывает выполнение VI. В коде программы реального времени вы открываете ссылку на файл FPGA из подпрограммы инициализации. Сконфигурируйте ее так, чтобы FPGA VI запускался автоматически. Это загружает битовый файл FPGA в FPGA и начинает его выполнение. Поскольку битовый файлы FPGA содержит и код LabVIEW FPGA, и код модулей для режима сканирования, этот VI должен выполняться для связи с любым типом ввода-вывода в вашей системе.

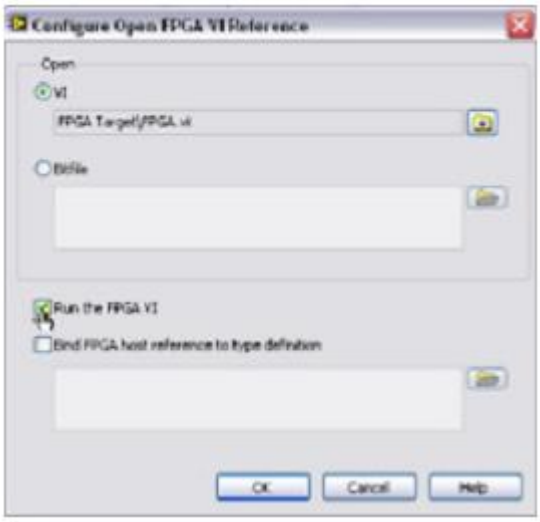


Рисунок 6.12. В диалоге конфигурирования открытия ссылки на FPGA вы можете выбрать VI или битовый файл и настроить его запуск при вызове

Далее перетащите синхронизируемый цикл для создания задачи коммуникации "FPGA Comm Task". В этом цикле прочитайте элементы лицевой панели FPGA VI. Выполните любые необходимые проверки на ошибки и, если данные действительны, передайте в табличную память, используя переменную общего доступа типа Single Process с FIFO реального времени.

Наконец в подпрограмме отключения закройте ссылку на VI FPGA. Не забудьте настроить блок так, чтобы FPGA VI НЕ останавливался. Если FPGA VI останавливается, он также останавливает ввод-вывод для модулей в режиме сканирования, что может привести к гонкам с другими настройками в подпрограмме отключения.

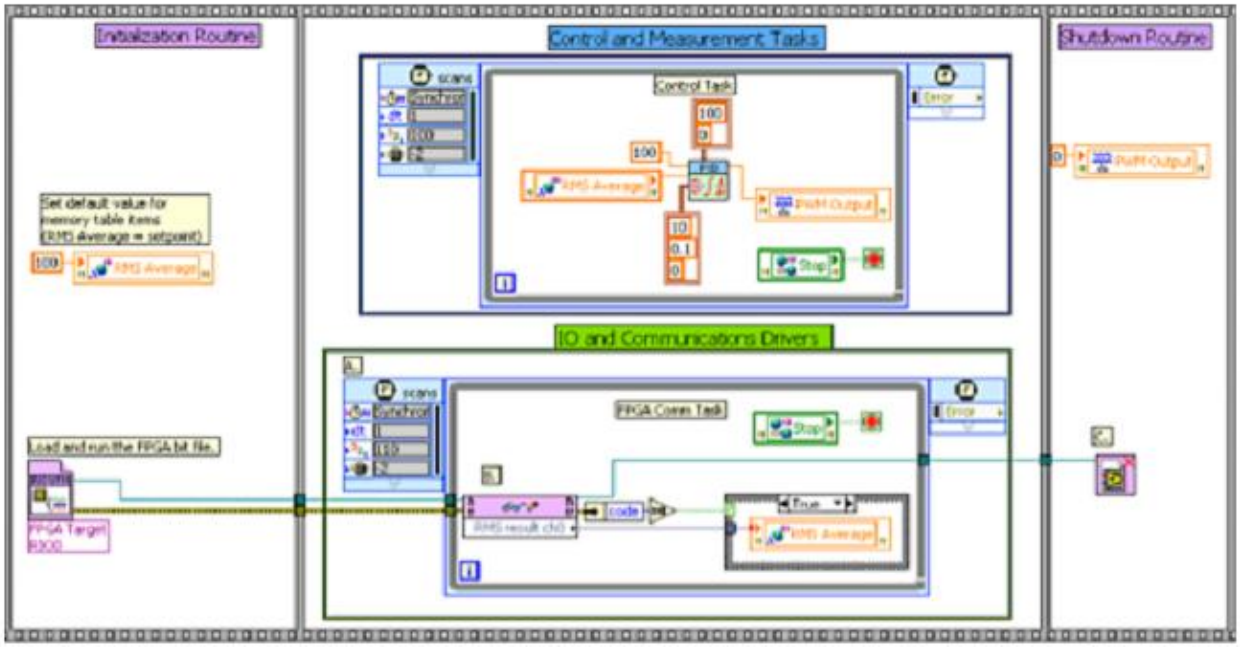


Рисунок 6.13. Завершенное приложение реального времени обменивается с кодом FPGA значениями по точкам и помещает данные в табличную память

Пример – синхронизированный обмен данными по точкам между FPGA и приложением реального времени

В данном разделе приведен пример кода LabVIEW

Для некоторых управляющих приложений вам может понадобиться синхронизировать цикл сбора и обработки данных в FPGA с циклом в процессоре реального времени. Это обеспечивает постоянство задержки ввода-вывода и уменьшает джиттер. Простой и эффективный способ синхронизации FPGA и кода реального времени – использование прерываний.

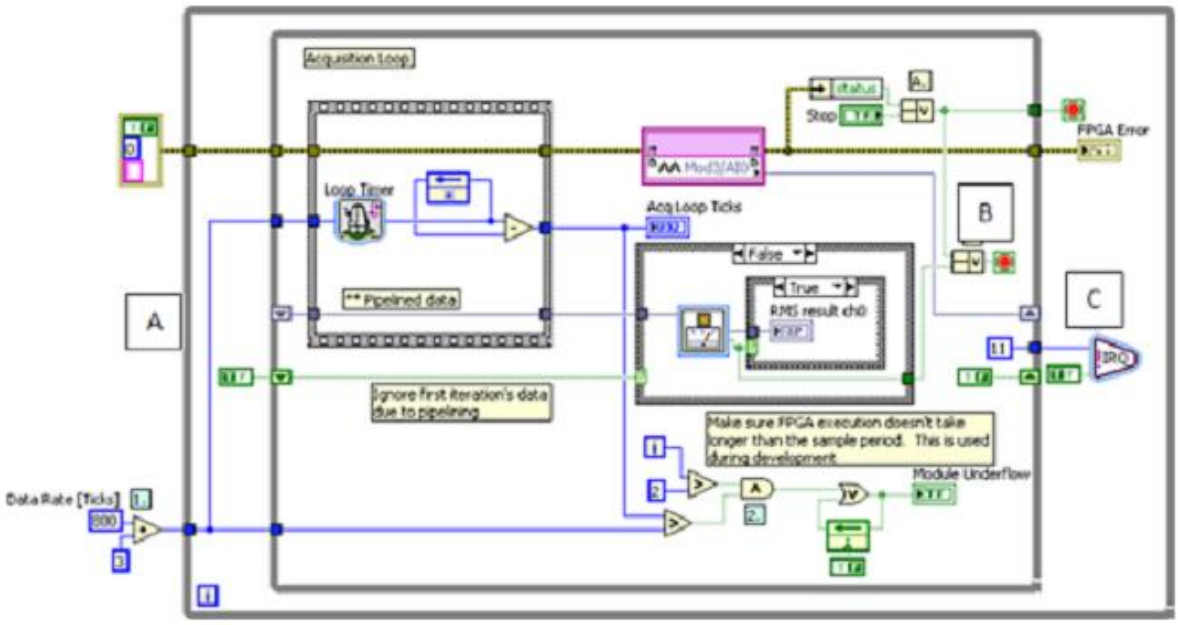


Рисунок 6.14. В LabVIEW FPGA вы можете устанавливать прерывания для контроллера реального времени и ожидать их обработки

Вы можете использовать простое прерывание для синхронизации выполнения сбора данных FPGA с контуром регулирования реального времени.

Что делает код в этом примере:

A. Внутренний цикл управляет частотой дискретизации сигналов, а внешний цикл - синхронизацией каждой подпрограммы обработки с использованием прерываний.

B. Как только оператор вычисления среднеквадратичного выдаст действительный результат, внутренний цикл завершает работу.

C. Внешний цикл устанавливает прерывание для контроллера реального времени и ожидает подтверждения от приложения реального времени.

В приложении реального времени вы можете модифицировать задачу обмена данными "FPGA Comm Task" для обработки этого прерывания. Приложение ждет прерывания от FPGA ,чтобы убедиться, что новые данные готовы. Затем оно считывает данные и подтверждает прерывание, чтобы вызвать запуск FPGA для измерения и обработки нового значения.

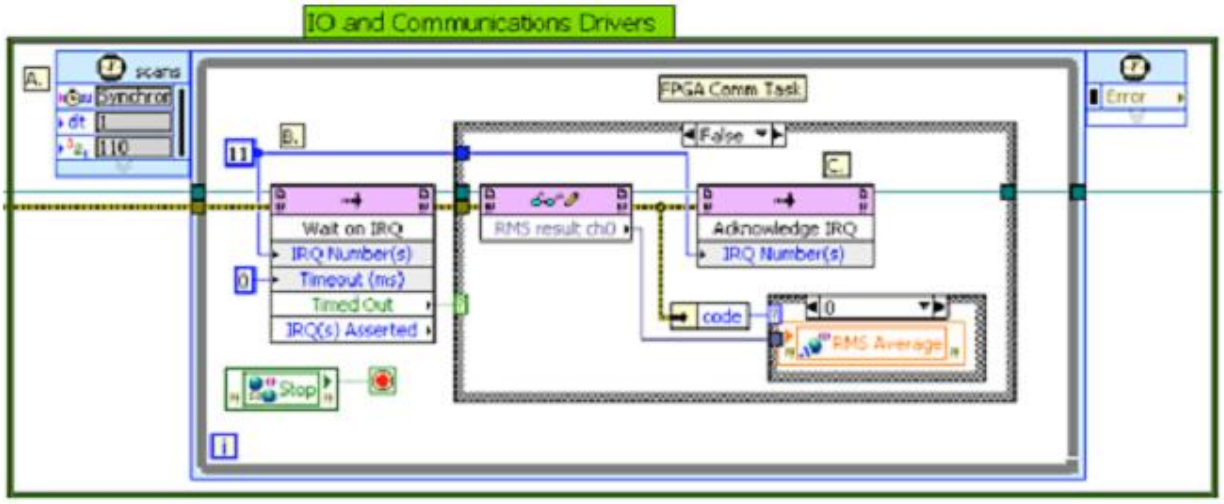


Рисунок 6.15. В коде реального времени подождите прерывания, считайте данные, и подтвердите прерывание

Пример - простой обмен данными по точкам между FPGA и приложением реального времени с использованием определяемых пользователем переменных ввода-вывода

В данном разделе приведен пример кода LabVIEW

Определяемые пользователем переменные ввода-вывода для данных специализированного ввода-вывода FPGA

Вместо использования функций интерфейса с хостом вы можете реализовать определяемые пользователем переменные ввода-вывода в механизме сканирования для передачи данных между FPGA VI и хост-VI реального времени. Использование механизма сканирования уменьшает размер кода, который вы должны разработать для доступа и передачи связанных наборов данных между FPGA и приложениями реального времени. При использовании детерминированных целевых устройств расширения ввода-вывода NI 9144, определяемые пользователем переменные ввода-вывода – единственный поддерживаемый механизм для передачи данных из пользовательского кода FPGA по сети.

Создание определяемых пользователем переменных ввода-вывода

Щелкните правой кнопкой мыши по элементу шасси в окне обозревателя проекта и выберите New»User-Defined Variable (Новый»Определяемые пользователем переменные) из контекстного меню. Поскольку все переменные ввода-вывода – однонаправленные, вы должны настроить направление каждой определяемой пользователем переменной ввода-вывода - или от FPGA к хосту, или от хоста к FPGA .

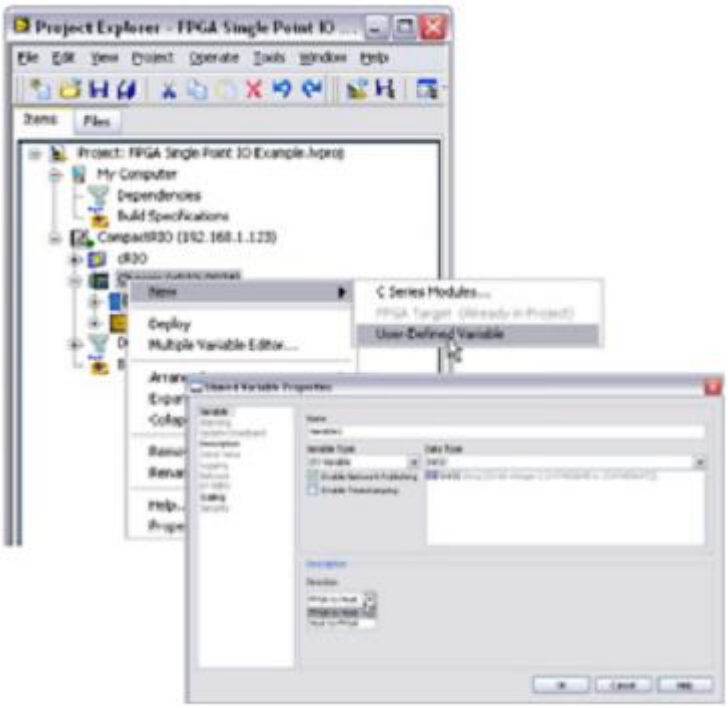


Рисунок 6.16. Определяемые пользователем переменные обеспечивают связь между FPGA и кодом реального времени

В коде FPGA вместо использования элементов управления и индикаторов лицевой панели вы можете перетащить на блок-диаграмму переменную ввода-вывода.

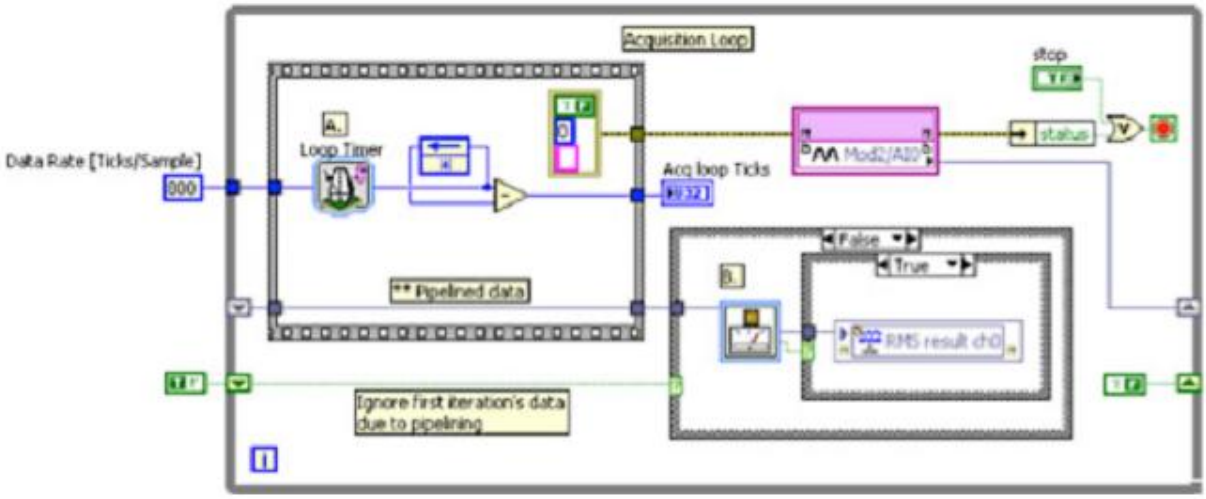


Рисунок 6.17. В коде FPGA вы можете писать напрямую в определяемую пользователем переменную

Переменная ввода-вывода читается в процессе сканирования и автоматически заносится в табличную память контроллера реального времени. В коде реального времени вы можете исключить цикл коммуникации с FPGA и просто читать переменную ввода-вывода из главного контура регулирования. Вам по-прежнему потребуется загружать и закрывать ссылку на код FPGA.

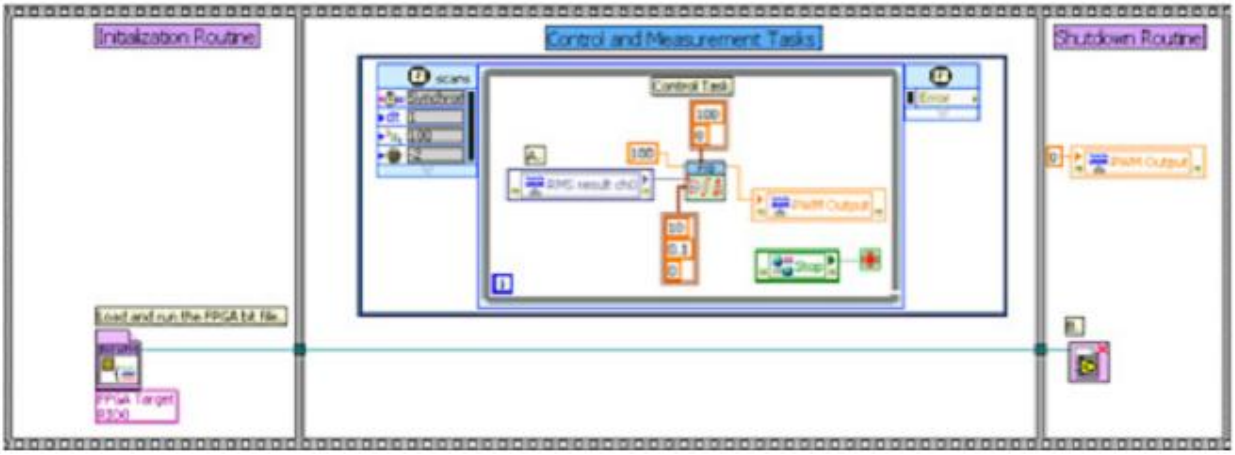


Рисунок 6.18. Данные переменной автоматически сканируются и заносятся в табличную память, так что вы можете непосредственно читать значения из управляющего кода

Пример - синхронизированный обмен данными между FPGA и приложением реального времени с использованием определяемых пользователем переменных ввода-вывода

В данном разделе приведен пример кода LabVIEW

Механизм сканирования предоставляет FPGA информацию о синхронизации. Вы можете получить доступ к этой информации, добавив элемент Scan Clock I/O (Сканирующие синхроимпульсы ввода-вывода) на блок-диаграмму FPGA VI. Этот элемент ввода-вывода передает информацию о синхронизации из механизма сканирования, например, количество импульсов FPGA с высоким уровнем сигнала, в FPGA VI. Используйте эту информацию о синхронизации для разработки приложения, гарантирующего когерентность наборов данных, которые вы передаете между FPGA и хост-VI реального времени. Используйте элемент Scan Clock I/O, его методы и свойства для наблюдения за информацией поставляемой механизмом сканирования.

Элемент Scan Clock I/O

Используйте узел ввода-вывода FPGA (FPGA I/O Node) для доступа к элементу Scan Clock I/O. Вы можете перетащить Scan Clock I/O из папки шасси ввода-вывода в проекте на блок-диаграмму FPGA VI, или поместить узел ввода-вывода FPGA на блок-диаграмму, щелкнуть по секции element узла ввода-вывода и выбрать из контекстного меню Chassis I/O»Scan Clock. Сигнал Scan Clock I/O является истинным, когда механизм сканирования не передает данные между FPGA VI и хост-VI реального времени, при этом можно "безопасно" записывать значения в переменные ввода-вывода. Если вы начнете запись, когда значение этого сигнала ложно, это не повлияет на детерминизм, но у вас не будет гарантий, что данные будут переданы при следующем сканировании.

Используйте узел FPGA I/O Method для ожидания положительного фронта импульса сканирования. Сделав это, вы сможете разработать код FPGA, который начинает выполнение, как только механизм сканирования прекращает передачу данных.

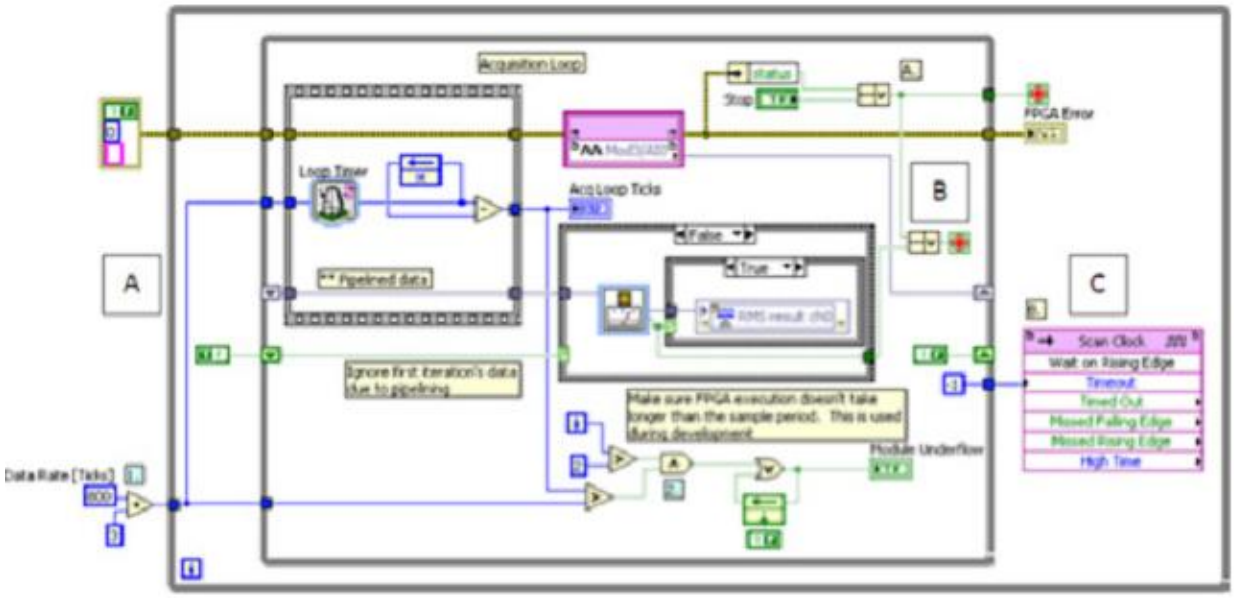


Рисунок 6.19. В LabVIEW FPGA вы можете подождать прихода положительного фронта импульса сканирования для синхронизации FPGA и кода реального времени

Для синхронизации FPGA, чтобы новые данные всегда были доступны хосту, вы можете настроить код FPGA на ожидание положительного фронта.

Что делает код в этом примере:

A. Внутренний цикл управляет частотой дискретизации сигнала, а внешний цикл - синхронизацией каждой подпрограммы обработки с использованием прерываний.

B. Как только оператор вычисления среднеквадратичного выдаст действительный результат, внутренний цикл завершает работу.

C. Внешний цикл ждет положительный фронт импульса сканирования.

Приложение реального времени не требует модификации, потому что механизм сканирования уже запущен и используется для синхронизации управляющих циклов.

Предостережения при работе с определяемыми пользователем переменными ввода-вывода

- Вы можете реализовать определяемые пользователем переменные ввода-вывода только на целевых устройствах FPGA, работающих с механизмом сканирования. Обратитесь к документации на ваше целевое устройство для получения сведений о поддержке механизма сканирования.
- Определяемые пользователем переменные ввода-вывода не поддерживают симуляцию. Для изменения целевого объекта выполнения FPGA VI щелкните правой кнопкой мыши по элементу шасси, содержащему этот VI, и выберите **Execute VI on»FPGA Target** (Выполнять на»Целевое устройство FPGA).
- Вы можете использовать определяемые пользователем переменные ввода-вывода для обмена данными только между FPGA VI и VI реального времени, работающими на одном шасси. Однако, если для определяемых пользователем переменных ввода-вывода разрешена публикация в сети, вы может использовать эти переменные в любом VI реального времени или VI, работающем под Windows, в том же проекте LabVIEW. Например, вы можете использовать публикуемые в сети переменные ввода-вывода для создания VI пользовательского интерфейса, который выполняется в Windows.

Пример – сбор сигнальных данных с помощью буфера FIFO с каналом прямого доступа в память (DMA FIFO)

В данном разделе приведен пример кода LabVIEW

Конфигурирование коммуникаций между FPGA и аппаратными средствами реального времени

Вы можете использовать прямой доступ к памяти (DMA) для потоковой высокоскоростной передачи данных между FPGA и аппаратными средствами реального времени. Для потоковых данных буфер DMA можно создать, щелкнув правой кнопкой мыши по объекту FPGA и выбрав **New...»FIFO** (Новый»Буфер FIFO).

Назовите структуру FIFO подходящим именем (например, DataU32) и выберите тип "target-to-host" (от целевого устройства к хосту). Также вы можете задать тип данных и глубину FPGA FIFO. Данные автоматически передаются из этого FIFO в буфер данных, выделенный в RAM контроллера реального времени. Глубина буфера FPGA FIFO не так критична, как размер и коэффициент использования буфера данных реального времени.

Нажмите в диалоге кнопку OK, чтобы добавить в ваш проект новый буфер FIFO, который вы можете перетащить на блок-диаграмму FPGA.

Здесь вы собираете данные с четырех каналов и помещаете их в DMA FIFO вместо выполнения оперативной обработки данных. Хост-приложение отвечает за извлечение данных для обработки и/или регистрации. Основные принципы обработки данных в хосте аналогичны тем, что использовались в примере с оперативной обработкой в FPGA. В целом:

- Таймер цикла генерирует временную метку при первой итерации и задерживает выполнение второй итерации для достижения требуемой длительности цикла. Если время выполнения цикла в FPGA превышает заданное время, то для последующих итераций создается новая ссылка. Вы можете использовать сдвиговый регистр, помещенный после таймера цикла, чтобы измерить реальную длительность цикла. Проверка скорости выполнения цикла удостоверяет, что данные собираются с требуемой быстродействием.
- Проверка ошибок необходима для удостоверения в целостности данных. Если FPGA теряет связь с модулем C-серии, это сообщается через кластер ошибок узла ввода-вывода.
- Время выполнения узла ввода-вывода FPGA может занимать значительную часть интервала времени между отсчетами, особенно при высокой частоте дискретизации. Конвейер обеспечивает преобразование данных параллельно операции ввода-вывода FPGA, что минимизирует риск недозагрузки (кода модуль собирает данные медленнее, чем требуется).
- Поскольку данные передаются по конвейеру, в первой итерации цикла нет действительных данных. Внешняя структура case отбрасывает первый отсчет.

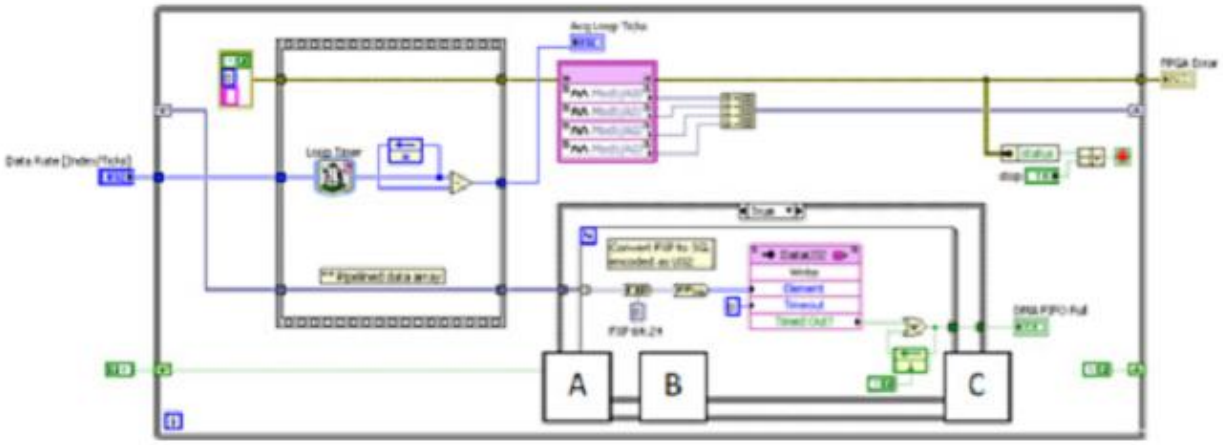


Рисунок 6.20. Помещение собранных данных в DMA FIFO

Что делает код примера:

A. В этом примере делается акцент на сборе данных из нескольких каналов. Цикл A (for loop) преобразует тип данных каждого отсчета и помещает каждый отсчет в DMA FIFO, что приводит к чередованию данных в FIFO.

B. Каждый отсчет данных должен быть переведен из типа данных с фиксированной точкой (FXP) к типу данных с плавающей точкой одинарной точности (SGL). Один из способов реализовать это – переслать данные типа FXP через DMA FIFO и преобразовать в тип SGL в приложении реального времени. Однако более эффективно выполнить это в FPGA и закодировать его типом U32. В приложениях реального времени приведение данных типа U32 в тип SGL выполняется почти на 40 процентов быстрее, чем преобразование из FXP в SGL. Эта простая техника сохраняет ценные ресурсы процессора хост-контроллера. Для получения дополнительной информации о преобразовании типов FXP-в-SGL в FPGA, а также для загрузки свежих материалов и документации, посетите сайт ni.com и ознакомьтесь с документом "FXP-to-Single Conversion on LabVIEW FPGA" (Преобразование FXP-в-Single в LabVIEW FPGA).

C. В этом примере также производится проверка на переполнение буфера DMA FIFO. Если узел ввода-вывода уходит в тайм-аут, то буфер DMA FIFO заполнен и данные были потеряны. Если это произойдет, защелкивается флаг ошибки, чтобы уведомить хост-приложение.

Недозагрузка модуля и поддержка различных режимов сбора данных

Вы можете расширить этот код для проверки на недозагрузку модуля и поддержки нескольких режимов сбора данных. Проверка на недозагрузку модуля выполняется так же, как было показано ранее в примере с оперативной обработкой данных. Снова вы задаете порог для указания, что модуль собирает данные медленнее, чем требуется, и даете FPGA пару итераций, чтобы очистить внутренние конвейеры и сдвиговые регистры, прежде чем сообщить о недозагрузке.

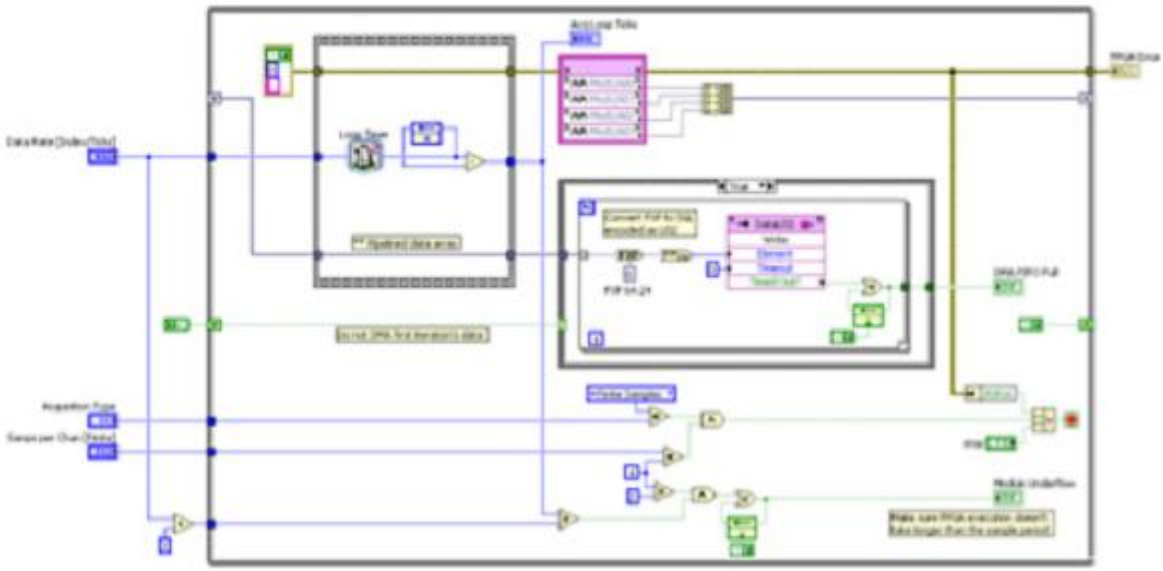


Рисунок 6.21. Режимы сбора данных и проверка на недозагрузку

В этом примере введены два новых управляющих элемента: Acquisition Type (Тип сбора) и Samples per Chan [Finite] (Отсчетов на канал [конечно]). Если выбран сбор конечного количества данных, цикл сбора данных автоматически завершается после прочтения заданного количества отсчетов на канал. Управляющий элемент "Тип сбора" дает приложению реального времени гибкость переключения между режимом непрерывного сбора данных и сбором конечного количества данных без необходимости перекомпиляции кода FPGA. Тип сбора данных также используется для определения того, какой объем памяти надо выделить буферу прямого доступа к памяти реального времени в хост-контроллере.

Синхронизация хоста и автоматический перезапуск

Синхронизация FPGA и хост-приложений особенно важна при выполнении непрерывных пересылок данных с прямым доступом в память. Если FPGA посылает данные до того, как приложение реального времени готово их принять, вы увеличиваете риск переполнения буфера DMA. Кроме того, приложение реального времени может уйти в тайм-аут, если начнет искать данные до того, как FPGA их пошлет. Вы можете использовать простое прерывание для синхронизации начала сбора данных в FPGA с готовностью приложения реального времени получать данные.

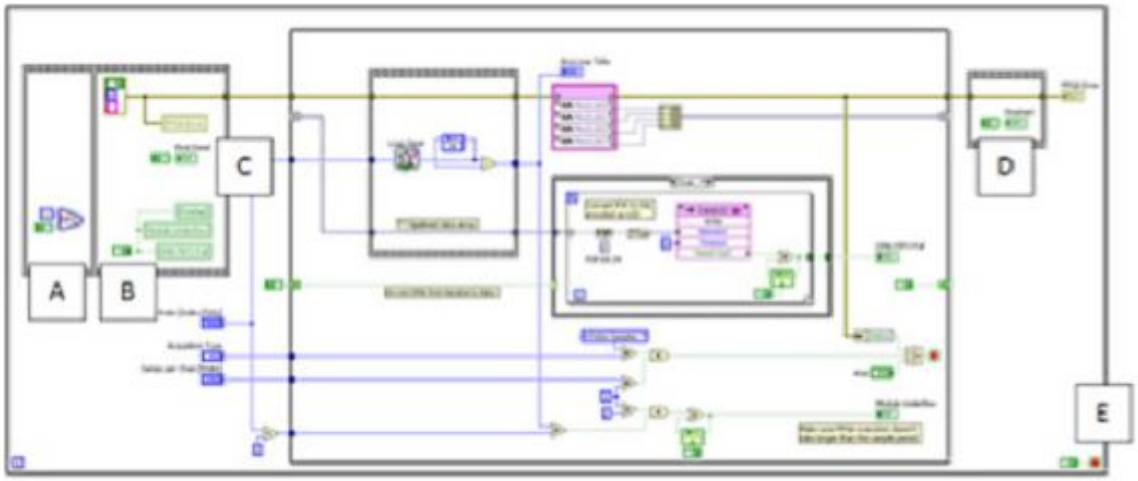


Рисунок 6.22. Сбор данных с прямым доступом в память синхронизирован с хостом

Что делает код в этом примере:

A. В начале приложения FPGA устанавливает прерывание 11 и ожидает подтверждения хост-приложения.

B. Как только хост подтверждает прерывание, инициализируются индикаторы ошибок, и начинается цикл сбора данных.

C. Индикатор "First Read" (Первое прочтение) сообщает приложению реального времени, что начался новый сбор данных. Этот флаг используется приложением реального времени для создания временных меток начала сбора в данных типа waveform LabVIEW

D. Сбор данных завершается по команде хост-приложения или при обнаружении ошибки, при этом включается индикатор "Finished" (Завершено) для подтверждения завершения цикла сбора данных.

E. Внешний цикл while перезапускает код FPGA для нового сбора данных. Он снова ждет подтверждения прерывания, прежде чем начать новый цикл сбора данных.

Встроенное масштабирование и согласование количества каналов

Когда данные помещаются в DMA FIFO, все каналы чередуются друг с другом. При чтении этих данных приложение реального времени должно расшифровать данные и надлежащим образом организовать их в двумерный массив. Если у вас 4 канала, то приложение реального времени должно преобразовать одномерный массив FIFO в двумерный массив с четырьмя строками. Соответственно, для приложения реального времени и FPGA VI очень важно согласовывать количество каналов, участвующих в сборе данных.

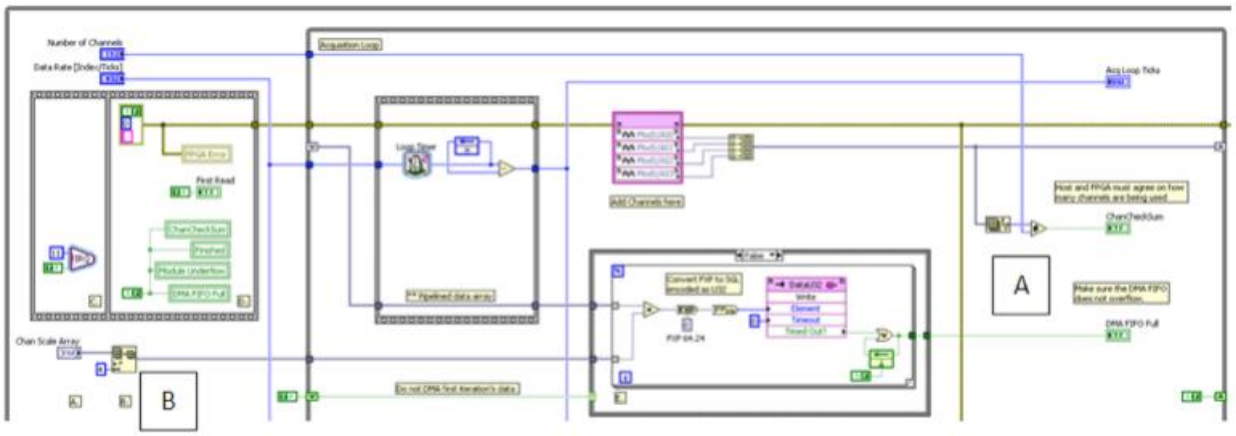


Рисунок 6.23. Проверка количества каналов и встроенное масштабирование

Что делает код в этом примере:

A. Проверка количества каналов реализована передачей ожидаемого количества каналов из приложения реального времени и сравнением с размером массива каналов в FPGA. Если количество каналов не совпадает, устанавливается флаг "ChanCheckSum".

B. В дополнение к проверке количества каналов, в примере также реализовано масштабирование для каждого отсчета данных. Вы можете масштабировать акселерометры, микрофоны и некоторые тензодатчики для получения верных инженерных единиц простым умножением напряжения на скалярное значение. "Chan Scale Array" получает одно скалярное значение на канал от приложения реального времени. Масштабирование реализуется умножением напряжения каждого канала на

соответствующий скаляр внутри цикла for. Теперь данные могут быть вернуться из FPGA в инженерных единицах вместо В или мВ. Более сложное масштабирование может быть добавлено в приложение при необходимости.

Чтение DMA FIFO в приложении реального времени

Приложение реального времени должно быть способно устанавливать режим и частоту взятия отсчетов, конфигурировать буфер DMA реального времени, подтверждать "стартовое" прерывание FPGA, опрашивать буферы DMA, читать из буферов DMA, восстанавливать чередующиеся данные и представлять их в готовой к использованию форме.

Вы можете реализовать все эти функции вручную с помощью VI интерфейса с FPGA. Однако существует набор VI, разработанных специально для сбора сигналов с прямым доступом в память. Эти VI поставляются вместе с руководством разработчика, но вы можете найти их последние версии и более подробную документацию на ni.com.

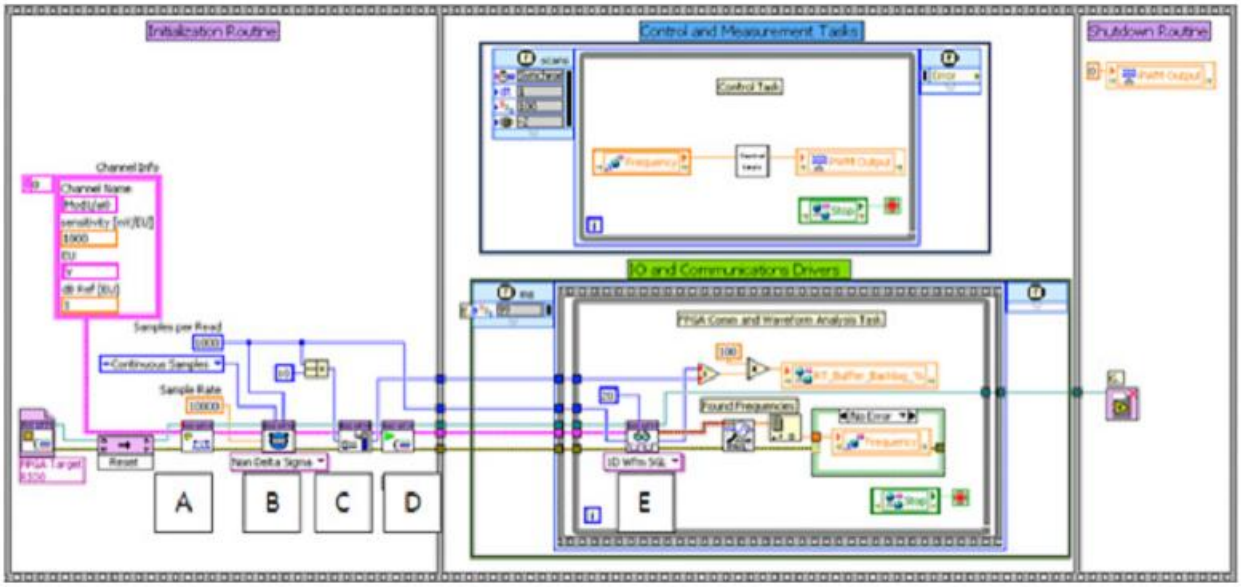


Рисунок 6.24. Использование VI CompactRIO сбора данных о сигналах

Функции для сбора данных о сигналах (CompactRIO Waveform VI) реализованы целиком в LabVIEW и использует те же VI интерфейса с FPGA, которые обсуждались в предыдущих разделах.

Ниже объясняется, что делает код в этом примере:

A. Этот VI собирает информацию о каналах, например, о масштабировании и именах каналов. Количество элементов в элементе управления Channel Info (Информация о канале) должно быть равно количеству каналов сбора данных в FPGA. Масштабные коэффициенты и количество каналов передаются в FPGA.

B. Timing VI устанавливает частоту и режим взятия отсчетов и передает информацию FPGA. Кроме того, если установлен режим дискретизации с конечным количеством отсчетов, то VI автоматически настраивает размер буфера прямого доступа к памяти реального времени равным точному количеству отсчетов. Поскольку в этом приложении используется непрерывный режим сбора данных, вам понадобится "вручную" контролировать буфер.

C. В случае непрерывного сбора данных размер буфера должен быть в несколько раз больше, чем заданный размер читаемого массива данных. Ели приложение реального времени не может прочитать значения из DMA FIFO достаточно быстро, буфер может переполниться. Чтобы этого избежать, вы должны уменьшить частоту взятия отсчетов, увеличить размер буфера для чтения или уменьшить объем операций обработки во время сбора в

приложении реального времени. Возможно, вам потребуется применить комбинацию этих факторов для выбора правильного решения в вашем приложении и для контроллера.

D. Start VI запускает откомпилированный битовый файл FPGA, ожидает прерываний от FPGA и подтверждает это прерывание для начала сбора данных.

E. Read VI опрашивает буфер DMA реального времени и читает из него отсчеты. Кроме того, он проверяет ошибочные состояния, заданные вами в FPGA (переполнение буфера, недозагрузка модуля, неверное число каналов, тайм-аут чтения и т.п.) Read VI гибок в представлении данных в приложении: он может возвращать данные как одномерный массив данных типа waveform, двухмерных массив данных типа SGL, одномерный массив данных типа U32. Напомним, значения U32 на самом деле - числа с плавающей точкой одинарной точности, закодированные в слова U32. Простое и дешевое приведение типов данных – все, что нужно для преобразования данных в нужную форму.

Приложение реального времени непрерывно читает данные четырех каналов, представляет их, как одномерный массив данных типа waveform, определяет самую высокую частоту сбора данных каждого канала и передает частоту канала 0 в табличную память. Вы можете использовать эти данные в цикле управления, выполняемом параллельно.

Сбор данных о сигналах модулями C-серии, использующих дельта-сигма АЦП

Не все модули аналогового ввода C-серии используют одну и ту же технологию аналого-цифрового преобразования. Некоторые из этих модулей используют АЦП с последовательным приближением (SAR), а некоторые – дельта-сигма АЦП. Предыдущие примеры FPGA демонстрировали, как собирать данные из модулей с АЦП типа SAR или из модулей с АЦП, не использующих дельта-сигма преобразование. Если вы используете модули NI 9234, NI 9237 или другие модули C-серии с АЦП, основанные на дельта-сигма преобразовании, вы должны рассмотреть особые требования к FPGA.

Основным отличием между двумя типами АЦП устройств является задание времени цикла сбора данных в FPGA. Для модуля с АЦП типа SAR таймер цикла управляет временем выполнения цикла, что дает хост-приложению требуемую частоту сбора данных. Состояния недозагрузки должны быть проверены вручную.

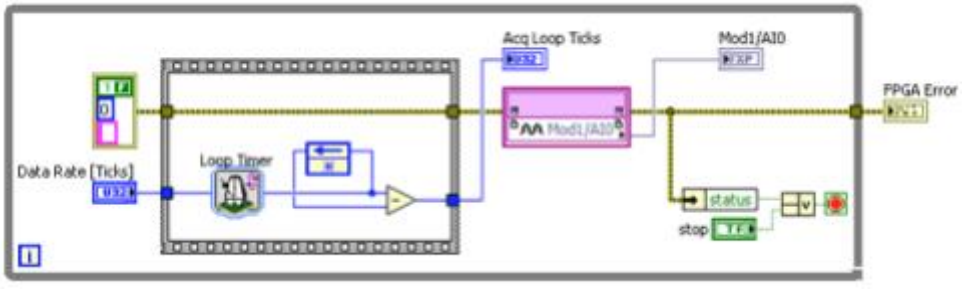


Рисунок 6.25. Задание времени выполнения цикла сбора данных в FPGA с модулем C-серии на основе АЦП типа SAR

На следующем рисунке показан сбор данных из модуля C-серии с дельта-сигма АЦП. Обратите внимание, что задание частоты отсчетов и синхронизации выполняются вне цикла сбора данных. В этом случае узел ввода-вывода FPGA сам управляет временем выполнения цикла, используя значение из элемента перечислительного типа Data Rate (Скорость данных). При вызове узла ввода-вывода FPGA выполнение ожидает, пока с модуля не будет поступит действительный отсчет. Если любая итерация цикла занимает больше времени, чем заданный интервал времени между отсчетами, это означает, что имеет место недозагрузка, и узел ввода-вывода возвращает предупреждение 65539. Опять же, при добавлении оперативной обработки данных или DMA FIFO, полезно использовать конвейер, чтобы дать узлу ввода-вывода FPGA столько времени, сколько нужно для извлечения данных из модуля.

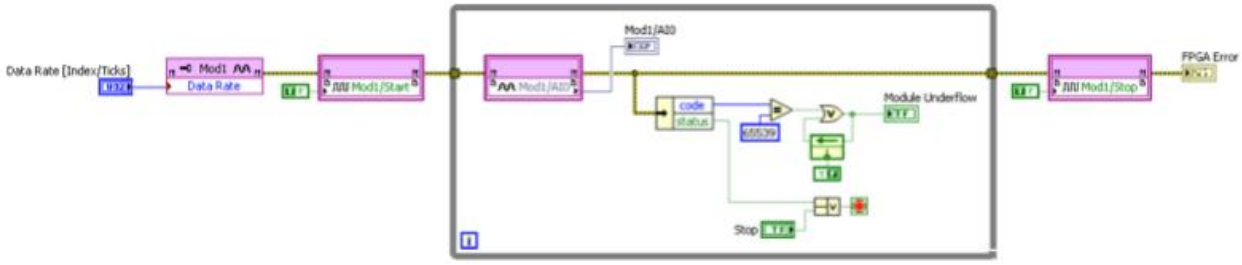


Рисунок 6.26. Задание времени выполнения цикла сбора данных в FPGA с модулем C-серии на основе сигма-дельта АЦП

FPGA VI должен также включать конвейер, проверку на переполнение, DMA FIFO и преобразование данных. Полнофункциональный пример с дельта-сигма АЦП включен в документ, опубликованный в зоне разработчика (Developer Zone), упоминавшийся в предыдущих разделах (Reference Applications for CompactRIO Waveform Acquisition)

Модули C-серии без поддержки режима сканирования

Большинство модулей C-серии поддерживают режим сканирования, но некоторые специализированные модули, например приводов двигателей, CAN, последовательных портов, модули SD-карт – нет. Кроме того, CompactRIO имеет открытую архитектуру, так что покупатели и сторонние фирмы могут создавать собственные модули C-серии. Для использования этих модулей в системе CompactRIO, убедитесь, что они находятся в разделе FPGA в проекте, чтобы вы могли задать для них гибридный режим. Поскольку это не типовые аналоговые или цифровые модули, для них нет общего API для FPGA, но для модулей NI в Поисковике примеров (NI Example Finder) можно найти примеры, включающие код LabVIEW FPGA и хост-код LabVIEW Real-Time. Большинство модулей сторонних фирм также поставляются с примерами.

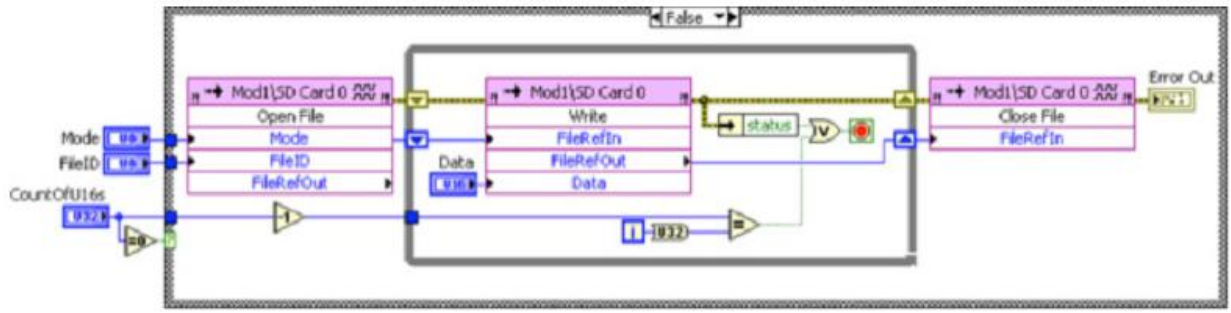


Рисунок 6.27. Пример записи SD-карты для NI 9802

Практический опыт разработки LabVIEW FPGA

В этом разделе рассматриваются полезные советы и приемы, чтобы сократить время, затрачиваемое на разработку высокопроизводительных систем управления с модулем LabVIEW FPGA и CompactRIO. Здесь рассматриваются технологии отладки, такие, как симуляция, а также несколько рекомендуемых приемов программирования, позволяющих избежать распространенных ошибок и множество методов для создания быстрых, эффективных и надежных приложений LabVIEW FPGA. Для получения наилучших результатов от этих советов и методов проектирования вы должны иметь базовые знания о программировании в LabVIEW FPGA.

В этом разделе приведены примеры по созданию высокопроизводительной системы управления щеточным двигателем постоянного тока. Здесь вы ознакомитесь с различными способами программирования, используемыми для создания subVI LabVIEW FPGA генерации ШИМ-сигналов для привода двигателя, декодирования цифровых импульсов от квадратурного энкодера и ПИД-регулирования в контуре с обратной связью по положению двигателя. В результате будет спроектирована высокопроизводительная система управления с субнаносекундным джиттером,

содержащая несколько циклов обработки данных с частотой 40 МГц, на реализацию которых будет израсходовано только 17 процентов из 3M вентилей FPGA.

Рассмотрим пять ключевых методов разработки надежных высокопроизводительных приложений LabVIEW FPGA.

Метод 1. Использование синхронизируемых циклов, выполняющихся за один такт (SCTL)

Первый метод разработки состоит в использовании из модуля LabVIEW FPGA синхронизируемых циклов, выполняющихся за один такт (Single-Cycle Timed Loop - SCTL). SCTL указывает компилятору LabVIEW FPGA оптимизировать расположенный внутри цикла код, добавляя специальное временное ограничение, требующее, чтобы код исполнялся за такт генератора FPGA. Код, компилируемый в SCTL, лучше оптимизирован и занимает меньше пространства в FPGA по сравнению в тем же кодом в обычном цикле while. Кроме того, код внутри SCTL выполняется невероятно быстро. При установленной по умолчанию частоте тактового генератора 40 МГц один цикл выполняется за 25 нс.

На рисунке 6.28 показаны два идентичных приложения LabVIEW FPGA. В левом используются обычные циклы while, а в правом - SCTLs в своих subVI. Этот пример иллюстрирует эффективность параллельной обработки. Верхний цикл читает и обрабатывает цифровые сигналы от квадратурного энкодера двигателя, а нижний – выполняет ШИМ для управления мощностью, посылаемой двигателю. Это приложение написано для модуля привода двигателя NI 9505, который управляет щеточными двигателями. Этот код выполняется экстремально быстро – приложение справа выполняет два разных цикла на частоте 40 МГц.

Циклы While

Количество SLICEs: 3245 из14336 22%

Циклы SCTL

Количество SLICEs: 2456 из 14336 17%

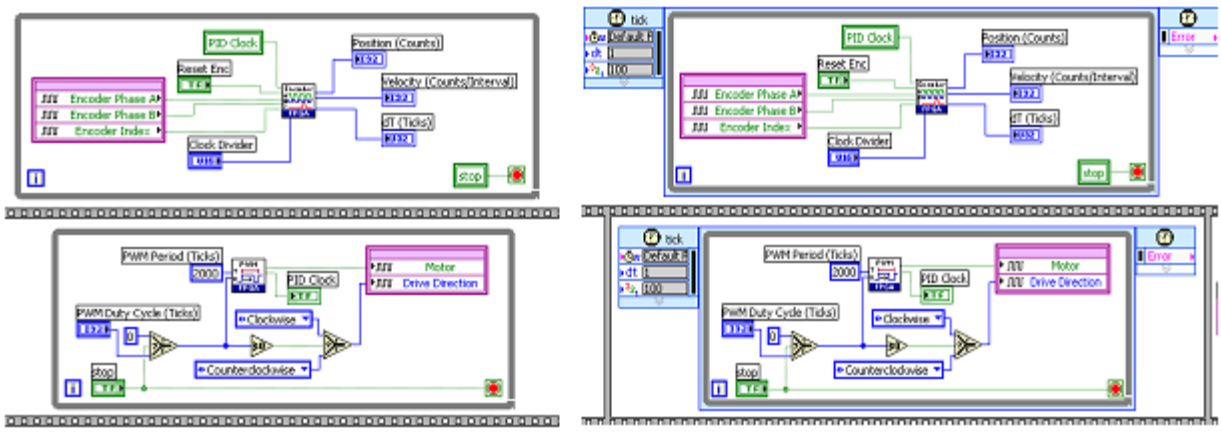


Рисунок 6.28. Эффективность параллельной обработки

На рисунке 6.28 показаны и отчета о результатах компиляции. Приложение, построенное с использованием SCTLs, использует меньше секций FPGA (SLICE), однако на его компиляцию уходит больше времени, потому что компилятор должен выполнить больше операций для удовлетворения ограничений по длительности выполнения цикла SCTL.

Теперь рассмотрим подробно, как работает SCTL.

Когда компилируется код обычного цикла while, LabVIEW FPGA добавляет триггеры для тактирования передачи данных от одной функции к другой, принудительно создавая таким

образом характерный для LabVIEW синхронный поток данных и предотвращая гонки. Триггеры помечены на рисунке 6.29 как прямоугольники с надписью FF на выходе каждой функции.

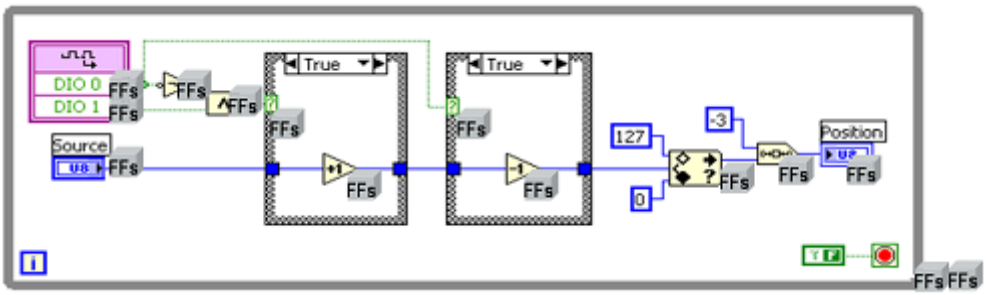


Рисунок 6.29. Когда компилируется код обычного цикла while, LabVIEW FPGA добавляет триггеры (обозначенные, как FF) для тактирования передачи данных от одной функции к другой

На рисунке 6.30 показан тот же код, компилируемый для цикла SCTL. Только входы и выходы цикла содержат триггеры. Внутренний код реализуется более параллельно, а при оптимизации кода между входами и выходами цикла существенно уменьшается размер логической схемы.

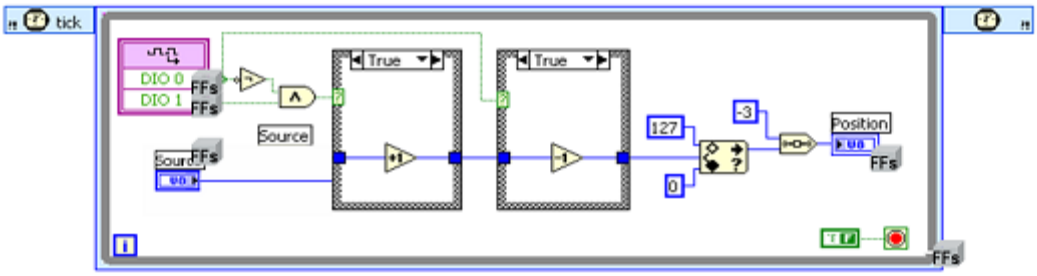


Рисунок 6.30. Код из рисунка 6.19, скомпилированный для цикла SCTL

Как вы видите, SCTL – простой способ оптимизировать код LabVIEW FPGA, однако существуют некоторые ограничения использования этих циклов (таблица 6.1).

Элементы, неразрешенные в SCTL	Предлагаемая альтернатива
Длинные последовательности кода	Сделайте код более параллельным. Добавляйте узлы обратной связи для конвейеризации
Частное и остаток (Quotient and remainder)	Используйте функцию Scale by Power of 2 для выполнения целочисленного деления, или используйте библиотеку математических функций с фиксированной точкой
Таймер цикла, функции задержки (Loop timer, wait functions)	Используйте функцию Tick Count для запуска событий.
Аналоговый ввод, аналоговый вывод	Поместите их в отдельный цикл While и используйте локальные переменные для передачи данных
Циклы While	Для вложенных subVI используйте узлы обратной связи для сохранения состояний

Таблица 6.1. Ограничения синхронизируемых циклов, выполняемых за один такт

Для использования SCTL все операции внутри цикла должны выполняться за один период импульсов тактового генератора FPGA. В начале процесса компиляции генератор кода передает сообщение об ошибке, если SCTL не может сгенерировать надлежащий код для компилятора. Это означает, что длинные последовательности кода могут не пройти в SCTL. Последовательный код – это код, в котором результаты одного вычисления требуются в следующей операции, не допуская параллельное выполнение вычислений. Чтобы это исправить, вы можете переписать код так, чтобы сделать его более параллельным. Например, вы можете добавить узел обратной связи (Feedback Node) для передачи результатов от одного вычисления другому в следующей итерации цикла – это известно как конвейерная обработка. Вы можете использовать конвейерную обработку для уменьшения длительности каждого запуска SCTL, разделив код между несколькими итерациями цикла.

Вы не можете использовать функцию **Quotient and Remainder** (Частное и остаток) в SCTL. Если вам нужно разделить целые числа, вы можете использовать функцию **Scale by Power of 2** (Отмасштабировать степенью 2). С помощью этой функции вы можете умножать или делить на степень двойки, то есть на 2, 4, 8, 16, 32 и т.д. Для значений с фиксированной точкой вы можете использовать библиотеку Fixed-Point Math Library из LabVIEW FPGA.

На рисунке 6.31 показан subVI, выполняющий деление с фиксированной точкой, и панель конфигурирования, включающая элемент управления Execution Mode (Режим выполнения), который позволяет использовать эту функцию в SCTL.

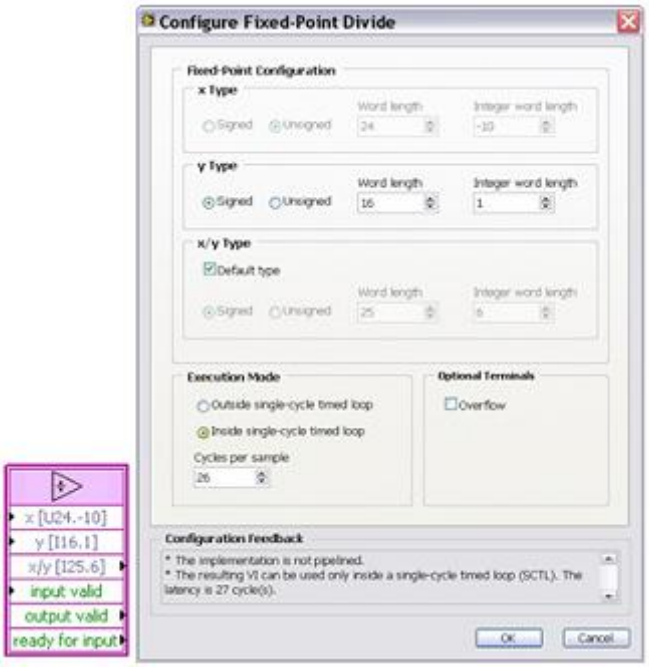


Рисунок 6.31. SubVI деления с фиксированной точкой и панель конфигурирования

Библиотека Fixed-Point Math Library содержит IP-блоки LabVIEW FPGA, реализующие различные элементарные и трансцендентные математические функции. Эти функции используют тип данных с фиксированной точкой, появившийся в LabVIEW 8.5, что расширяет возможности выполнения операций деления, вычисления синуса и косинуса. Все функции проверены и могут использоваться и внутри, и вне цикла SCTL, а также при симуляции в Windows и FPGA. Библиотека снабжена справочной информацией по каждой функции. Вы можете свободно загрузить ее с сайта ni.com.

Если вы пытаетесь создать subVI для реализации внутри SCTL, вы можете использовать узел обратной связи для сохранения информации о состоянии subVI. Это исключает необходимость использования цикла while внутри SCTL. В примере LabVIEW FPGA на рисунке 6.32 рассчитывается одно из дифференциальных уравнений для двигателя постоянного тока с использованием функций из библиотеки Fixed-Point Math Library. После каждой математической операции с фиксированной

точкой используется узел обратной связи для конвейерной обработки результата и, соответственно, передачи значений от одной итерации другой. Функция **Tick Count** (в верхнем правом углу) с узлом обратной связи служит для расчета скорости цикла выполнения subVI.

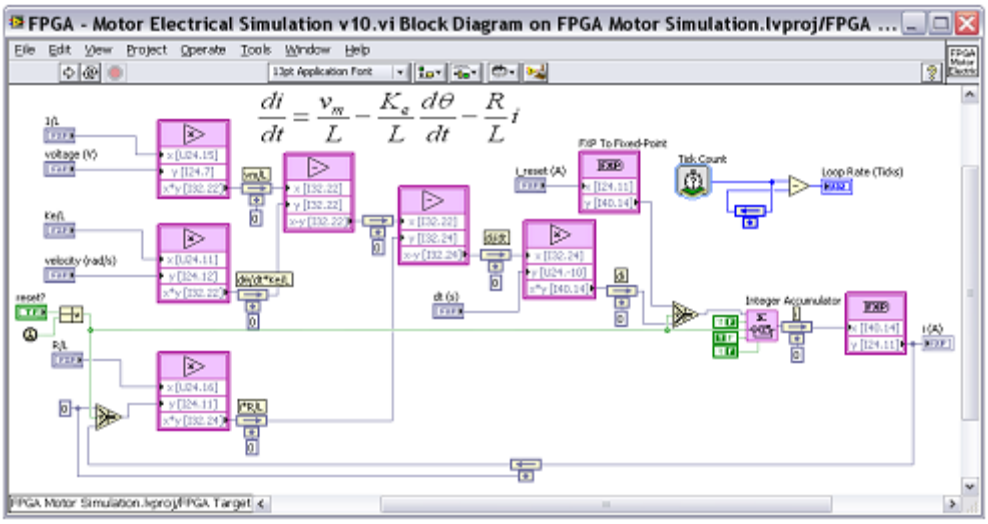


Рисунок 6.32. Расчет дифференциального уравнения для двигателя постоянного тока с использованием функций библиотеки Fixed-Point Math Library

На рисунке 6.33 вы видите SCTL верхнего уровня в приложении FPGA, который вызывает subVI симуляции двигателя. Поскольку этот subVI вложен в цикл SCTL, значение времени выполнения цикла (**Loop Rate (Ticks)**) всегда равно единице. Однако конвейерная обработка вызывает задержку в шесть импульсов тактового генератора от входа **напряжения** (V) до выхода **по току** i (A) subVI.

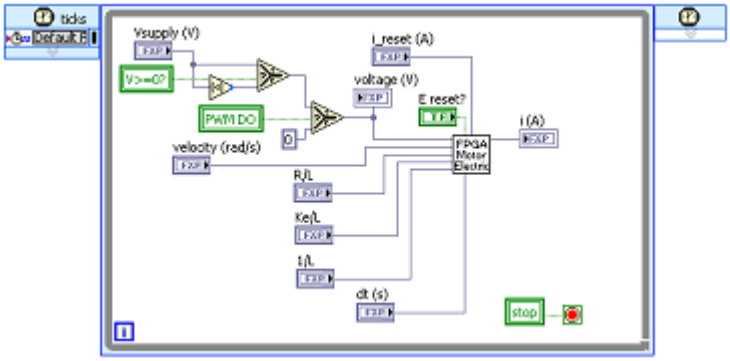


Рисунок 6.33. Конвейерная обработка вызывает задержку в шесть импульсов тактового генератора от входа напряжения (V) до выхода по току i (A) subVI

Помимо конвейерной обработки вы можете использовать в SCTL конечный автомат состояний для лучшей организации кода и выполнения последовательности шагов. Базовый компонент конечный автомат - структура case, в которой каждый кадр содержит одно состояние, а сдвиговый регистр используется для определения следующего состояния после каждой итерации цикла. Каждое состояние должно выполняться за один период тактового генератора, если subVI помещен внутрь SCTL. Кроме того, вы может использовать сдвиговые регистры и значения счетчика для реализации функциональности **цикла for** или добавления заданного количества **состояний ожидания** при выполнении программы.

Примечание: Добавление таймера цикла (Loop timer) или функции задержки (Wait) заставляет код выполняться медленнее, чем за период импульсов тактового генератора, поэтому их нельзя использовать в SCTL. Функции аналогового ввода и вывода также выполняются дольше одного периода тактового генератора и не могут быть использованы в SCTL. Однако вы можете поместить их в обычный цикл while и использовать локальные переменные для передачи данных в SCTL.

Совет: Создание счетчиков и таймеров

Если вам нужно запустить событие после некоего периода времени, используйте функцию Tick Count для измерения длительности прошедшего времени, как показано на рисунке 6.34. Не используйте терминал итерации, встроенный в цикл while и SCTL, потому что в конце концов он достигает своего предельного значения. Это происходит после 2,147,483,647 итераций цикла. На частоте 40 МГц это занимает 53.687 секунд. Вместо этого создайте собственный счетчик, используя беззнаковое целое и узел обратной связи, а также функцию Tick Count для получения значений времени по тактовому генератору FPGA частотой 40 МГц.

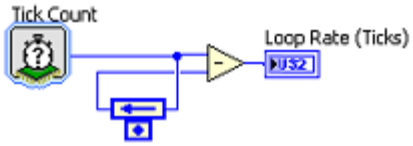


Рисунок 6.34. Используйте функцию Tick Count для измерения прошедшего времени

Поскольку вы используете беззнаковое целое для значений счетчика, вычисления прошедшего времени останутся верными, даже когда счетчик переполнится. Это происходит потому, что если вы вычтите одно значение счетчика из второго, используя беззнаковые целые числа, вы все равно получите верный ответ даже при переполнении счетчика.

Еще один распространенный тип счетчика – счетчик итераций, считающий количество выполнений цикла. Для этого обычно также используются беззнаковые целые числа, потому что они обеспечивают самый большой диапазон до переполнения. Беззнаковый 64-разрядный целый тип данных обеспечивает диапазон счета порядка $18*10^{18}$.

Даже если тактовый генератор FPGA работает с частотой 40 МГц, счетчик не переполнится еще 14 000 лет.

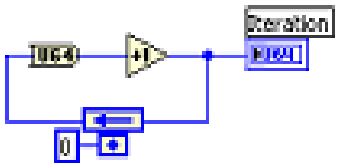


Рисунок 6.35. Для счетчиков итераций обычно используются беззнаковые целые числа, потому что они обеспечивают самый большой диапазон до переполнения

Теперь изучим другой метод создания хорошо написанного и эффективного кода LabVIEW FPGA.

Метод 2. Разработка кода FPGA как модульных, повторно используемых subVI

Следующий основной метод проектирования – модульная разработка, заключается в разбиении приложения на независимые функции, которые можно определять, разрабатывать и тестировать по отдельности. Это кажется достаточно простой идеей, но при разработке FPGA она может давать большие преимущества. Например, представьте функцию измерения частоты цикла, в который она помещена, и подсчета количества ее выполнений. Внутри цикла - функция Tick Count, которая считывает текущее значение генератора FPGA и вычитает его из предыдущего, которое хранится в

сдвиговом регистре. Кроме того, у вас есть 64-разрядный счетчик, который увеличивается на единицу каждый раз при вызове функции. Эта функция использует SCTL, так что на ее выполнение требуется всего один тик в 25 нс. Таким образом, этот subVI разработан для помещения в обычный цикл while и не оказывает влияния на скорость его выполнения.

Лицевая панель показана на рисунке 6.36. Индикаторы назначены двум правым терминам subVI, чтобы данные могли передаваться в приложение LabVIEW FPGA верхнего уровня, куда subVI помещается.

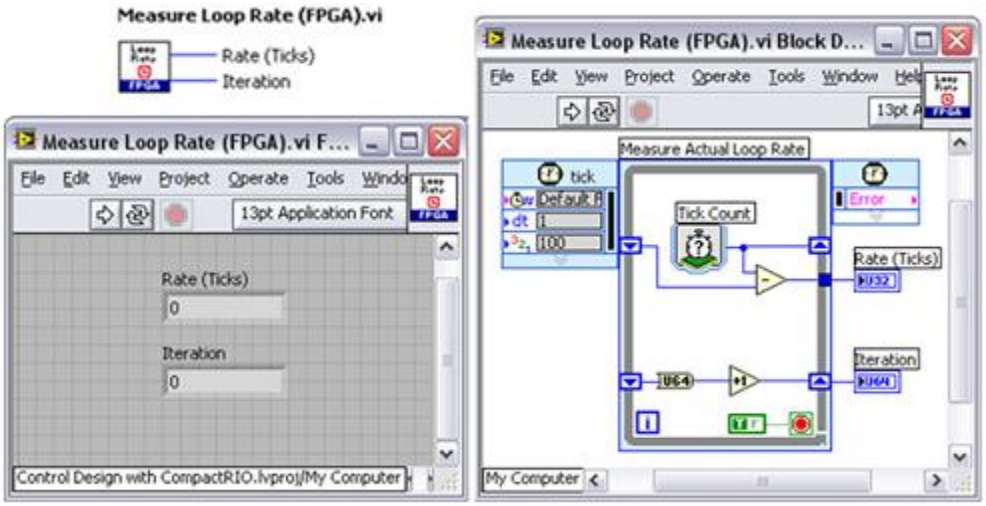


Рисунок 6.36. Лицевая панель subVI, разработанного для использования в обычном цикле While и не влияет на скорость его выполнения.

Функция используется в примере, приведенном на рисунке 6.27. SubVI помещается внутрь другого цикла для измерения скорости выполнения кода верхнего уровня.

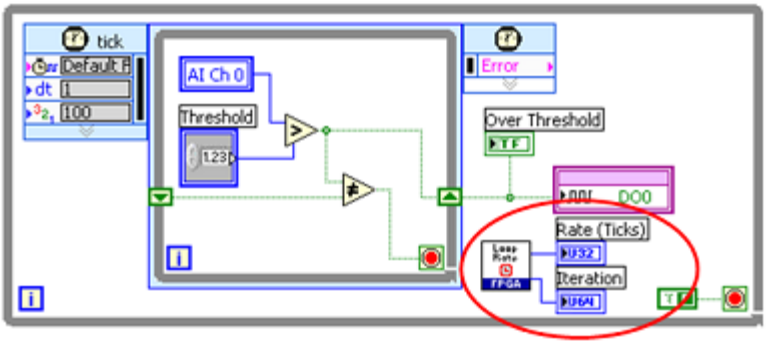


Рисунок 6.37. SubVI помещается внутрь другого цикла для измерения скорости выполнения кода верхнего уровня

Сводка некоторых преимуществ подобного проектирования кода приведена в таблице 6.2.

Преимущество	**Объяснение**
Проще отладка и нахождение проблем	Вы можете тестировать код в Windows до компиляции
Проще документировать и отслеживать изменения	Вы можете включать справку в документацию VI
Создает более ясную и простую для понимая блок-диаграмму верхнего уровня	Код интуитивно более понятен другим программистам

Таблица 6.2. Преимущества разработки кода FPGA в виде модульных, повторно используемых subVI

Написание модульного кода - почти всегда хорошая идея, но когда вы разрабатываете логику FPGA, она дает дополнительные преимущества.

Во-первых, код проще отлаживать и находить ошибки. Большое преимущество в том, что вы можете протестировать subVI в Windows, прежде чем компилировать в FPGA. Примеры приведены далее в этом разделе.

Во-вторых, это упрощает документирование и отслеживание изменений, потому что код модульный, и вы можете включать справочную информацию в документацию VI .

В-третьих, функциональные возможности кода, как правило, яснее прослеживаются, код проще для понимания и пригоден к повторному использованию. Опции, которые вы хотите предложить программисту, обычно доступны через терминалы subVI. Чаще всего пользователям не нужно изменять код нижнего уровня, они могут только использовать предоставленные вами параметры, например, как в этом примере ШИМ (**Pulse Width Modulation (FPGA).vi**).

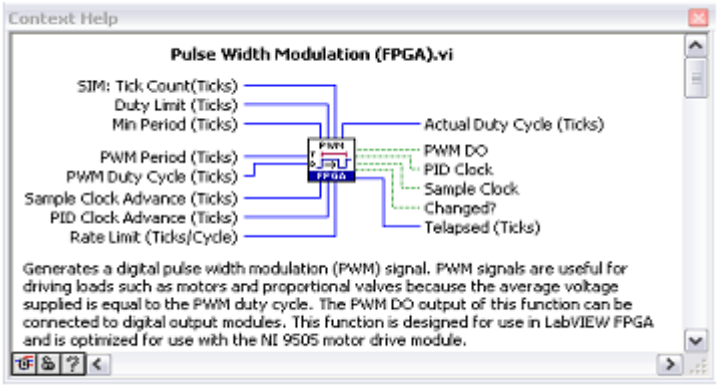


Рисунок 6.38. Программисты могут использовать параметры, предоставленные вами в VI

Теперь приведем несколько советов по созданию модульных повторно используемых subVI в LabVIEW FPGA.

Совет: Отделяйте логику от ввода-вывода.

Первый совет – размещать узлы ввода-вывода вне subVI. Это делает их более модульными и компактными, а диаграмму верхнего уровня - более легкой для чтения. Особенно удобно для управляющих приложений, чтобы все операции ввода-вывода были четко видимы при просмотре диаграммы верхнего уровня, как показано в цикле ШИМ, разработанном для модуля NI 9505 привода двигателя (рисунок 6.39).

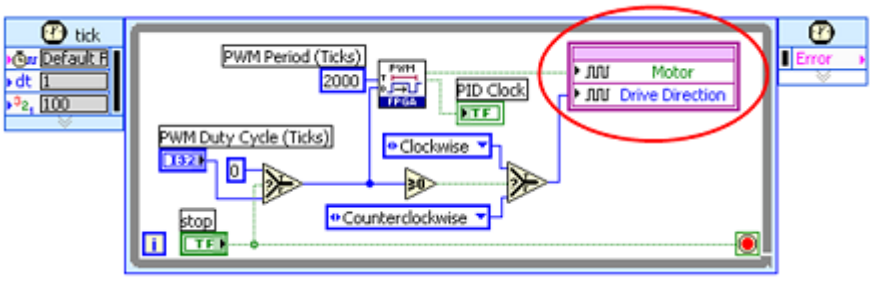


Рисунок 6.39. Размещайте узлы ввода-вывода вне subVI и убедитесь, что они ясно видны в блок-диаграмме верхнего уровня вашего приложения

Вместо встраивания узла ввода-вывода в subVI используется терминал для передачи данных от subVI диаграмме верхнего уровня. Это делает код FPGA проще в отладке, потому что вы можете тестировать subVI отдельно в Windows, симулируя ввод-вывод.

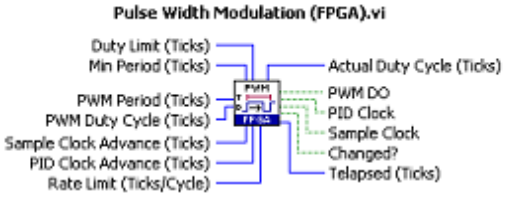


Рисунок 6.40. Вместо встраивания узла ввода-вывода в subVI, используется терминал для передачи данных от subVI диаграмме верхнего уровня.

Использование такого подхода уменьшает также внешние копии узлов ввода-вывода, которые иначе могут быть несколько раз включены в subVI, вызывая ненужное использование вентилей, потому что компилятору приходится добавлять дополнительную арбитражную логику для обработки множества запросов, требующих доступ к одному и тому же ресурсу.

Наконец, такой подход делает диаграмму верхнего уровня более читаемой – все операции чтения и записи ввода-вывода явно показаны на диаграмме и не скрыты.

Часто, когда вы так проектируете функциональные блоки, subVI требуется некоторый объем локальной памяти для хранения значений состояний, например, прошедшего времени, и передавать эти значения между итерациями.

Совет: Храните значения состояния в функциональном блоке

На рисунке 6.41 показано, как вы можете добавлять в цикл сдвиговые регистры для передачи информации между итерациями. Счетчик итераций увеличивается каждый раз при вызове функционального блока.

Обратите внимание, что к терминалу **Loop Condition** (Условие цикла) подключена константа, что заставляет цикл выполняться только один раз при вызове. В этом случае вам на самом деле не нужен цикл - вы просто используете SCTL для оптимизации кода и хранения значений состояний в сдвиговых регистрах.

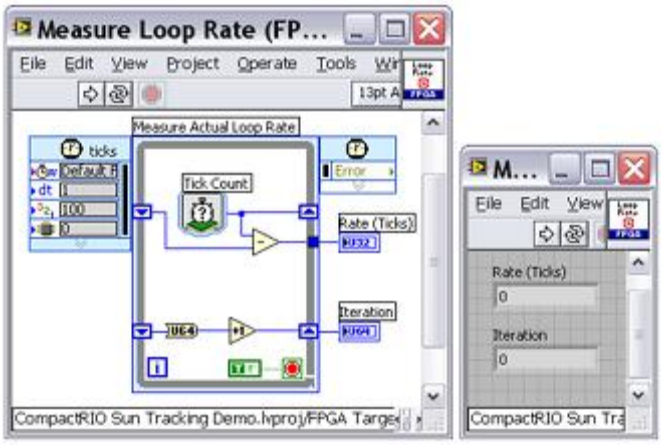


Рисунок 6.41. Добавьте в цикл сдвиговые регистры для передачи информации между итерациями цикла

Примечание: Сдвиговый регистр должен быть неинициализирован, чтобы subVI хранил состояния. При первом вызове значение сдвигового регистра должно принимать значение по умолчанию для соответствующего типа данных – 0 для целых чисел, false – для логического типа. Если вам нужно инициализировать сдвиговый регистр другим значением, используйте функции First Call? (Первый вызов?) и **Select** (Выбор).

Вы можете задуматься, как создать модульный функциональный блок, который работает в SCTL, потому что нельзя помещать один SCTL в другой.

Вы можете использовать узлы обратной связи для достижения этой цели, как показано на рисунке 6.42. Главное преимущество подобного подхода – вы можете легко инициализировать узлы

обратной связи и поместить subVI в цикл SCTL верхнего уровня, потому что он не содержит вложенных циклов.

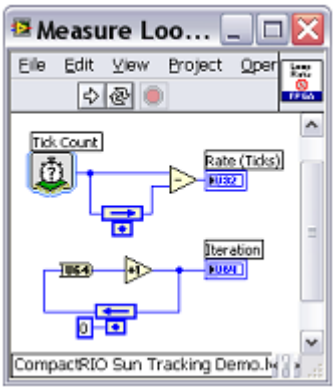


Рисунок 6.42. Вы можете использовать узлы обратной связи для создания модульного функционального блока, который работает внутри цикла SCTL.

Третий вариант – использование памяти VI-scoped memory. Это блок памяти, который subVI может использовать локально, и который не должен вручную добавляться в проект. Это делает код более модульным и компактным при перемещении между проектами.

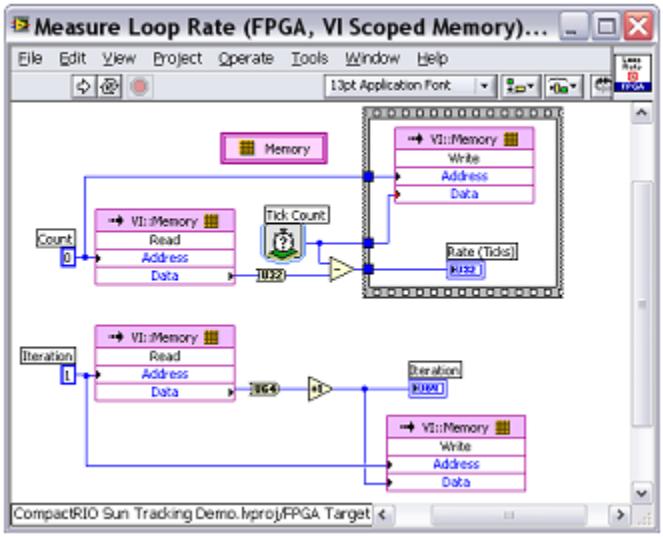


Рисунок 6.43. Используйте subVI с памятью VI-scoped memory, чтобы сделать код более модульным и компактным

В этом простом примере использование памяти VI-scoped memory – скорее всего, излишество для приложения. У вас память нужна только в двух местах, и вы храните только одно значение данных в каждом из них. Однако, память VI-scoped memory - мощный инструмент для приложений, которые должны хранить массивы данных. Вообще вы должны всегда избегать использования больших массивов на лицевой панели в качестве механизма хранения данных – используйте вместо этого VI-scoped memory.

Оперативно обновляемая таблица преобразования (LUT)

Обычно локальная память в управляющих FPGA –приложениях используется для хранения табличных данных, например, таблицы калибровки нелинейного датчика, заранее рассчитанной математической формулы (логарифмической или экспоненциальной) или значений сигнала произвольной формы, которую вы можете воспроизвести, индексируя адреса таблицы. На рисунке 6.44 показана таблица преобразования (lookup table FPGA), сконфигурированная на хранение 10 000 значений с фиксированной точкой и выполнение линейной интерполяции хранимых значений. Поскольку используется память VI-scoped memory, вы можете изменить значения в таблице преобразования в ходе выполнения приложения без необходимости перекомпилировать FPGA.

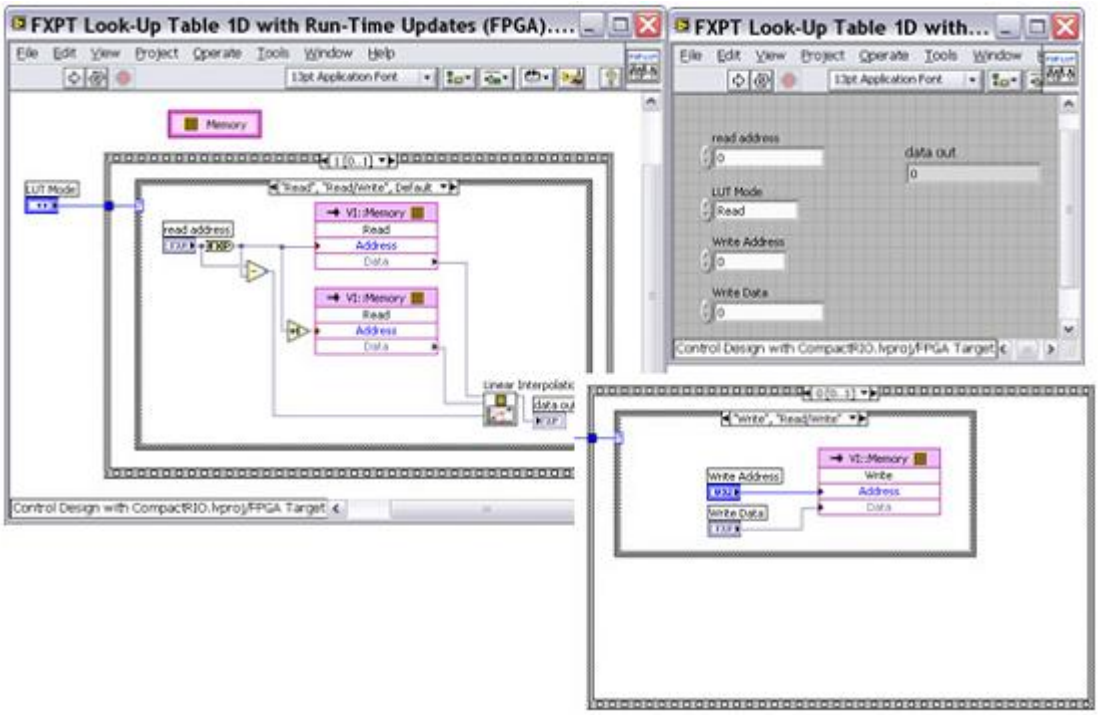


Рисунок 6.44. Таблица преобразования (lookup table), настроенная на хранение 10 000 значений с фиксированной точкой и выполнение линейной интерполяции хранимых значений

Теперь ознакомьтесь со страницами конфигурирования блока памяти VI-scoped memory в этом примере. Вы можете настроить глубину и тип данных, а также определить начальные значения для элементов памяти.

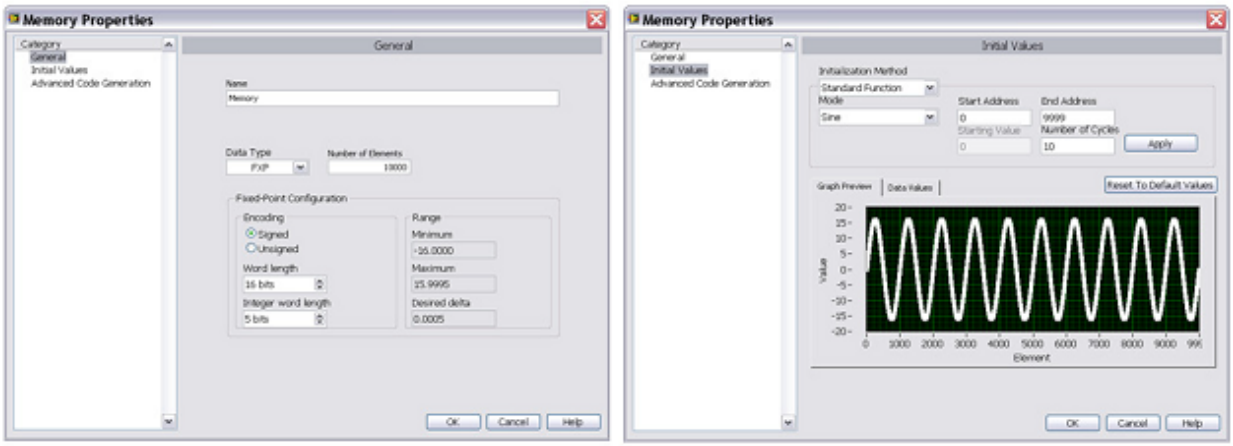


Рисунок 6.45. Страницы конфигурирования блока памяти VI-scoped memory в этом примере

Рассмотрим еще один совет для создания модульных subVI FPGA, имеющий отношение к синхронизации выполняемого кода.

Совет: Не помещайте таймеры задержки в subVI

Вообще рекомендуется избегать функций таймера цикла (Loop timer) или задержки (wait) в модульных subVI. Если в subVI нет задержек, он выполняется на максимальной скорости, как можно быстрее, таким образом, сохраняя временные свойства вызывающего VI, не замедляя при этом его выполнение. И вы можете проще адаптировать код в SCTL, если в нем не будет функций, приводящих к задержкам.

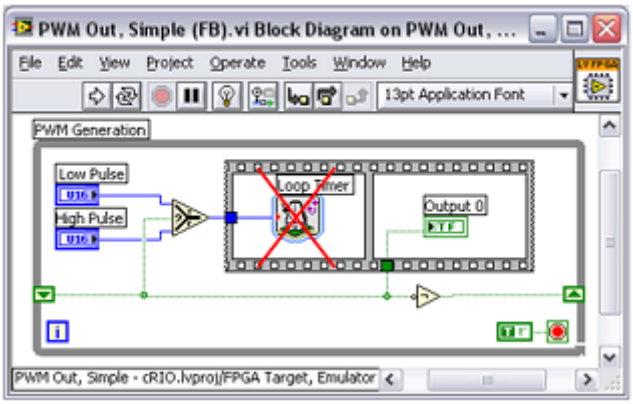


Рисунок 6.46. Избегайте использования таймеров цикла или задержек в модульных subVI

Слева на рисунке 6.47 показано, как вы можете адаптировать ШИМ-код для использования функции Tick Count вместо функции Loop Timer.

Используя узел обратной связи для запоминания значения прошедшего времени, вы включаете и выключаете выход в нужное время и сбрасываете счетчик прошедшего времени в конце цикла ШИМ. Код может выглядеть чуть более сложным, но вы можете поместить его в цикл верхнего уровня, не затрагивая общее время цикла - этот код более компактный.

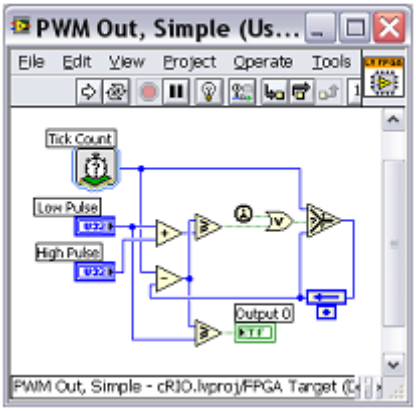


Рисунок 6.47. Адаптация ШИМ-кода для использования функции Tick Count вместо функции Loop Timer

Прежде чем перейти к следующей теме, изучите еще один совет,– проектировать код так, чтобы можно было помещать несколько копий subVI в одно приложение, и каждая копия была независима друг от друга.

Совет: Пользуйтесь преимуществами реентерабельности

Реентерабельность – настройка в свойствах выполнения subVI, позволяющая запускать несколько копий функционального блока параллельно с разными и отдельными хранилищами данных.

Рисунок 6.48. Используйте реентерабельное выполнение, чтобы сделать несколько копий функционального блока, выполняющихся параллельно с разными и отдельными хранилищами данных

На рисунке 6.49 приведен пример. В этом случае ваш subVI настроен на реентерабельное выполнение, так что все четыре цикла выполняются одновременно, и любые внутренние сдвиговые регистры, локальные переменные или память VI-scoped memory уникальны для каждой копии.

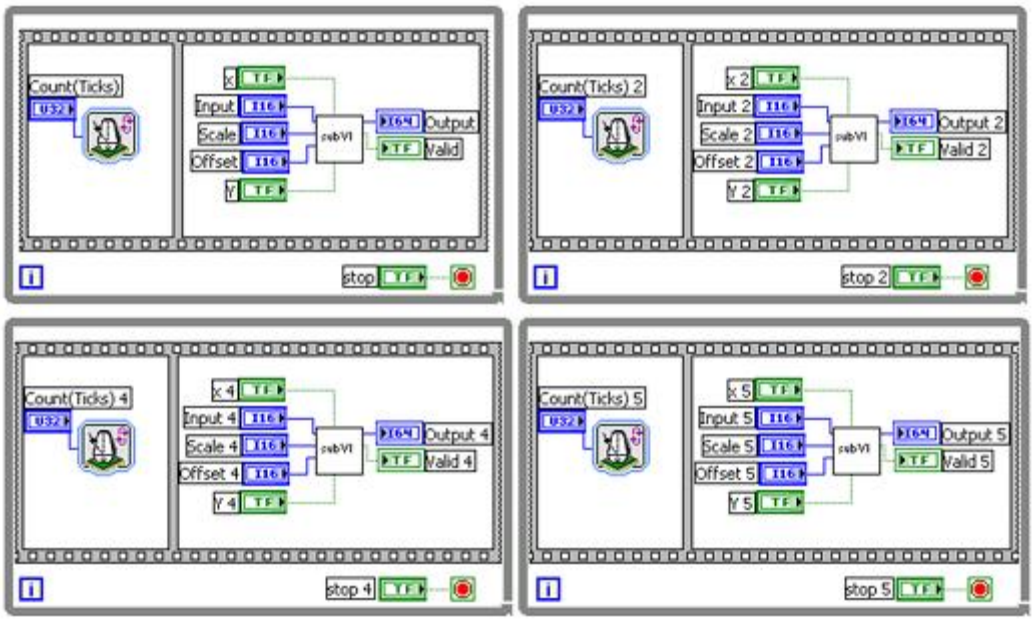


Рисунок 6.49. SubVI настроен на реентерабельное выполнение, так что все четыре цикла выполняются одновременно, и любые внутренние сдвиговые регистры, локальные переменные или память VI-scoped memory уникальны для каждой копии

В случае LabVIEW FPGA это также означает, что каждая копия функции использует собственные FPGA секции – так что реентерабельность отлично подходит для увеличения компактности кода, но при этом используется больше вентилей.

Если вам не хватает вентилей FPGA, вы можете сделать функцию мультиплексированной, а не реентерабельной. Этот улучшенный способ не рассматривается в данном разделе, но в его основе – использование локальной памяти для запоминания значений регистров для каждого из вызывающих циклов, которые идентифицируются целым значение ID tag. Поскольку все циклы используют одни и те же секции FPGA (с различными адресами памяти для данных), каждая вызывающая программа блокирует другие вызовы, что замедляет выполнение. Однако количество вентилей значительно сокращается, поскольку одна и та же аппаратная логика SLICE используется многократно. Для многих управляющих приложений, где FPGA уже намного быстрее, чем ввод-вывод, это удобный вариант экономии вентилей. Несколько функций в палитре LabVIEW FPGA используют технику мультиплексирования для многоканальных операций с минимальным использованием вентилей FPGA. Среди них **PID** (ПИД), **Butterworth Filter** (Фильтр Баттерворта), **Notch Filter** (Режекторный фильтр) и **Rational Resampler** (Устройство передискретизации). Чтобы посмотреть, как это работает, перетащите одну из этих функций на блок-диаграмму и сконфигурируйте для работы с несколькими каналами. Затем щелкните по функции правой кнопкой мыши и выберите **Convert to SubVI** (Преобразовать в subVI) для исследования лежащего в основе кода.

Теперь рассмотрим главное преимущество, которое вы получите при разработке кода LabVIEW FPGA, как описано в предыдущих разделах.

Метод 3. Использование симуляции до компиляции

Третий способ разработки очень мощный, поскольку позволяет обойтись без компиляции, требующей значительных затрат времени, несмотря на ограниченные возможности отладки кода

LabVIEW FPGA. Одним из наиболее важных аспектов кода LabVIEW для встроенных разработок – его компактность. Код, написанный в LabVIEW FPGA, остается кодом LabVIEW – вы можете запускать его в Windows или на других устройствах и в других операционных системах. Главное отличие между этими целевыми устройствами - скорость выполнения кода и то, и поддерживает ли целевое устройства истинно параллельную обработку, как FPGA, или симулирует ее, как многопоточная операционная система микропроцессора.

LabVIEW FPGA обладает способностью запуска всего приложения LabVIEW FPGA в режиме симуляции, и это вы можете делать совместно с хост-процессором при тестировании со случайными данными, используемыми для операций чтения ввода-вывода FPGA, или со специальными VI для генерации симулированных сигналов ввода-вывода. Это особенно полезно для тестирования коммуникаций между FPGA и хостом, включая передачи данных с прямым доступом в память.

Однако неудобство такого подхода в том, что симулируется все приложение FPGA. Для разработки и тестирования новых функций LabVIEW, может быть выгоднее тестировать код каждой функции отдельно. Этот раздел фокусируется на возможности, называемой функциональной симуляцией, которая при отладке реализует подход "разделяй и властвуй", так что вы можете тестировать каждую функцию индивидуально до компиляции FPGA. На рисунке 6.50 показаны два примера функциональной симуляции, выполняющейся в Windows и используемой для целей тестирования и отладки.

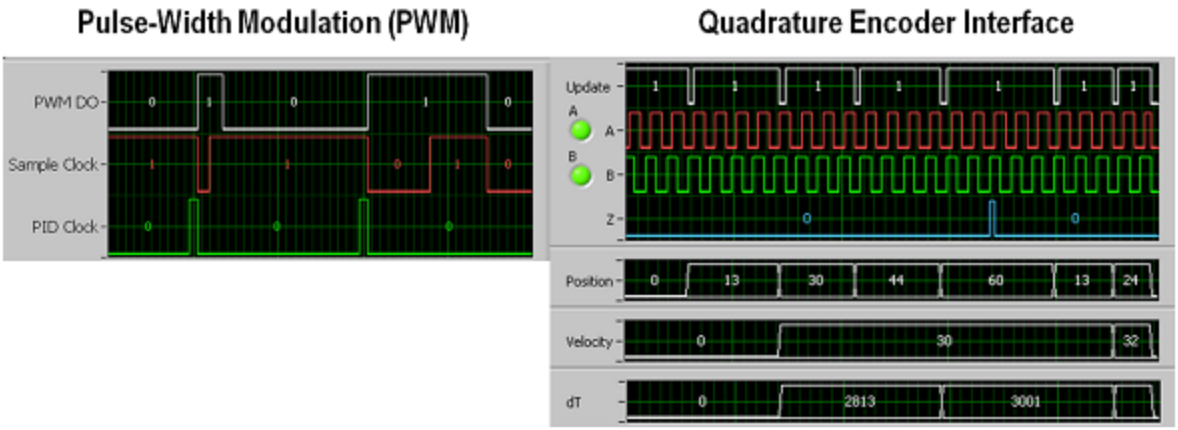


Рисунок 6.50. Скриншоты двух примеров функциональной симуляции, выполняющейся в Windows и используемой для целей тестирования и отладки
Pulse-Width Modulation – широтно-импульсная модуляция (ШИМ), Quadrature Encoder Interface – интерфейс квадратурного энкодера

В примере на рисунке 6.41 показана лицевая панель и блок-диаграмма тестового приложения, используемого для отладки subVI LabVIEW FPGA для генерации ШИМ. Тестовое приложение находится в разделе "My Computer" проекта LabVIEW, и при открытии запускается в Windows.

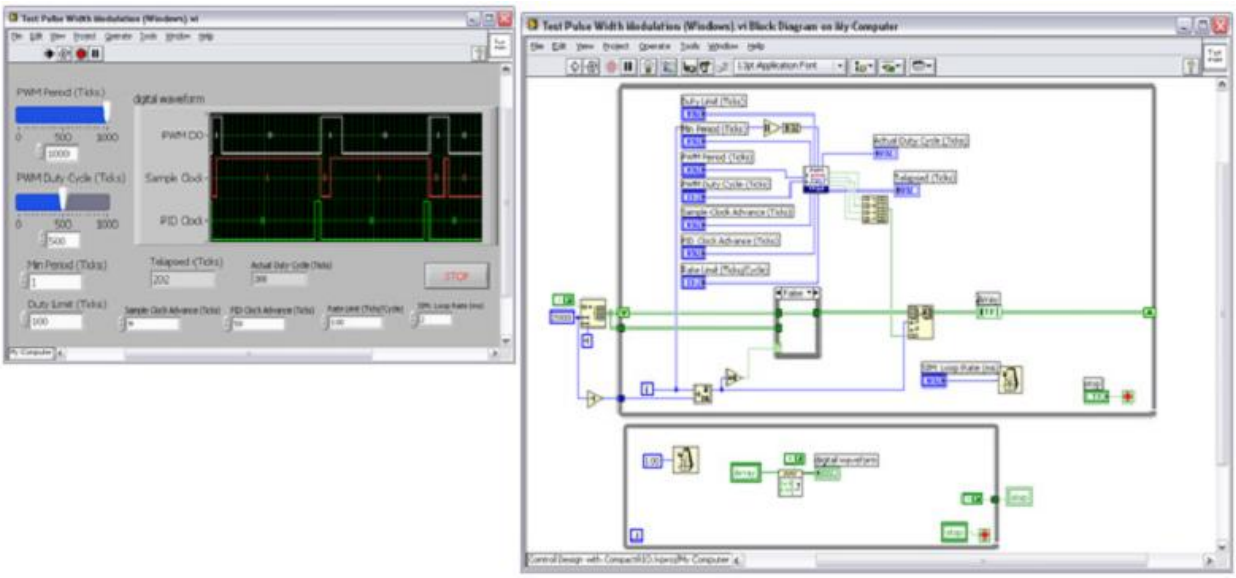


Рисунок 6.51. Лицевая панель и блок-диаграмма тестового приложения, используемого для отладки subVI LabVIEW FPGA для генерации ШИМ

Совет: Предусматривайте подсчет тактов для симуляции

С помощью структуры Conditional Disable вы можете в LabVIEW определить, какой лежащий в основе код используется при компиляции subVI для различных целевых устройств. В этом случае у вас есть функция Tick Count (счетчик тиков), выполняемая, если код скомпилирован для FPGA, и элемент лицевой панели, который выполняется, если код запускается в Windows. Благодаря этому вы можете при тестировании кода в Windows использовать симулированное значение счетчика импульсов времени, чтобы точнее моделировать и "битовые", и временные свойства.

Это техника используется в примере тестирования ШИМ (рисунок 6.51) – если subVI выполняется в Windows, а симулированный тактовый сигнал FPGA передается в subVI через терминал номера итерации цикла while верхнего уровня.

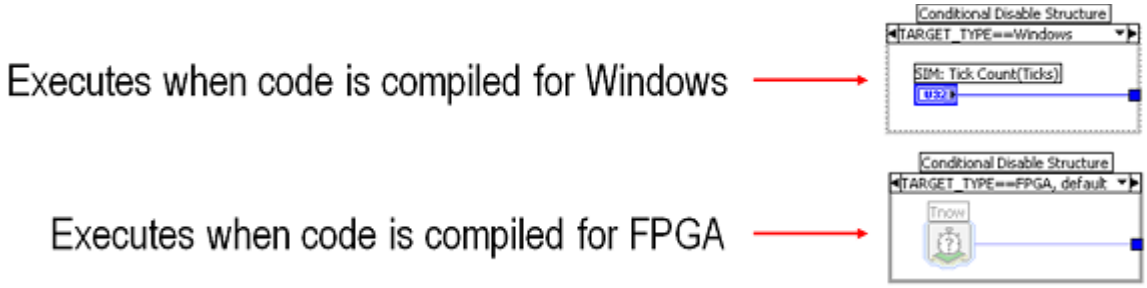


Рисунок 6.52. Функция Tick Count выполняется, если код скомпилирован для FPGA, а элемент лицевой панели реализуется, если код выполняется в Windows

Execute when code is compiled for Windows – выполняется, если код скомпилирован для Windows,
Execute when code is compiled for FPGA – выполняется, если код скомпилирован для FPGA

Как было показано выше, вы можете использовать функциональную симуляцию. чтобы до компиляции проводить тестирование, повторяя и убеждаясь в работоспособности логики FPGA,. Это также помогает использовать полный комплект инструментов отладки LabVIEW в процессе выполнения кода, и вы можете создавать "тестовые шаблоны" для проверки кода при различных условиях, которые иначе было бы сложно протестировать.

Используя симуляцию как этап процесса разработки, вы получаете следующие преимущества:

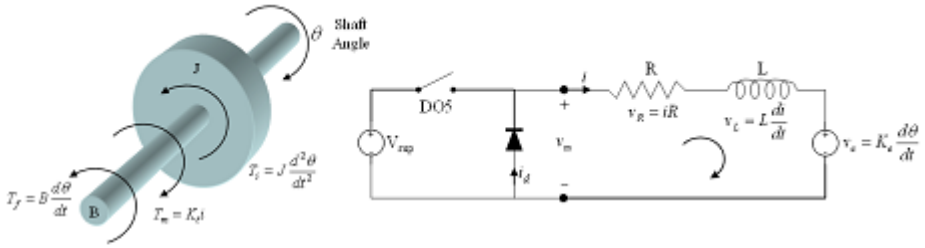
- Быстрое повторение и добавление свойств
- Получение уверенности в коде LabVIEW FPGA до его компиляции

- Использование всех возможностей LabVIEW при отладке (пробники, подсветка выполнения и т.д.)
- Проверка кода при различных условиях

Теперь продолжим симуляцию на шаг дальше для точного воспроизведения динамического поведения физической системы с обратной связью, с которой взаимодействует код LabVIEW FPGA.

Совет: Тестируйте код LabVIEW FPGA, используя модуль LabVIEW Control Design and Simulation

Модуль LabVIEW Control Design and Simulation (Разработка и симуляция управления) включает современную технологию симуляции мехатронных систем, таких, как двигатель постоянного тока, который управляется приложением LabVIEW FPGA. На рисунке 6.53 показаны уравнения теоретической модели щеточного двигателя постоянного тока, управляемого ШИМ-преобразователем и связанного с простой инерционной нагрузкой с внутренним трением.



Sum of Forces

$$\frac{d^2\theta}{dt^2} = \frac{K_t}{J} i - \frac{B}{J}\frac{d\theta}{dt}$$

Kirchhoff's Voltage Law

$$\frac{di}{dt} = -\frac{R}{L} i - \frac{K_e}{L}\frac{d\theta}{dt} + \frac{v_{in}}{L}$$

Рисунок 6.53. Уравнения теоретической модели щеточного двигателя постоянного тока, управляемого ШИМ-преобразователем и связанного с простой инерционной нагрузкой с внутренним трением
Sum of Forces – сумма сил, Kirchgoff's Voltage Law – закон Кирхгофа

Вы симулируете систему, используя подсистему LabVIEW Control Design and Simulation, содержащую узел формулы (formula node). Затем вы вводите в текстовом формате два дифференциальных уравнения, показанные на рисунке 6.53, в узлы формулы, как показано на рисунке 6.54. Используйте функции интегратора() для преобразования производных высокого порядка, например, для перехода от ускорения к скорости, а от скорости к перемещению.

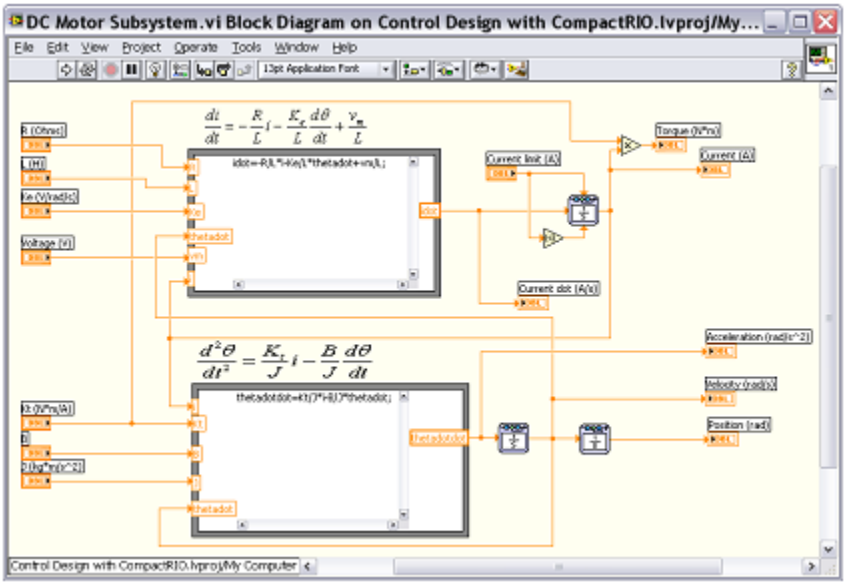


Рисунок 6.54. Текстовый формат двух дифференциальных уравнений, показанных на рисунке 6.53

Затем поместите подсистему Brushed DC Motor.vi в цикл симуляции верхнего уровня и подключите к функции LabVIEW FPGA для симуляции пульсирующего сигнала напряжения, используемого для управления двигателем. Результат - высокоточная симуляция поведения кода LabVIEW FPGA при интеграции его с реальной электромеханической системой с обратной связью.

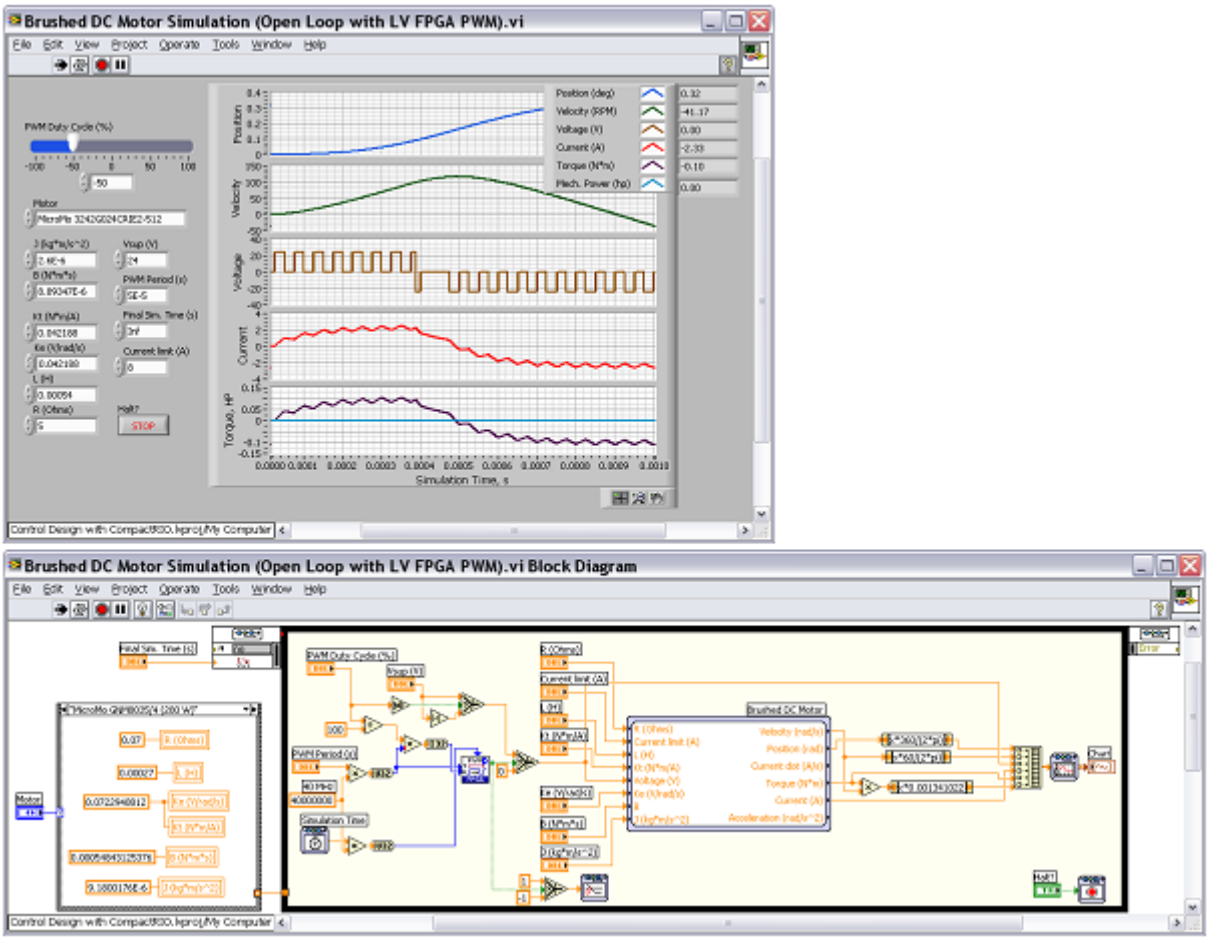


Рисунок 6.55. Высокоточная симуляция поведения кода LabVIEW FPGA при интеграции его с реальной электромеханической системой с обратной связью

Достоверность результатов симуляции была подтверждена реальными измерениями из развернутого приложения LabVIEW FPGA, управляющего двигателем с помощью модуля привода двигателя NI 9505, при этом получено почти идентичное совпадение между симулированными и измеренными сигналами.

С таким подходом вы можете проверять достоверность кода на более высоком уровне, чем базовая функциональная проверка. Представьте это как виртуальный машинный эмулятор, который вы можете использовать для предсказания того, как ваш код поведет себя в реальном мире. Вы можете использовать симуляции для облегчения проектных решений, оценки производительности, выбора компонентов и проверки наихудших условий. Вы можете даже настраивать контуры ПИД-регулирования, симулируя систему управления и наблюдая, как они работают с различными двигателями и при разных условиях нагружения. Симуляция могут также помочь выбрать правильные физические компоненты для системы, например, наилучшего двигателя для удовлетворения требований к производительности.

Метод 4. Синхронизация циклов

Перейдем к четвертому методу разработки: управление временными параметрами и синхронизацией кода LabVIEW FPGA.

Для большинства управляющих приложений время выполнения кода очень важно для производительности и надежности системы. К счастью, LabVIEW FPGA дает вам одновременно беспрецедентную скорость и полный контроль времени выполнения кода. В отличие от процессора, FPGA выполняет код действительно параллельно, а не только по одной инструкции за один раз. Это упрощает программирование, потому что вам не нужно волноваться о задании приоритетов и разделении процессорного времени между задачами. Каждый цикл управления – как изготовленный по заказу процессор, полностью выделенный под одну задачу. В результате получается высоконадежный высокопроизводительный код. Одним из преимуществ подобных свойств является то, что контуры регулирования при работе на высокой скорости, как правило, более стабильны, легче настраиваются и лучше реагируют на помехи.

В этом примере управления двигателем у вас есть два разных сигнала синхронизации – тактирования дискретизации и тактирования ПИД-регулирования. Это булевские сигналы, которые генерируются приложением для синхронизации циклов. Вы можете выполнять запуск как по положительному, так и по отрицательному фронту этих таковых сигналов.

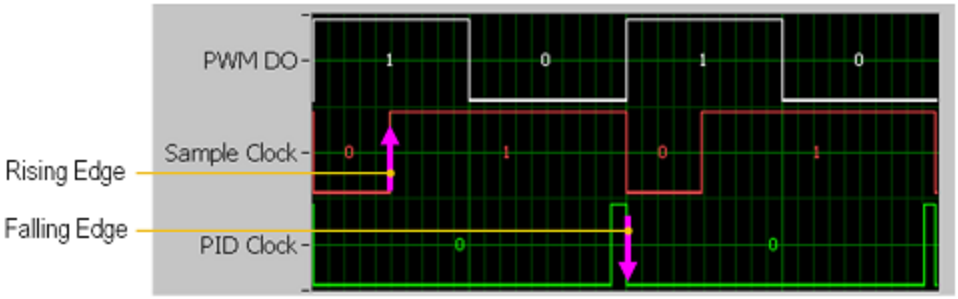


Рисунок 6.56. Пример управления двигателем с двумя разными тактовыми сигналами
Rising Edge – положительный фронт, Falling Edge – отрицательный фронт, PWM DO – цифровой выход ШИМ, Sample Clock – сигнал дискретизации, PID Clock – сигнал ПИД.

Теперь рассмотрим код LabVIEW FPGA, используемый для контроля этих сигналов и запуска по положительному или отрицательному фронту.

Обычно синхронизирующий тактовый булевский сигнал формируется в цикле следующим образом: сначала ожидается положительный или отрицательный фронт, а затем выполняется код LabVIEW FPGA, который вы хотите запустить при появлении условия запуска. Инженеры часто реализуют структуры последовательности, где первый кадр последовательности используется для ожидания запуска, а второй – для выполнения кода, как показано на рисунке 6.57.

Запуск по положительному фронту: в этом случае вы ждете, пока сигнал не перейдет из состояния "false" (0) в состояние "true" (1). Это осуществляется сохранением значения в сдвиговом регистре и использованием функции Greater Than? (Больше чем?). (Примечание: для инициализации итерационного терминала и предотвращения запуска при первой итерации к нему подключена константа " true ".).

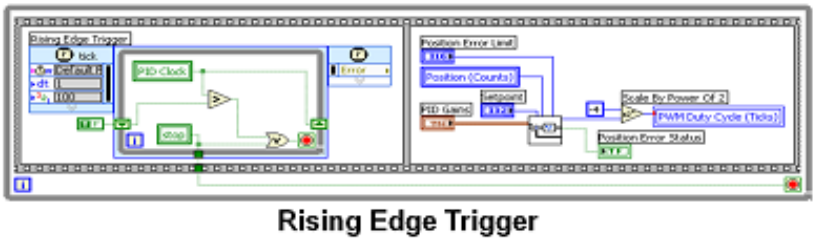


Рисунок 6.57. Пример запуска по положительному фронту

Запуск по отрицательному фронту: В этом случае используйте функцию Less Than? (Меньше чем?) для определения перехода из состояния "true" (1) в состояние "false" (0).

(Примечание: к терминалу итераций для инициализации подключена константа "false").

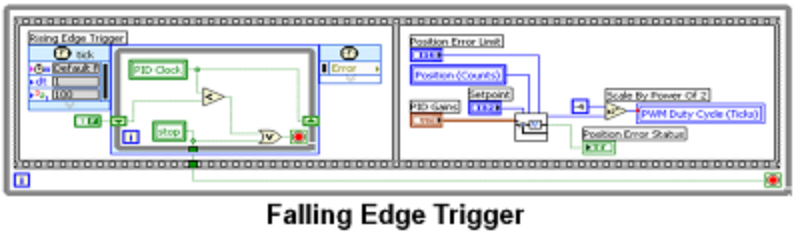


Рисунок 6.58. Пример запуска по отрицательному фронту

Запуск по аналоговому уровню: Используйте функцию Greater Than? (Больше чем?) для определения момента превышения аналоговым сигналом заданного вами порога, а затем используйте логический выход этой функции в качестве сигнала запуска. Этот случай, на самом деле - детектор переднего или заднего фронта, потому что вы используете функцию Not Equal? (Не равно?) для обнаружения переключения.

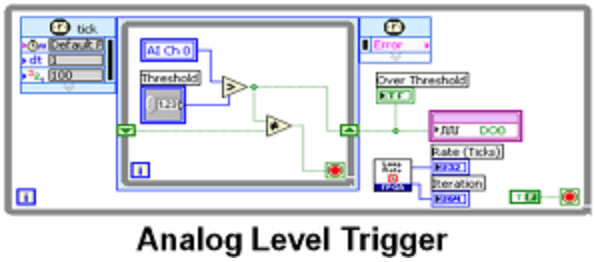


Рисунок 6.59. Пример запуска по аналоговому уровню

Теперь рассмотрим другое частое применение запуска – защелкивание значения сигнала при обнаружении события запуска.

Совет: Защелкивание значений

В этом случае вы используете запуск по положительному фронту для защелкивания значения на аналоговом входе из другого цикла в регистре "Latched Analog Input" (Зафиксированный аналоговый ввод). Это значение сохраняется постоянным, пока не произойдет следующее событие запуска. В этом примере операция с аналоговым входом фактически происходит в другом цикле, а вы используете локальную переменную для обмена данными между циклами. (Примечание: локальные переменные – хороший способ разделения данных между асинхронными циклами в LabVIEW FPGA).

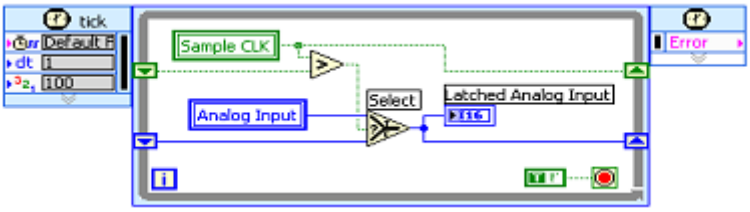


Рисунок 6.60. Операция с аналоговым входом в действительности происходит в другом цикле, а вы используете локальную переменную для обмена данными между циклами

Синхронизация с запуском и защелкиванием

На рисунке 6.61 показан пример практических способов запуска и защелкивания. LabVIEW FPGA обеспечивает истинный параллелизм выполнения операций. В этом примере у вас три независимых цикла. Это эквивалентно использованию трех заказных процессоров, работающих одновременно в одном чипе. Каждый цикл полностью загружен одной задачей, что ведет к наивысшему уровню надежности. Это также упрощает архитектуру программного управления в FPGA – в отличие от процессора, вам не нужно заботиться о том, что ваш код замедлится при добавлении нового кода.

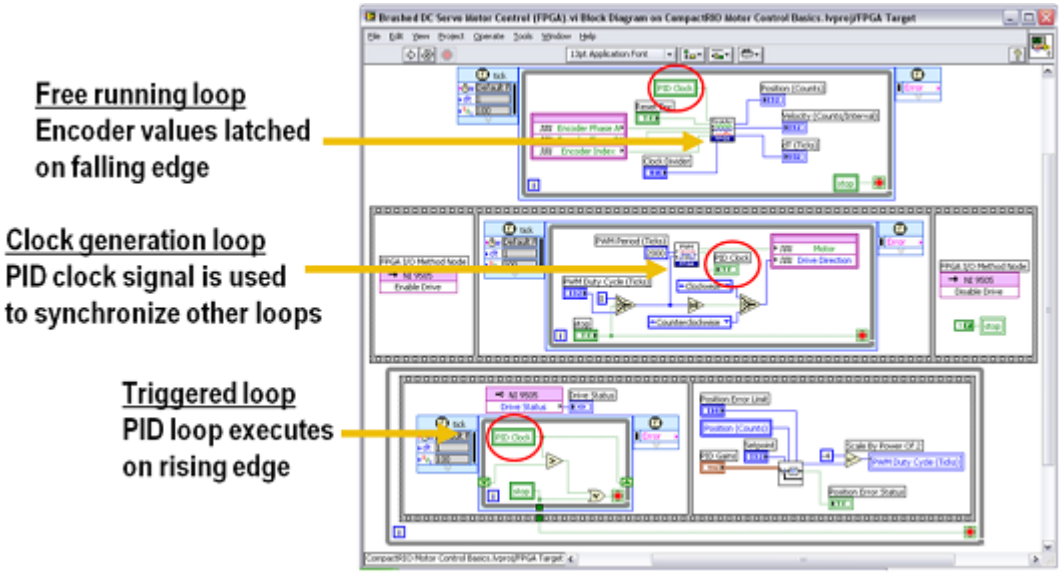


Рисунок 6.61. Каждый из трех независимых циклов полностью загружен собственной задачей, что ведет к наивысшему уровню надежности

Free running loop/Encoder values latches in falling edge – свободно выполняющийся цикл/ значения энкодера защелкиваются по отрицательному фронту, Clock generation loop/PID clock signal is used to synchronize other loops – цикл генерации тактового сигнала/тактовый сигнал ПИД используется для синхронизации других циклов, Triggered loop/PID loop executes on rising edge – синхронизирующий цикл/цикл ПИД выполняется по положительному фронту

Результаты наблюдений:

- Один цикл используется для генерации импульсов синхронизации, используемых другими циклами.
- Функция энкодера должна работать на полной скорости, чтобы не пропустить какие-нибудь цифровые импульсы. Эта функция работает с частотой 40 МГц, но защелкивает сигналы Position (Положение) and Velocity (Скорость) для синхронизации данных с другими циклами.
- Функция ПИД должна выполняться с заданной скоростью (20 кГц или 2000 тиков) и избегать джиттера, поскольку интегральный и дифференциальный коэффициенты усиления зависят от интервала времени, Ts. Если временной интервал будет изменяться,

или если одно и то же старое значение будет несколько раз передаваться в функцию, то эти коэффициенты усиления будут неправильными.

- В нижнем цикле вы видите, что выполнение кода запускается по положительному фронту тактового сигнала ПИД. Считайте локальную переменную для этого сигнала в этом цикле SCTL и выйдите из цикла при появлении положительного фронта. Затем выполните алгоритм 32-разрядного ПИД, который находится в модуле NI SoftMotion Development Module for LabVIEW. Здесь считывается заданное положение, сравнивается с положением, измеренным энкодером, и генерируется команду для цикла ШИМ. В этом случае вы используете функцию Scale by Power of 2 (Масштабирование степенью 2) для деления выходного сигнала ПИД 2^-4, что эквивалентно делению на 16. Тем самым производится масштабирование значений к диапазону ±2,000 тиков, что необходимо для функции ШИМ. Значение в 1000 тиков эквивалентно 50-процентому коэффициенту заполнения, поскольку период ШИМ составляет 2000 тиков.
- Обратите внимание, что два верхних цикла работают на частоте 40 МГц. Нижний цикл запускается на выполнение с частотой 20 кГц тактовым сигналом ШИМ. (На выполнение синхронизируемой функции ПИД из палитры NI SoftMotion требуется 36 тиков).

Метод 5. Избегайте "пожирателей вентилей"

Теперь, когда вы ознакомились с четырьмя ключевыми способами разработки кода LabVIEW FPGA, рассмотрим последний метод: как избежать "пожирателей вентилей" (gate hogs). Этот, часто невинно выглядящий, код может "съесть" много вентилей FPGA (также известных как slices). Известно три наиболее распространенных "пожирателя вентилей":

Большие массивы или кластеры: создание на лицевой панели большого массива или кластера индикаторов или элементов управления - наиболее распространенная ошибка программистов, которая съедает много вентилей FPGA. Если вам не нужен индикатор на лицевой панели для коммуникации с хост-процессором – не создавайте его. Если вам нужно передавать больше дюжины элементов массива, используйте прямой доступ к памяти (DMA). Избегайте также, если это возможно, функции манипуляций с массивами, подобных этой: **Rotate 1D Array** (Вращение одномерного массива).

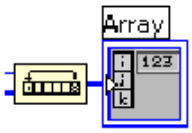


Рисунок 6.62. Избегайте, если это возможно, функции манипуляций с массивами, подобных Rotate 1D Array

Quotient and Remainder (Частное и остаток): эта функция реализует целочисленное деление. Выход частного, floor(x/y) дает результат деления x на y, округленный до ближайшего целого. Выход остатка – x-y*floor(x/y). Например, если разделить 23 на 5, то получится частное 4 и остаток 3. Эта функция интенсивно использует вентили, и ее выполнение требует несколько тактов, так что ее нельзя использовать в SCTL. Подключите к терминалам данные, минимально нужного вам типа при работе с этой функцией и используйте по возможности константы, а не элементы управления.

Рисунок 6.63. Функция вычисления частного и остатка

Scale by Power of 2 (Масштабирование степенью 2): если терминал n положительный, то эта функция умножает вход x на 2 в степени n (2^n). Если n отрицательно, функция делит на 2^n. Например, при n, равном +4 происходит умножение на 16, а при n, равном -4 выполняется деление на 16. Эта функция значительно более эффективна, чем функция "Частное и остаток". Однако по возможности используйте константу минимального размера для терминала n.

Рисунок 6.64. Функция масштабирования степенью 2

Примечание: прямой доступ к памяти (DMA) – намного более лучший способ для передачи массива данных к хосту, чем создание для массива индикаторов на лицевой панели и с использование метода **FPGA Read/Write**. Массивы полезны для хранения набора одновременно собранных данных для помещения в буфер DMA и передачи их на хост-компьютер. Хорошо использовать массивы при упаковке отсчетов данных для индексации в функции DMA Write, если только вы не создадите массив индикаторов на лицевой панели. Использование автоиндексации цикла for для записи данных в буфер DMA также нормально, если вы не создадите массив индикаторов на лицевой панели, потому что компилятор хорошо оптимизирует массивы, передаваемые в циклы for для целей индексации.

Совет: Избегайте массивов на лицевой панели для передачи данных

При оптимизации кода для уменьшения занимаемых им ресурсов в FPGA, вы должны рассмотреть реализуемые вами на лицевой панели элементы управления и индикаторы. Каждый объект на лицевой панели и представленные в нем данные занимают значительную часть пространства FPGA. Уменьшая количество этих объектов и уменьшая размер массивов на лицевой панели, вы можете значительно сократить размер пространства FPGA, необходимого вашему VI.

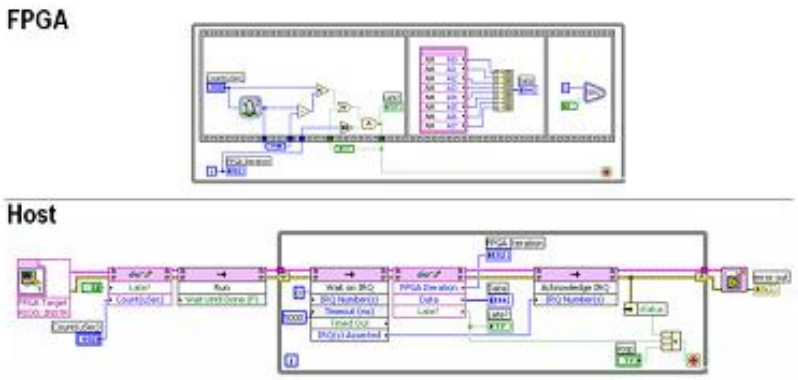


Рисунок 6.65. Создание больших массивов для хранения данных и передачи хост-приложению

Вместо создания больших массивов для хранения данных и передачи хост-приложениям (как показано на рисунке 6.65), используйте DMA для передачи массива значений вводимых аналоговых сигналов хост-процессору, как показано на рисунке 6.66.

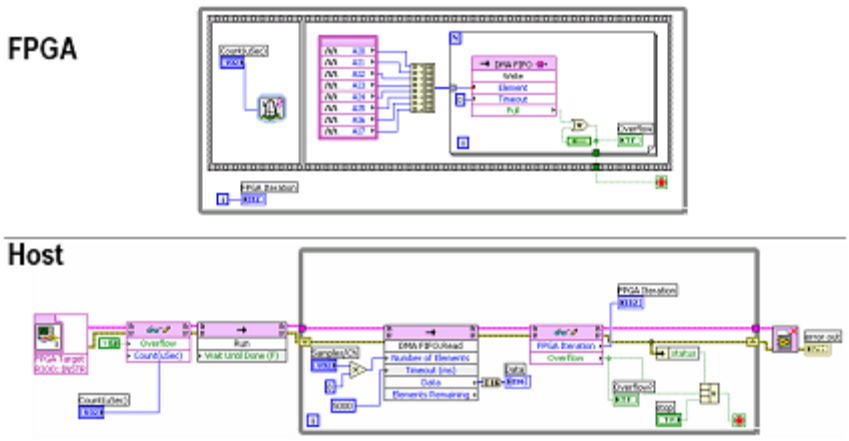


Рисунок 6.66. Использование DMA для передачи массива значений вводимых аналоговых сигналов хост-процессору

Совет: Используйте DMA для передачи данных

DMA использует память FPGA для хранения данных и высокоскоростной передачи в память хост-процессора при очень небольшой загрузке процессора. При этом для передачи больших блоков данных требуется значительно меньше циклов процессора по сравнению с использованием метода FPGA Read/Write и индикаторов на лицевой панели.

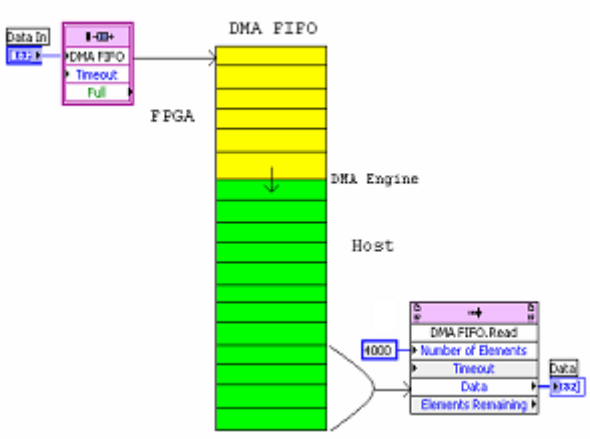


Рисунок 6.67. DMA использует память FPGA для хранения данных и их высокоскоростной передачи в память хост-процессора при очень небольшой загрузке процессора

Советы программисту при реализации DMA

- При определении размера буфера FPGA, можно использовать размер по умолчанию (1023). Как правило, незачем создавать буфер большего размера.
- Вы должны задать размер буфера на хосте больше, чем определено по умолчанию. По умолчанию размер буфера хоста в два раза больше, чем буфера в FPGA. Вы должны установить его равным как минимум двойному количеству элементов (**Number of Elements**), которые собираетесь использовать.
- Если вы передаете массив данных, вход **Number of Elements** (количество элементов) всегда должен быть целым числом, кратным размеру массива. Например, если вы передаете массив из 8 элементов, **Number of Elements** должно быть целым числом, кратным 8 (например, 80, что дает 10 отсчетов каждого из 8 элементов).
- Каждая транзакция DMA использует служебные данные, так что обычно лучше читать данные большими блоками. Функция **DMA FIFO.Read** автоматически ожидает, пока станет доступно затребованное вами количество элементов, что минимизирует загрузку процессора.
- Упаковка 16-битных данных канала в формат U32 (поскольку в DMA используется тип данных U32) обычно не дает преимуществ в CompactRIO, поскольку шина PCI обладает очень высокой пропускной способностью передачи данных в режиме DMA, и, скорее всего, вы даже не приблизитесь к использованию всего быстродействия шины. Обычно процессор представляет собой узкое место при обработке потоковых данных. Упаковка данных в FPGA означает, что они должны быть распакованы в процессоре, что добавляет процессору работы. Поэтому вы должны посылать каждый канал как U32, даже если собираете 16-битные данные.
- Выход "Full" (Переполнение) функции DMA FIFO Write на самом деле - индикатор ошибки. Переполнение не должно происходить при нормальной работе, так что NI рекомендует остановить приложение, если вы увидите эту ошибку, и сбросить FPGA до повторного запуска.

Совет: Используйте минимально необходимый тип данных

При программировании в LabVIEW FPGA важно использовать минимальный необходимый тип данных. Например, использование 32-разрядного целого со знаком (I32) для индексации структуры case, скорее всего, излишне, потому что вряд ли вы напишете код для **двух миллиардов**

различных вариантов. Обычно беззнакового 8-разрядного целого (U8) вполне хватает для обработки до 256 вариантов.

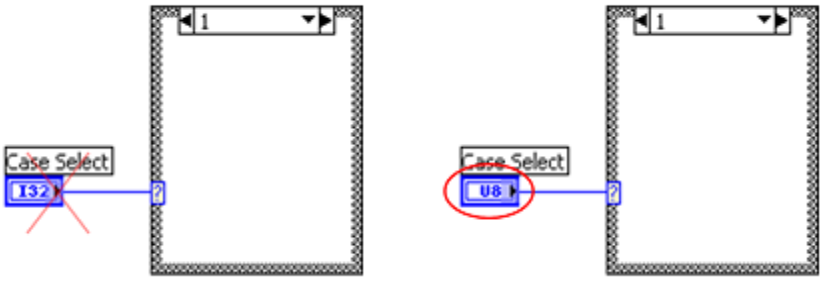


Рисунок 6.68. Используйте беззнаковое 8-разрядное целое (U8) для индексации структуры case, поскольку так можно обработать до 256 различных вариантов

Совет: Оптимизируйте размер

Это приложение FPGA слишком велико для компиляции. Почему?

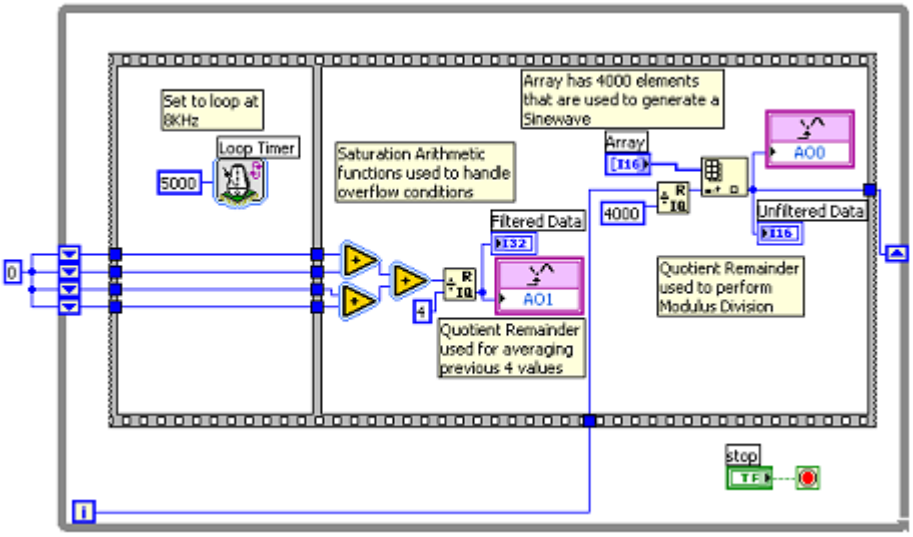


Рисунок 6.69. Этот VI слишком велик для компиляции

Это приложение использует массивы для хранения значений синусоиды. Массив индексируется для получения значений. Кроме того, четыре предыдущих значения запоминаются в сдвиговых регистрах и усредняются. Этот VI слишком велик для компиляции. Что можно сделать для оптимизации кода?

"Пожиратели вентилей", найденные в коде: большие массивы на лицевой панели, функции частного и остатка.

Для улучшения этого приложения замените массив таблицей преобразования (LUT), как показано на рисунке 6.70.

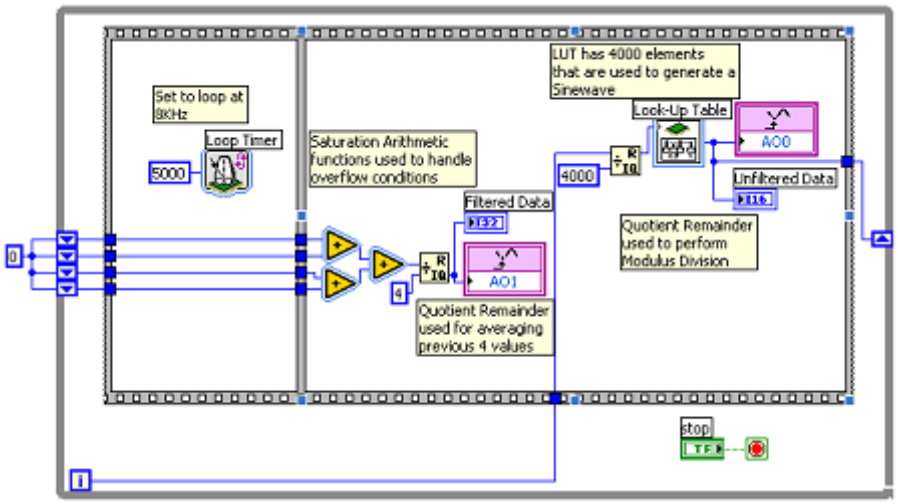


Рисунок 6.70. Улучшите приложение, заменив массив таблицей преобразования

Только одно это изменение позволит скомпилировать программу, и она будет использовать только 18 процентов из 1M вентилей FPGA. Можно ли еще оптимизировать программу?

Далее удалите обе функции Quotient and Remainder (Частное и остаток). Одна из них использовалась для индексации LUT и теперь заменена счетчиком со сдвиговым регистром - очень распространенная техника в FPGA. Другая функция вычисления частного и остатка заменена масштабированием степенью 2. Поскольку функция "scale by 2" имеет на входе константу, она использует очень мало ресурсов FPGA. Примечание: масштабирование на 2^ -2 равноценно делению на 4.

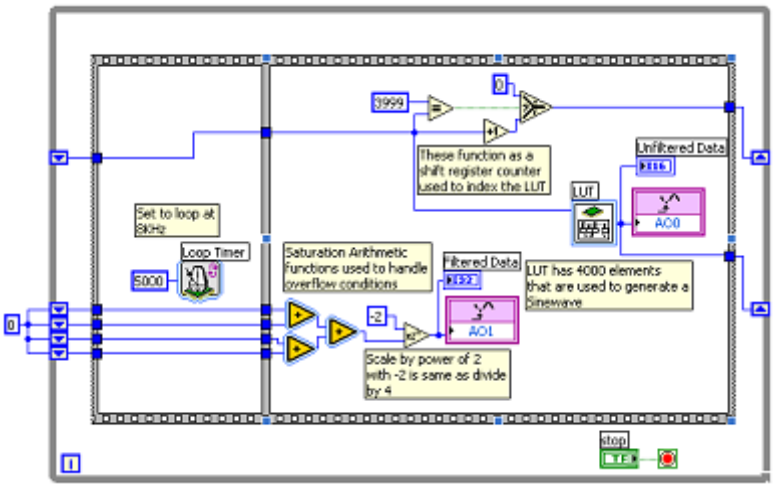


Рисунок 6.71. Удалите обе функции Quotient and Remainder (Частное и остаток) для большей оптимизации программы

Теперь приложение занимает всего 9 процентов вентилей FPGA.

Совет: Дополнительные приемы оптимизации FPGA приложений

Для получения дополнительной информации по этой теме обратитесь к документу *Optimizing FPGA VIs for Speed and Size* (Оптимизация VI FPGA по быстродействию и размеру) в Зоне разработчика NI (NI Developer Zone). Этот документ содержит подробную информацию о более чем десяти способах оптимизации приложений LabVIEW FPGA.

Техника оптимизации	**Скорость FPGA**	**Размер FPGA**
Сокращайте комбинаторные пути	√	
Используйте конвейерную обработку	√	
Используйте циклы SCTL	√	√
Используйте параллельные операции	√	
Выбирайте подходящие варианты арбитража	√	√
Используйте нереентерабельные subVI		√
Используйте реентерабельные subVI	√	
Ограничивайте количество объектов лицевой панели, таких, как массивы		√
Используйте наименьший возможный тип данных	√	√
Избегайте, когда возможно, больших VI и функций	√	√
Планируйте синхронизацию с использованием сигналов рукопожатия	√	√

Таблица 6.3. Дополнительные приемы оптимизации FPGA приложений

РАЗДЕЛ 7
Создание сетевого пользовательского интерфейса для взаимодействия с CompactRIO

Построение пользовательских и человеко-машинных (HMI) интерфейсов с помощью LabVIEW

В настоящем документе описывается проектирование архитектуры программного обеспечения, предназначенного для построения интерфейса оператора с масштабируемым средством навигации, с помощью которого осуществляется циклическое перемещение по различным страницам человеко-машинного интерфейса (human-machine interface – HMI). Такую архитектуру можно использовать в LabVIEW для построения HMI интерфейса на основе для любых аппаратных платформ HMI, в том числе:

- NI TPC-2512 под Windows XP Embedded (TPC, touch panel computer – компьютер с сенсорной панелью);
- NI TPC-2106 под Windows CE;
- NI PPC-2115 под Windows XP (PPC, panel PC – пультовый персональный компьютер).

LabVIEW является полноценным языком программирования, который дает одно решение для множества задач проектирования – от HMI/SCADA систем до надежных и детерминированных систем управления. Программный модуль LabVIEW Touch Panel Module предоставляет графический программный интерфейс, который можно использовать для разработки HMI приложения в среде проектирования под Windows и развертывания его затем в компьютере с сенсорной панелью типа TPC производства National Instruments или любых HMI системах, работающих под управлением Window CE. Данный документ является руководством для инженеров, разрабатывающих HMI системы как под Windows Vista/XP, так и Windows CE.

Основы HMI архитектуры

HMI архитектура может быть простой или сложной в зависимости от требуемой функциональности. Выбор правильной архитектуры программного обеспечения определяет как функциональность HMI интерфейса, так и возможность его расширения и адаптации для будущих технологий. Базовый HMI интерфейс состоит из трех основных программных модулей:

1. Программные модули инициализации и отключения (служебные подпрограммы)
2. Цикл сканирования каналов ввода-вывода (табличная память, драйверы ввода-вывода и коммуникационных портов)
3. Цикл навигации (для пользовательского интерфейса)

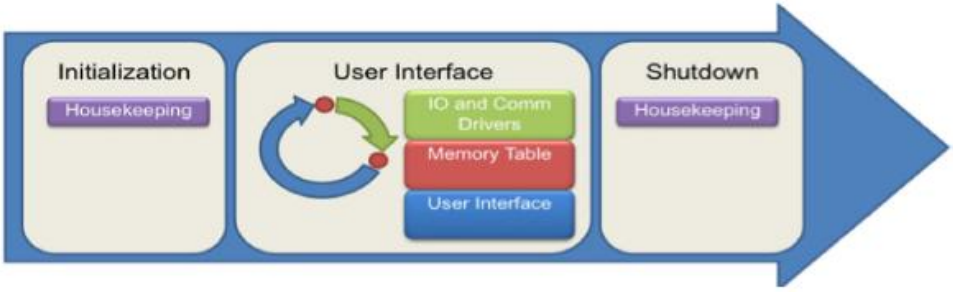


Рисунок 7.1. Программные модули типового HMI интерфейса

Housekeeping – Служебные подпрограммы, Initialization – Инициализация, User Interface – Пользовательский интерфейс, IO and Comm Drivers – Драйверы ввода-вывода и коммуникационных портов, Memory Table – табличная память, Shutdown – Отключение

Инициализация и отключение

Перед выполнением цикла сканирования каналов ввода-вывода и цикла навигации HMI интерфейс должен выполнить подпрограмму инициализации, которая устанавливает все органы управления, индикаторы, внутренние переменные и переменные для обмена информацией с аппаратурой (контроллером) в исходные состояния (по умолчанию). Вы можете дополнить подпрограмму инициализации таким образом, чтобы подготовить HMI интерфейс для операций, подобных записи данных в файлы. Перед остановкой системы задача отключения закрывает все ссылки и выполняет дополнительные задачи, например, запись в файлы ошибок.

Цикл сканирования каналов ввода-вывода

В целом HMI архитектура очень похожа на архитектуру Compact RIO. Для обмена данными между задачами в ней одновременно используется табличная память и командно-ориентированная архитектура.

Табличная память

В рассматриваемой архитектуре в цикле сканирования каналов ввода-вывода происходит запись и считывание данных из сети, а также размещение их в табличной памяти. Другие задачи, такие, как страницы пользовательского интерфейса, считывают и записывают данные из (в) табличной памяти. Табличная память создается с использованием переменных общего доступа типа Single Process и сканирования ввода-вывода и считывает данные из переменных общего доступа, публикуемых в сети (Network-Published Shared Variables). Таким образом уменьшается объем служебных данных, и создается более масштабируемое приложение.

Командно-ориентированная архитектура

При сканировании ввода-вывода запускается также обработчик команд, который преобразует пользовательские команды для системы реального времени в сетевые команды. Чтобы обеспечить надежность функционирования, каждая пользовательская команда HMI интерфейса должна быть гарантированно передана в контроллер реального времени. Это осуществляется с помощью командно-ориентированной архитектуры, которая включает в себя очередь команд типа FIFO.

Рисунок 7-2. Простая командно-ориентированная архитектура с использованием сетевых переменных

Commander – Начальник, Command – команда, Network Variable – сетевая переменная, Worker – Исполнитель

В сетевой переменной общего доступа, для которой разрешена буферизация, создается память FIFO, которая используется в качестве транспортного уровня для передачи команд из

программного интерфейса HMI в контроллер. В то же время, очередь команд LabVIEW используется для передачи команд из страниц пользовательского интерфейса обработчику команд в цикле сканирования каналов ввода-вывода.

За подробной информацией о сетевых командах и архитектуре обработчика для получения и обработки команд от HMI обратитесь к параграфу "Обмен информацией".

Цикл навигации

Этот цикл предназначен для управления пользовательскими интерфейсами (отображением страниц). Он включает в себя страницы пользовательского интерфейса и механизм навигации, который помогает циклически перемещаться по различным доступным страницам HMI интерфейса.

Страницы пользовательского интерфейса

Каждая страница пользовательского интерфейса является VI LabVIEW. К наиболее распространенным компонентам страницы пользовательского интерфейса относятся навигационные и командные кнопки, числовые индикаторы, графики, логические (булевские) органы управления и индикаторы и изображения. Значения, которые должны отображаться на странице графического интерфейса, считываются из переменных общего доступа типа Single Process.

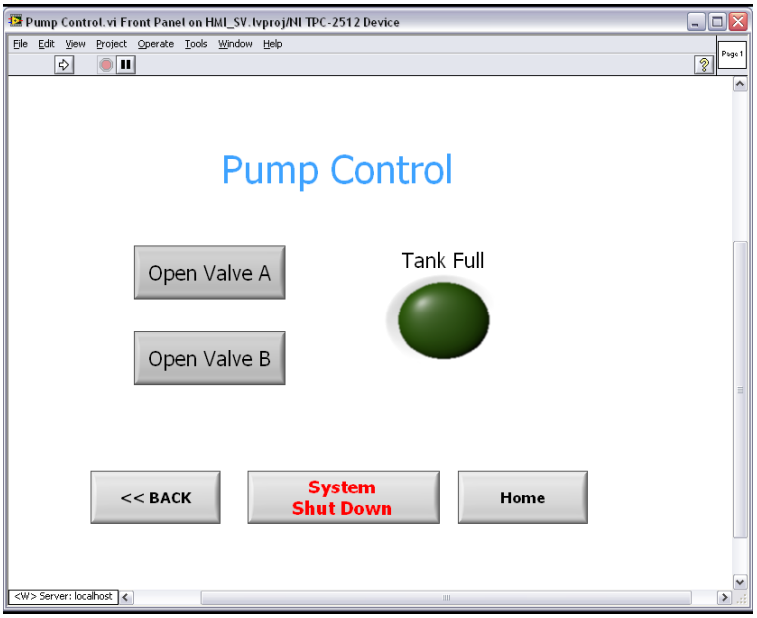


Рисунок 7.3. Пример типовой страницы пользовательского интерфейса в LabVIEW

Pump Control – управление насосом, Open Valve – открытие клапана, Tank Full – бак заполнен, Back – назад, System Shut Down – выключение системы, Home – в начало

Такие действия пользователя, как нажатие кнопки, являются командами, как для механизма навигации, так и для контроллера реального времени. С помощью структуры обработки событий (event structure) вы перехватываете команды и посылаете их механизму навигации или посредством очереди передаете обработчику команд, который размещает их в сети. Облегчить создание управляемого событиями графического интерфейса можно с помощью стандартных шаблонов обработчиков, которые есть в LabVIEW. Событие следует преобразовать в соответствующую команду, поставив данные в очередь на обработку обработчиком команд.

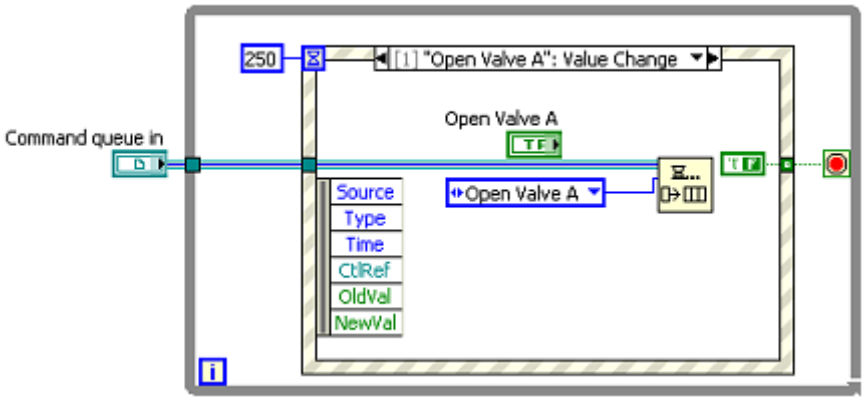


Рисунок 7.4. Передача данных со страницы пользовательского интерфейса в очередь команд

Механизм навигации

Механизм навигации является частью рассматриваемой архитектуры, которая находится в цикле навигации и отвечает за управление страницами графического интерфейса. Этот механизм представляет собой конечный автомат, который состоит из цикла while (с предусловием) и структуры выбора (case). Внутри каждого кадра case содержится своя страница графического интерфейса, которая реализуется набором VI LabVIEW и открывается при вызове.

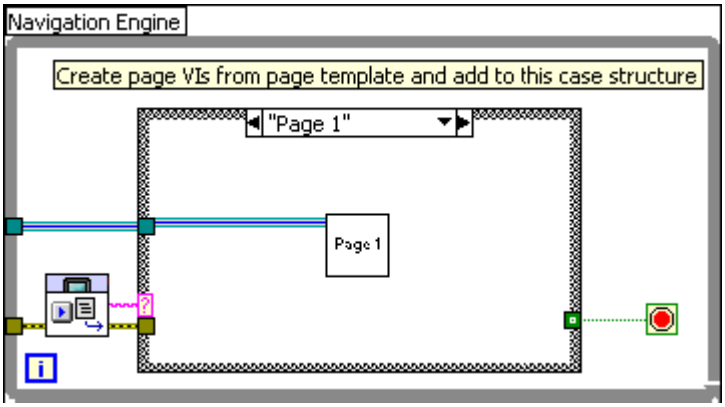


Рисунок 7.5. Блок-диаграмма цикла навигации

VI, используемые механизмом навигации, находятся в палитре Touch Panel Navigation.

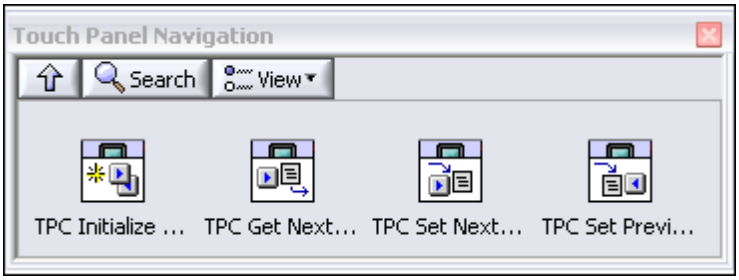


Рисунок 7.6. Палитра навигации

TPC Initialize Navigation

Данный VI инициализирует механизм навигации путем установки глубины предыстории и имени первой отображаемой страницы.

TPC Get Next Page

Данный VI возвращает имя следующей страницы и передает его в case структуру механизма навигации HMI.

TPC Set Next Page

Этот VI задает имя следующей отображаемой страницы. Этот VI следует использовать внутри HMI страниц для поддержки навигационных кнопок.

TPC Set Previous Next Page

Данный VI задает имя предыдущей страницы из предыстории страниц. Этот VI следует использовать внутри HMI страниц для поддержки навигационной кнопки "back" (назад).

Состояние страницы и предыстория хранятся в функциональной глобальной переменной, к которой имеет доступ каждый из рассмотренных выше VI.

Базовая HMI архитектура для операционных систем Windows XP, XP Embedded и CE

Операционные системы XP и XP Embedded рекомендованы для полнофункциональных HMI приложений. У них возможности графики значительно выше, и LabVIEW под их управлением работает более гибко. Кроме того, вы можете использовать Windows XP для более простой интеграции других устройств ввода-вывода или для обмена данными с устройствами, в которых применяются другие механизмы обмена данными, например, OPC. Тем не менее, чтобы без затруднений обеспечить масштабируемость, в целом HMI архитектура является одинаковой для платформ под Windows XP, XP Embedded и CE.

Чтобы продемонстрировать эту архитектуру, создадим простое HMI приложение, которое состоит из двух страниц графического интерфейса – начальной страницы и страницы управления насосом. На каждой из этих страниц имеются навигационные кнопки, с помощью которых можно перейти на другую страницу. Кроме навигационной кнопки на страницах есть элементы управления и индикаторы, значения которых считываются с контроллера через сетевые переменные общего доступа в цикле сканирования каналов ввода-вывода и сохраняются на локальном компьютере в переменных общего доступа типа Single Process. К такому каркасу можно добавлять еще много страниц, которые хорошо согласуются с вашим приложением.

Таблица ввода-вывода

Все данные, относящиеся к процессу, хранятся в табличной памяти HMI. Табличная память построена на основе переменных общего доступа типа Single Process. Из нее можно считывать или записывать в нее данные при сканировании ввода-вывода или с помощью отдельных страниц графического интерфейса. Чтобы создать таблицу ввода-вывода, выполните следующие действия:

- Добавьте в проект LabVIEW сенсорную панель в качестве целевого устройства.
- Создайте папку subVI под целевым устройством. Внутри этой папки создайте папку I/O Memory Table.
- Щелкните по этой папке правой кнопкой мыши и выберите команду **New»Variable** (Новая»Переменная).
- Создайте переменные типа Single Process. Поскольку сенсорная панель работает не под операционной системой реального времени, для этих переменных не разрешена память FIFO реального времени.
- Сохраните созданные вами проект и библиотеку

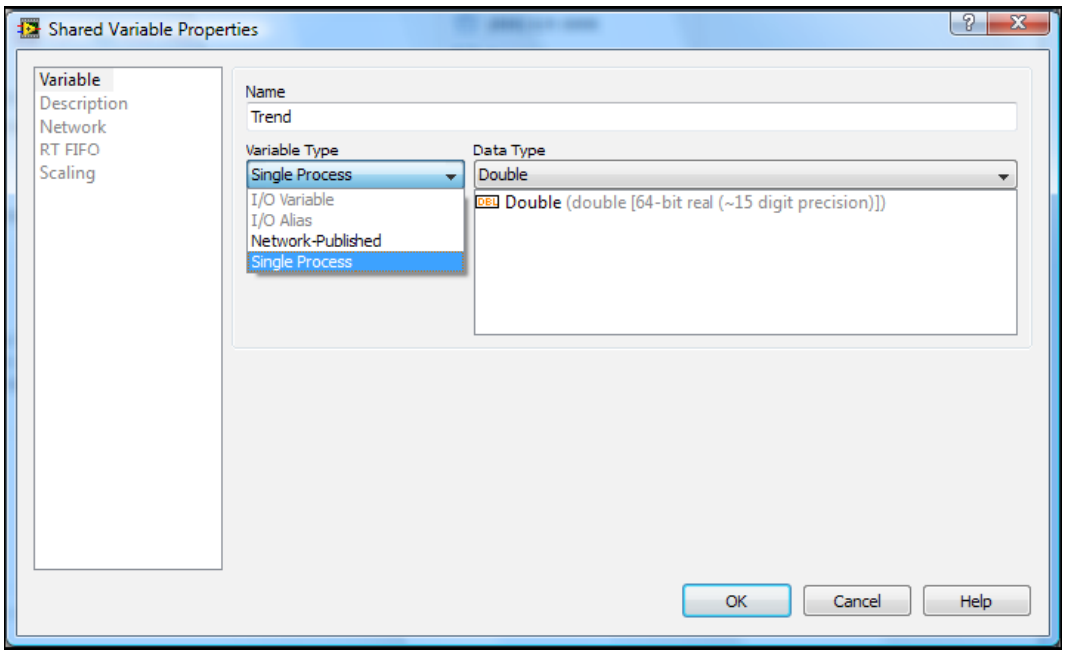


Рисунок 7.7. Создание переменных типа Single Process

Задача инициализации

При запуске HMI интерфейса задача инициализации устанавливает органы управления, индикаторы и внутренние переменные в состояния по умолчанию, а также создает очередь для внутренних команд.

1. Откройте VI верхнего уровня и создайте структуру последовательности (flat sequence – "плоская последовательность").
2. В первом кадре создайте subVI и сохраните его в качестве HMI интерфейса по умолчанию (Default HMI).
3. Откройте этот subVI и задайте значения по умолчанию для табличной памяти путем записи в переменные общего доступа типа Single Process. Сохраните эти изменения.

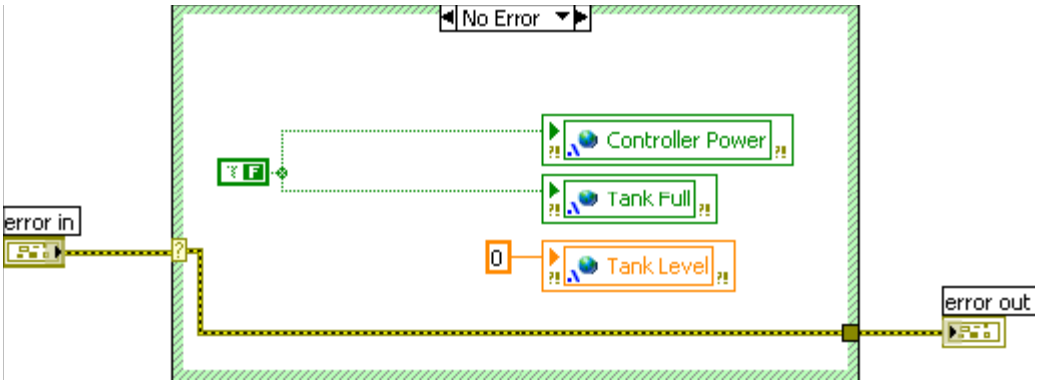


Рисунок 7.8. Присваивание переменным общего доступа типа Single Process значений по умолчанию

4. К задаче инициализации также относится создание очереди команд, чтобы любые команды, которые необходимо отправить контроллеру, выстраивались в очередь без потерь. Чтобы программа получалась более масштабируемой, задайте для данных, которые выстраиваются в очередь, тип enum (перечислительный). Это необходимо, для того, чтобы при добавлении команд все изменения автоматически распространялись на весь код программы.

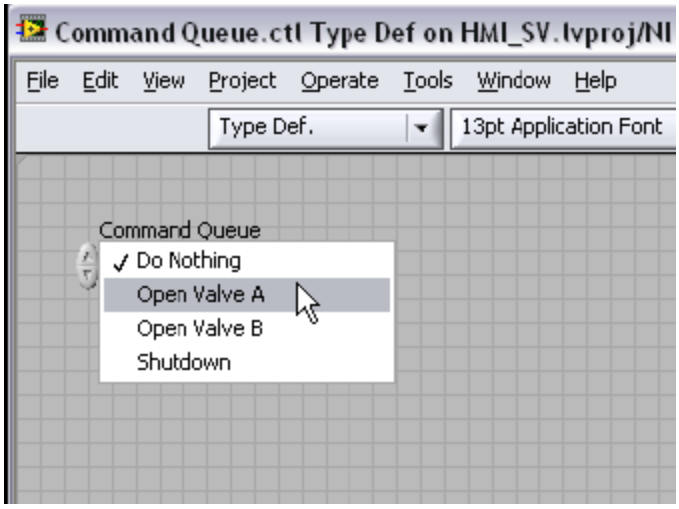


Рисунок 7.9. Тип данных enum обеспечивает масштабируемость

5. D первом кадре создайте также очередь команд и подключите к ней только что созданный элемент управления перечислительного типа.

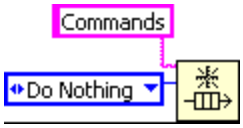


Рисунок 7.10. Создание очереди команд на основе элемента перечислительного типа

Цикл сканирования каналов ввода-вывода

В цикле сканирования происходит обмен данными и командами между сетью, табличной памятью и очередью команд. Чтобы создать цикл сканирования, выполните следующие действия:

1. На блок-диаграмме во второй кадр структуры последовательности поместите цикл while. С помощью функции Wait(ms) (Ожидание, мс) задайте время выполнения цикла, соответствующую вашему приложению.
2. На блок-диаграмме создайте два subVI: Network Communication.vi и HMI Command Handler.vi.

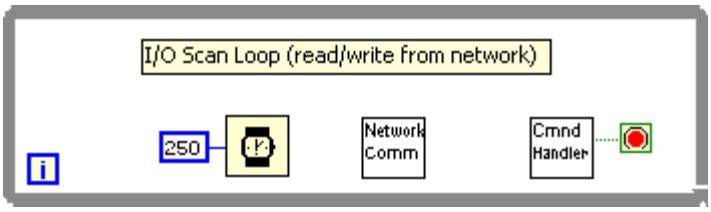


Рисунок 7.11. Цикл сканирования ввода-вывода

3. Откройте subVI Network Communication. На блок-диаграмму поместите переменные общего доступа типа Single Process и соедините их с соответствующими сетевыми переменными общего доступа типа Network-Published Shared Variables (обслуживаемыми в контроллере Compact RIO). В этом примере простая проверка ошибок организована таким образом, что, если возникают какие-либо ошибки или предупреждения в сетевой переменной, никакие данные в табличную память (в переменные типа Single Process) не помещаются.

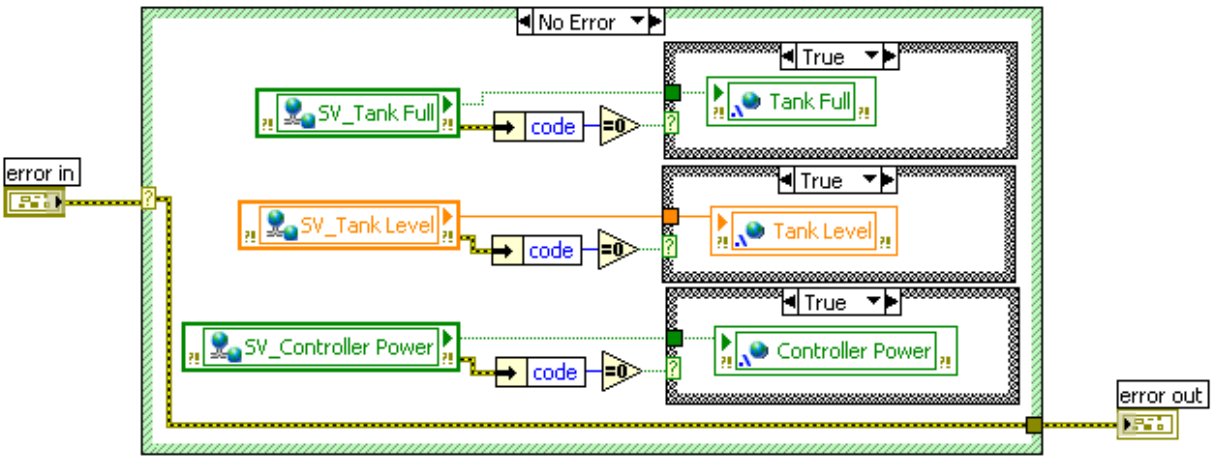


Рисунок 7.12. Данные из сетевых переменных вставляются в табличную память

4. Откройте SubVI HMI Command Handler (Обработчик команд HMI), который пересылает по сети любые команды в контроллеры реального времени. Соответствующие команды, помещенные в очередь на страницах графического интерфейса, извлекаются из очереди и пересылаются в контроллер с помощью сетевой переменной "Command", обслуживаемой в контроллере реального времени. Создайте блок-диаграмму, приведенную на рисунке 7.13 с case структурой, где каждому варианту выбора соответствует определенная команда, посылаемая в контроллер. В качестве выбора по умолчанию принята команда "Do Nothing" (Бездействие).

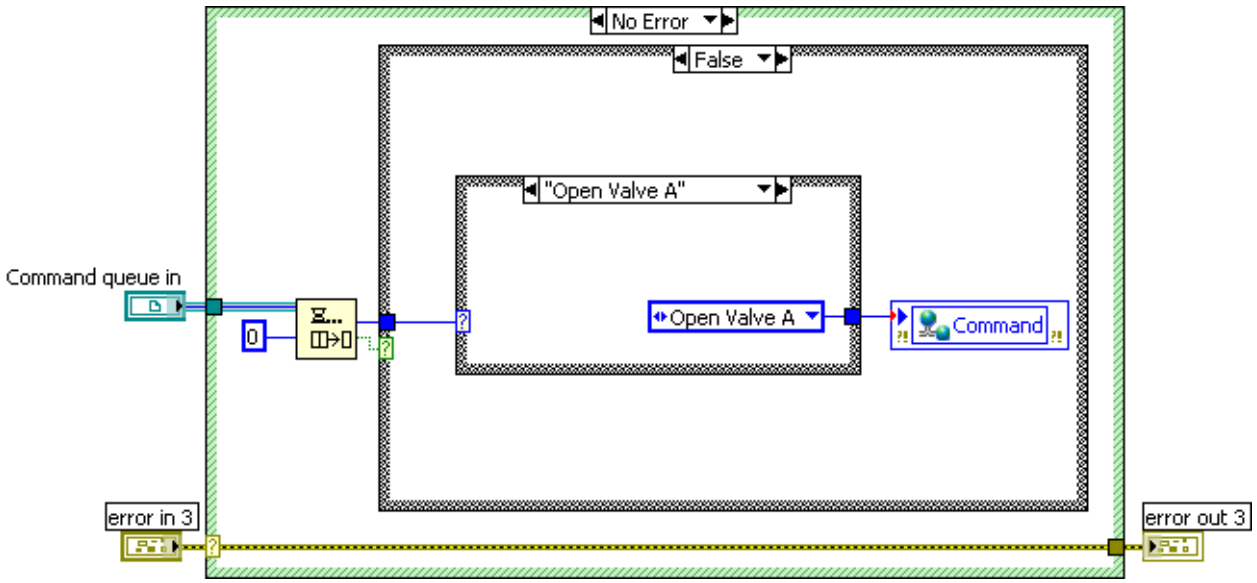


Рисунок 7.13. SubVI Command Handler

5. Сохраните и закройте оба эти subVI.
6. Соедините проводником subVI инициализации и цикл сканирования. В вашем приложении обработчик команд также останавливает цикл при получении команды выключения.

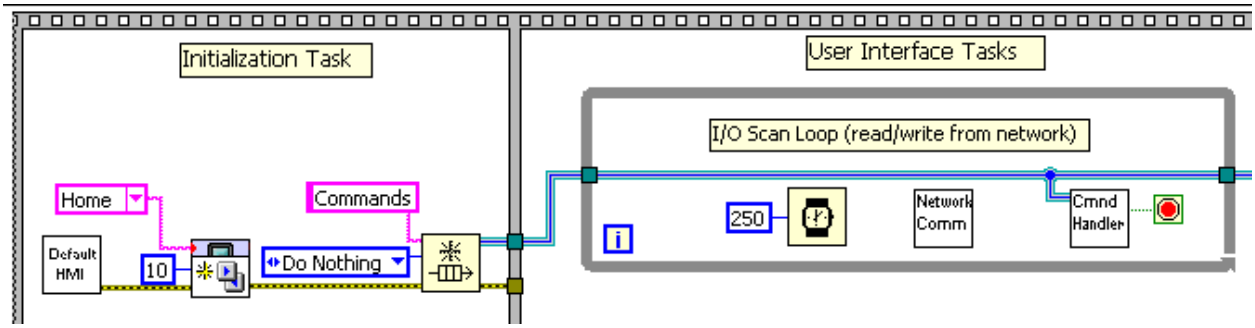


Рисунок 7.14. Цикл сканирования каналов ввода-вывода (I/O Scan Loop) и задача инициализации

Initialization Task – задача инициализации, User Interface Tasks – задачи пользовательского интерфейса, I/O Scan Loop (read/write from network) – цикл сканирования каналов ввода-вывода (чтение/запись по сети)

Цикл навигации

Цикл навигации состоит из механизма навигации и страниц пользовательского интерфейса.

Механизм навигации

Управление страницами графического интерфейса осуществляется с помощью механизма навигации, который выполнен по архитектуре конечного автомата и использует функции API, поставляемые с LabVIEW Touch Panel Module версии 8.6 и выше. Если вы работаете с более ранней версией, то можете загрузить необходимые VI с сайта ni.com.

Механизм навигации проще всего разработать, скопировав программный код из примера.

1. Откройте шаблонный VI, который находится в папке C:\Program Files\National Instruments\LabVIEW 8.6\examples\TouchPanel\navigation\Design Pattern Template.

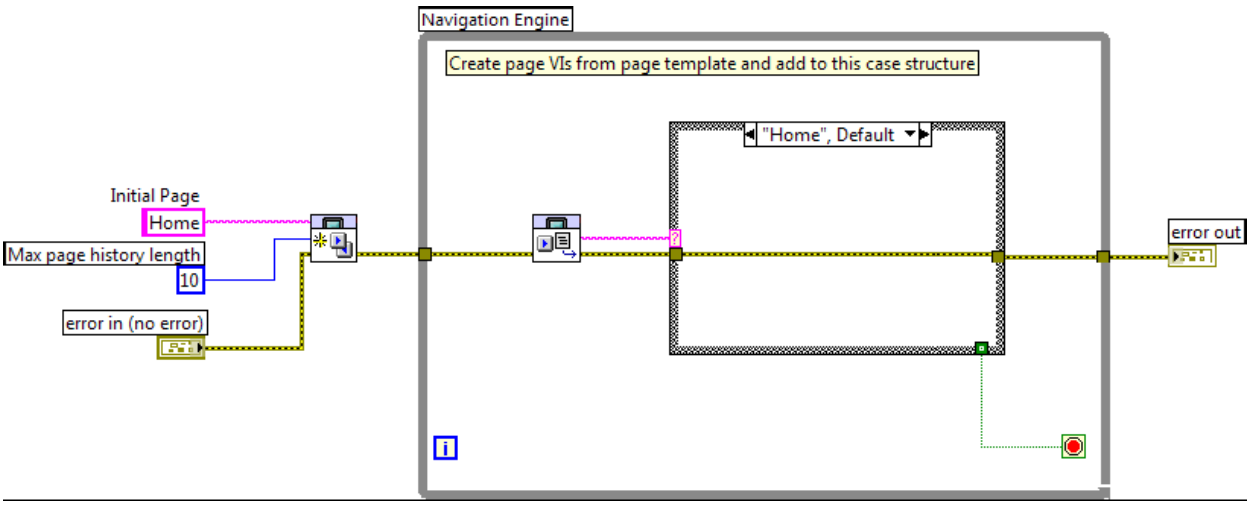


Рисунок 7.15. Шаблонный VI

2. Скопируйте всю блок-диаграмму из шаблона и вставьте ее во вторую последовательность, как цикл, параллельный циклу сканирования.
3. Создайте для каждой страницы пользовательского интерфейса свой вариант выбора в case структуре и дополнительный вариант для выключения.

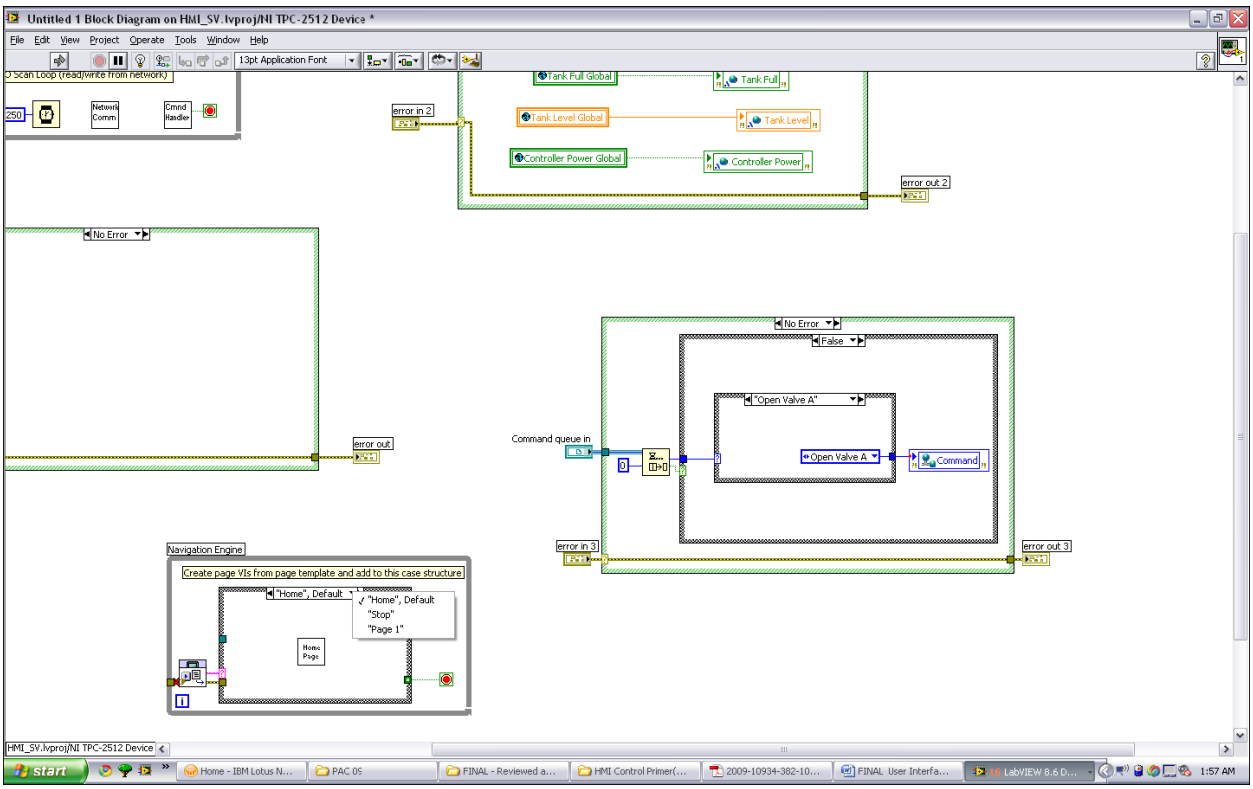


Рисунок 7.16. Добавление вариантов выбора в ваш HMI интерфейс

4. Выберите страницу, которая откроется при первом запуске HMI интерфейса. В этом приложении страница, соответствующая варианту "Home" case-структуры, работает как стартовая страница. Стартовая страница назначается с помощью VI TPC Initialize Navigation.vi перед входом в цикл while.

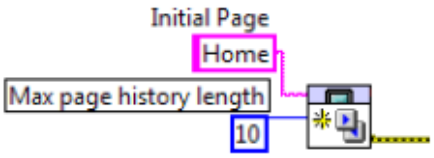


Рисунок 7.17. TPC Initialize Navigation.vi

5. Перейдите в кадр останова и добавьте команду "Shutdown" в очередь команд, которая была создана в цикле инициализации. Эта команда отключает контроллер и закрывает HMI интерфейс.

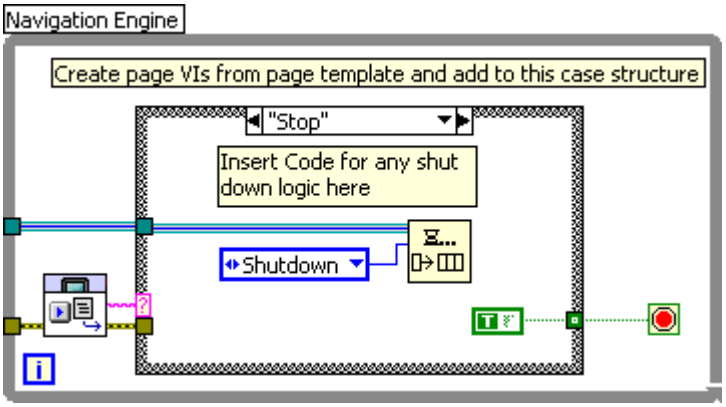


Рисунок 7.18. Добавление команды отключения в очередь команд

На этом разработка механизма навигации завершается. На следующем этапе проектируются страницы пользовательского интерфейса.

Страницы пользовательского интерфейса

При создании страницы HMI интерфейса в первую очередь необходимо установить размеры ее лицевой панели, Поскольку она может быть больше HMI дисплея, важно задать ограничения на предельные размеры элементов пользовательского интерфейса. Например, при использовании компьютера типа NI TPC-2512 HMI размер лицевой панели LabVIEW должен соответствовать разрешению дисплея этого TPC-устройства, которое составляет 800×600 пикселей.

1. Откройте новый VI и установите размеры, соответствующие вашему целевому устройству. В рассматриваемом примере VI настроены для работы с устройством TPC-2512.

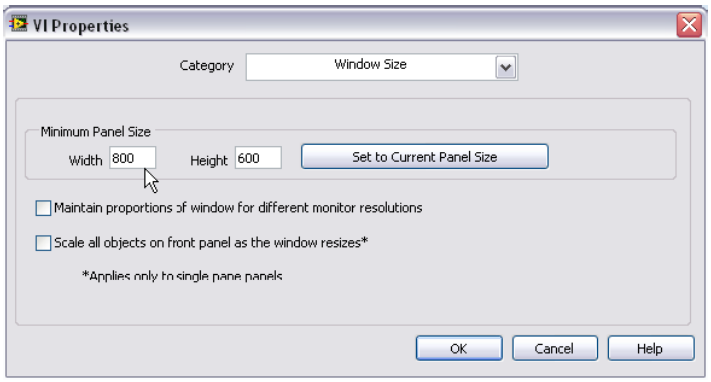


Рисунок 7.19. Установка размеров лицевой панели в соответствии с разрешением сенсорной панели

2. На следующем этапе выбираются элементы пользовательского интерфейса для реализации функций управления с данной страницы. В рассматриваемом примере на странице управления насосом находятся несколько булевских элементов управления и индикаторов, а также кнопки навигации.

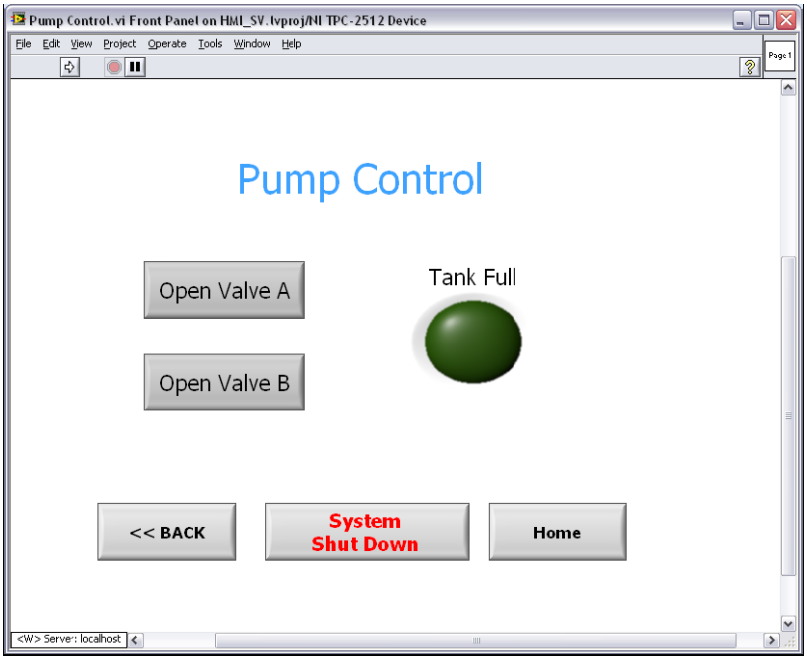


Рисунок 7.20. Пример страницы пользовательского интерфейса

3. К навигационным кнопкам относятся Back (Назад) и Home (В начало). По кнопке Back осуществляется переход к предыдущей странице, на которую вы заходили. По кнопке Home происходит переход на начальную страницу, которая вызывается в кадре "home" case-структуры VI навигации, который был рассмотрен в предыдущем разделе.

4. Поместите на блок-диаграмму страницы пользовательского интерфейса цикл while и структуру обработки событий (event). Подсоедините константу 250 ко входу timeout в верхнем левом углу структуры event. При этом код обработки события timeout будет выполняться каждые 250 мс, если не обнаружатся никакие другие события пользовательского интерфейса.

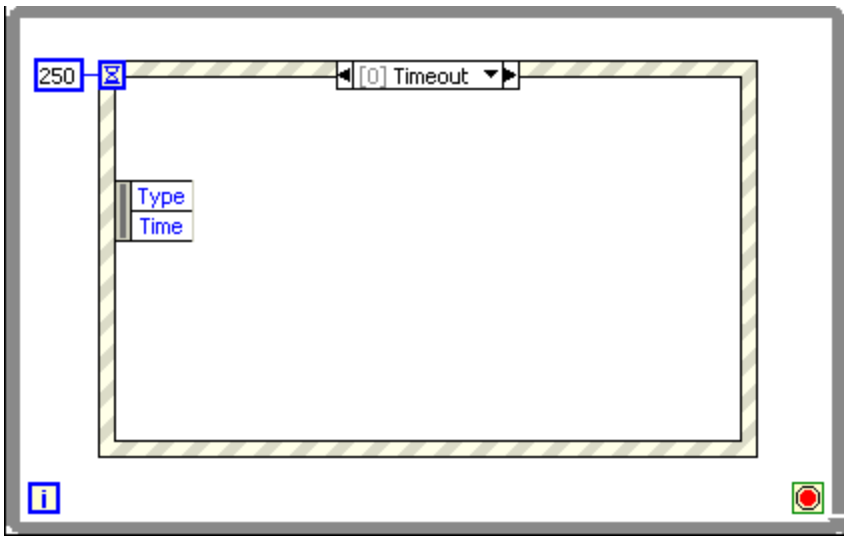


Рисунок 7.21. Обработчик события timeout выполняется каждые 250 мс при отсутствии каких-либо других событий

5. В обработчике тайм-аута (timeout) происходит считывание/запись значений из переменных общего доступа типа Single Process в органы управления и индикаторы этой странице. Поместите в обработчик одну из таких переменных "Tank Full" и соедините ее с соответствующим элементом управления. Обратите внимание на то, что константа "False" соединена с терминалом stop (останов) цикла while. Это гарантирует, что цикл продолжит работать при обработке тайм-аута.

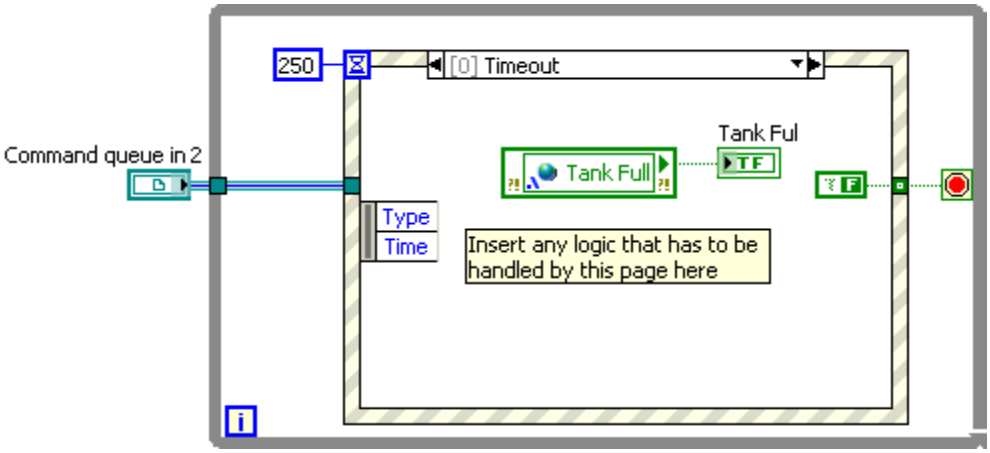


Рисунок 7.22. Константа "False" подсоединена к терминалу останова цикла while, что гарантирует продолжение работы цикла при обработке тайм-аута

6. Теперь команды "Open Valve A" и "Open Valve B" заносятся в очередь команд, созданной в цикле инициализации. Добавьте обработчики событий, которые посылают команду при изменении состояния кнопки. В каждый из обработчиков добавьте соответствующую команду. Убедитесь в том, что состояние кнопки считывается в том же обработчике, в котором она отпускается (если кнопка настроена на фиксацию).

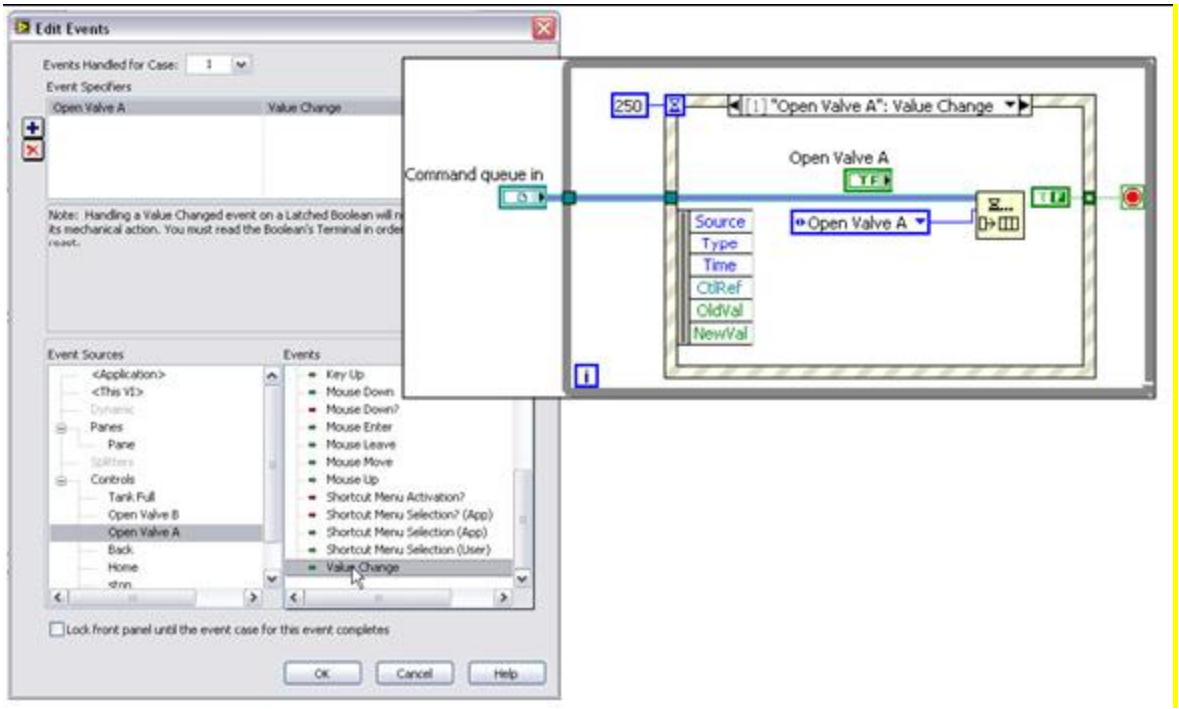


Рисунок 7.23. Команда "Open Valve A" ставится в очередь команд

7. Добавьте обработчики событий для каждой команды или кнопки.
8. В обработчик события "Home" (Переход на начальную страницу) поместите кнопку Home булевского типа и блок TPC Set Navigation Page (Установка страницы навигации для TPC компьютера) из палитры Touch Panel Navigation Palette (Палитра навигации по сенсорной панели) и присоедините к входу этого блока строковую константу с именем Home. Таким способом в качестве следующей страницы устанавливается та, что находится в кадре "Home" механизма навигации. Подсоедините константу "True", поместив ее в обработчик события "Home", к терминалу останова цикла while. Эта константа останавливает цикл, закрывает пользовательский интерфейс и возвращает механизм навигации в состояние, когда оно может открывать следующую страницу, заданную пользователем.

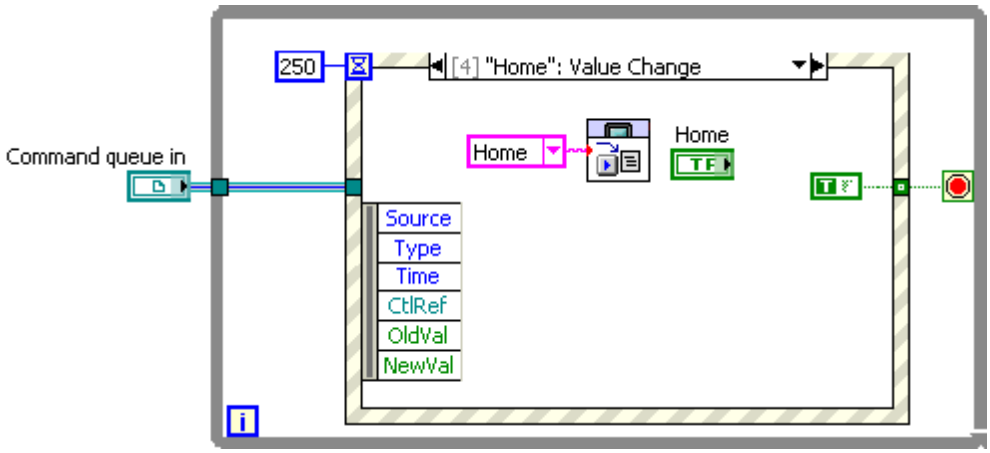


Рисунок 7.24. Константа "True", подсоединенная к терминалу останова, завершает выполнение цикла и возвращает механизме навигации данные о том, что оно может открыть следующую страницу

9. В обработчике нажатия на кнопку "Back" поместите VI TPC Previous Navigation Page.vi, который осуществляет возврат на ранее открытую страницу и остановку VI Pump Control.vi (рисунок 7.20).

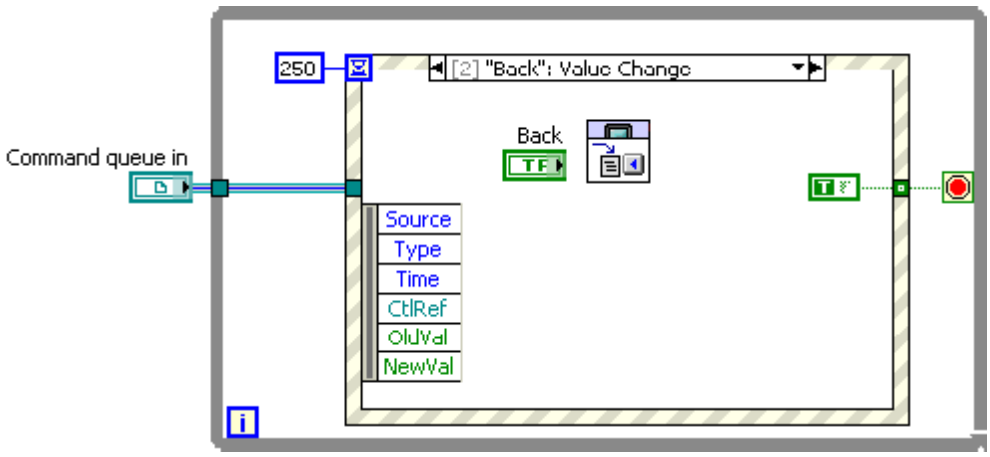


Рисунок 7.25. Возврат на предыдущую страницу с помощью TPC Set Previous Navigation Page.vi

10. Наконец, в обработчике выключения системы присоедините константу перечислительного типа со значением "Stop" ко входу VI TPC Set Next Navigation Page.vi.

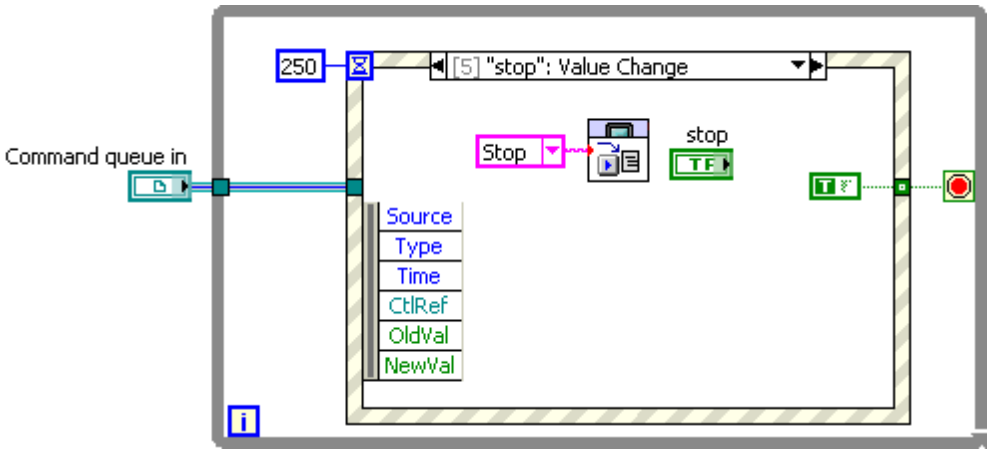


Рисунок 7.26. Присоединения константы перечислительного типа со значением "Stop" ко входу VI TPC Set Navigation Page

11. Сохраните VI Pump Control.vi и поместите его в кадр Page 1 структуры case механизма навигации, чтобы эта страница обязательно открывалась при нажатии кнопки, по которой управления насосом.

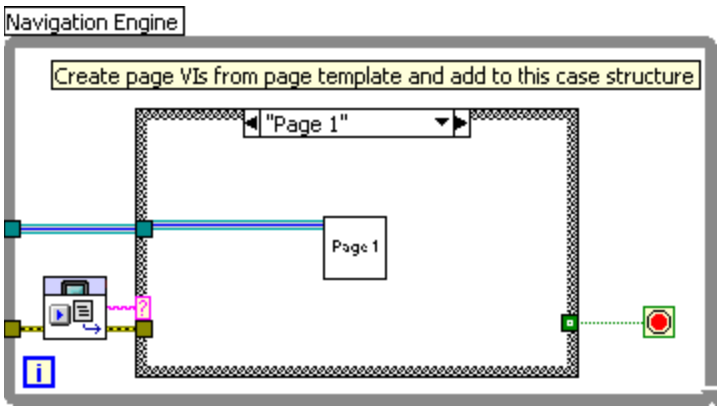


Рисунок 7.27. Обеспечение открытия страницы управления насосом по нажатию кнопки

12. Щелкните правой кнопкой мыши по subVI, выберите команду SubVI Node Setup и проверьте, чтобы были установлены опции, которые обеспечивают появление лицевой панели при вызове и ее исчезновение при закрытии.

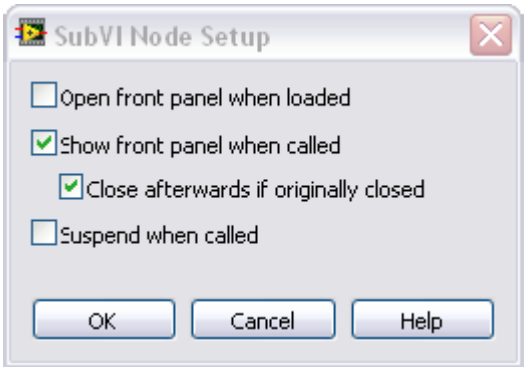


Рисунок 7.28. Проверка опций SubVI

13. Чтобы создать VI для других страниц, повторите приведенные выше действия.
14. На рисунке 7.29 приведен окончательный вариант приложения. Добавьте необходимые логические команды для выключения, например, закрытие очереди.

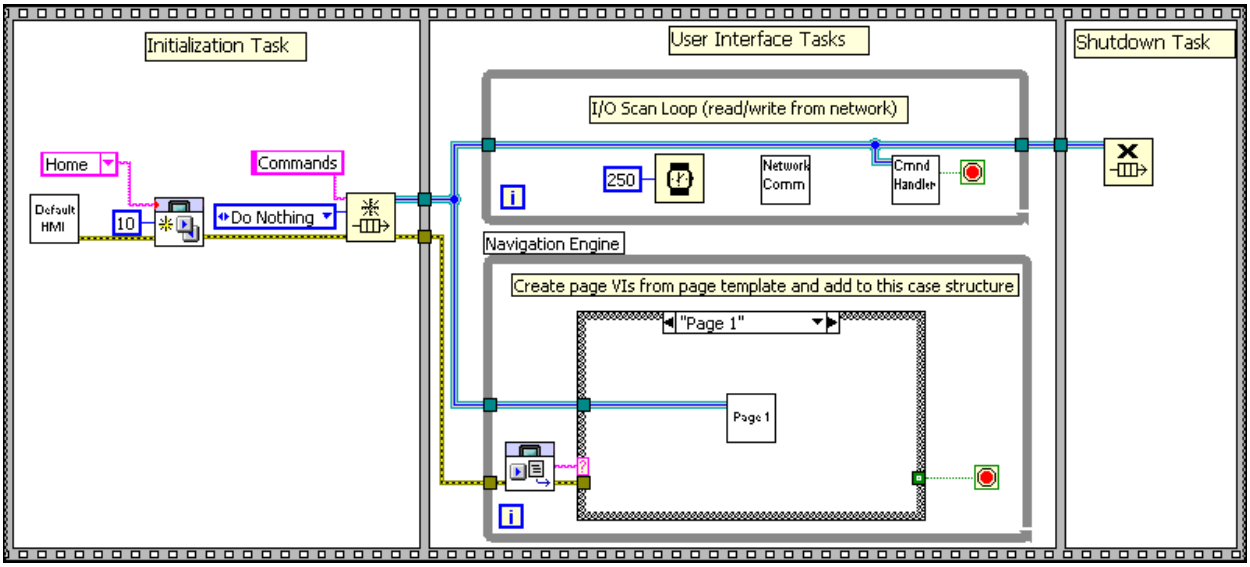


Рисунок 7.29. Цикл сканирования и инициализация механизма навигации в структуре последовательности

Начало работы – редактирование примера

В данном разделе приведен пример кода LabVIEW

Самый простой способ начать работать с HMI архитектурой – разработка путем модификации уже существующего примера. Для модификации предыдущего примера выполните следующие основные действия:

Шаг 1. Редактирование табличной памяти

1. Отредактируйте библиотеку IO_Table.lvlib, которая содержит переменные общего доступа типа Single Process. В этих переменных хранятся данные процесса, которые вам нужно записывать или считывать из страниц пользовательского интерфейса.

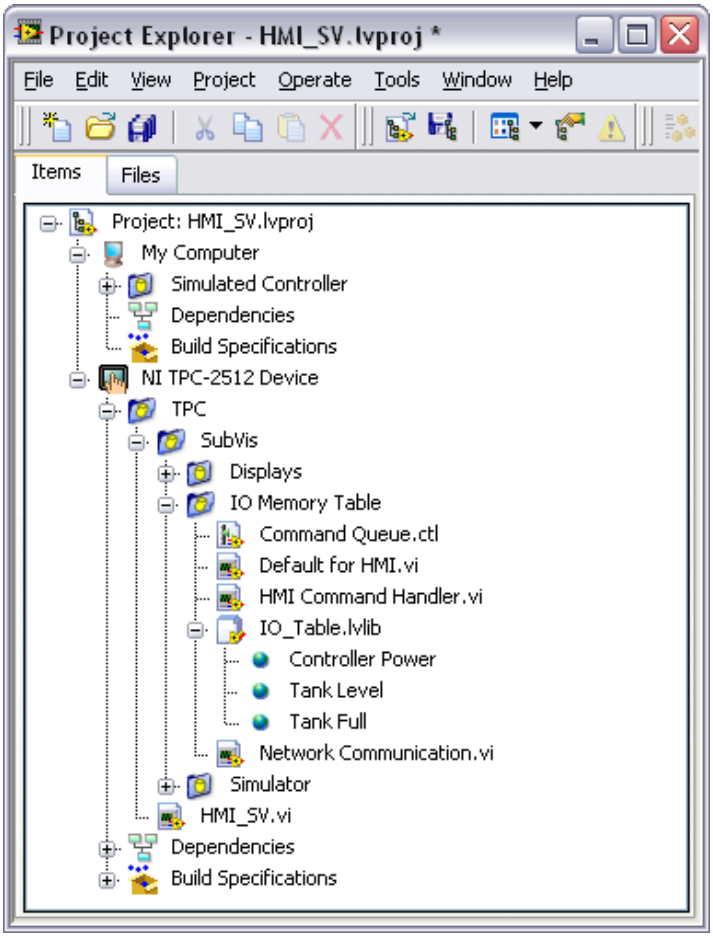


Рисунок 7.30. Добавление переменных общего доступа типа Single Process в библиотеку IO-Tabel.lvlib

2. Отредактируйте VI Default HMI.vi таким образом, чтобы записать начальные значения в ваши органы управления и индикаторы.

Шаг 2. Редактирование списка команд

1. Отредактируйте список команд таким образом, чтобы там были только команды, которые необходимо передавать в приложение.

Шаг 3. Редактирование цикла сканирования каналов ввода-вывода

1. Отредактируйте цикл сканирования каналов ввода-вывода так, чтобы данные записывались или читались из соответствующих сетевых переменных и передавались в табличную память (переменные общего доступа типа Single Process).
2. Отредактируйте обработчик команд таким образом, чтобы команды вашего интерфейса пользователя считывались и посылались в виде соответствующих сетевых команд в переменную команд, обслуживаемую в CompactRIO.

Шаг 4. Редактирование цикла навигации

1. Добавьте дополнительные кадры в механизм навигации, чтобы добавить дополнительные страницы.
2. Для каждой страницы интерфейса создайте новые VI и добавьте их в соответствующие кадры.

РАЗДЕЛ 8
Развертывание и тиражирование приложений

Развертывание приложения

Весь процесс разработки LabVIEW приложений для целевых устройств реального времени и устройств с сенсорными панелями, происходит на персональном компьютере, работающем под управлением ОС Windows. Чтобы запустить код для исполнения в этих устройствах, необходимо развернуть приложения. Контроллеры реального времени и устройства с сенсорными панелями, как и персональные компьютеры, имеют энергозависимую память (RAM) и энергонезависимую память (жесткий диск). Вы можете выбрать, где развертывать приложение.

Развертывание в энергозависимой памяти

Если вы развертываете приложение в энергозависимой памяти, оно не сохраняется после отключения и повторного включения питания. Такой вариант развертывания является полезным в процессе разработки приложения и тестирования спроектированной вами программы.

Развертывание в энергонезависимую память

Если вы развертываете приложение в энергонезависимой памяти устройства, после отключения и включения питания оно остается в памяти. Приложения, которые хранятся в энергонезависимой памяти, можно настроить таким образом, чтобы они автоматически запускались при загрузке устройства. Этот вариант полезен после завершения проектирования и проверки правильности работы приложения, после чего можно создать автономную встраиваемую систему.

Развертывание приложений на платформу CompactRIO

Развертывание VI LabVIEW в энергозависимую память

Когда вы развертываете приложение в энергозависимую память контроллера Compact RIO, LabVIEW собирает все необходимые файлы и загружает их через Ethernet в контроллер. Чтобы развернуть приложение, необходимо выполнить следующие действия:

- Назначить в LabVIEW целевым устройством контроллер CompactRIO
- Открыть VI, работающий в контроллере
- Нажать на кнопку "Run" (запуск)

LabVIEW проверяет, сохранены ли VI и все subVI, развертывает код в энергозависимую память контроллера CompactRIO и запускает это встраиваемое приложение в контроллере.

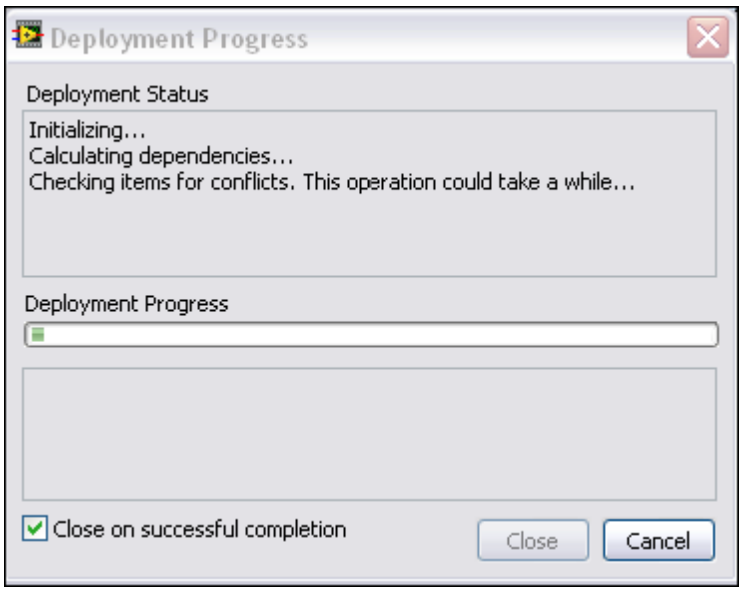


Рисунок 8.1. Развертывание приложения LabVIEW в энергозависимую память контроллера

Deployment Progress – процесс развертывания, Deployment Status – состояние процесса развертывания, Initializing – инициализация, Calculating dependences – вычисление зависимостей, Checking items for conflict – проверка на отсутствие конфликтных ситуаций, Close on successful completion – закрыть при успешном завершении

Развертывание VI LabVIEW в энергонезависимую память

После завершения разработки и отладки приложения, скорее всего, вы захотите развернуть вашу программу в энергонезависимой памяти контроллера. В этом случае она сохраняется при отключении/включении питания, а система настраивается таким образом, чтобы приложение запускалось при включении контроллера. Чтобы развернуть приложение в энергонезависимой памяти, в первую очередь необходимо преобразовать VI в исполняемый файл.

Построение исполняемого приложения на основе VI

Проект, созданный в LabVIEW, предоставляет возможность построения на основе VI исполняемого приложения реального времени. Для этого в навигаторе проекта LabVIEW (LabVIEW Project Explorer) создается спецификация построения исполняемого файла под целевое устройство реального времени. Щелчком правой кнопки мыши по элементу Build Specifications (Построить спецификацию) открывается меню выбора цели: Real-Time Application (исполняемое приложение реального времени), Source Distribution (распространение исходных файлов), Zip File (Zip-архив) и т.д.

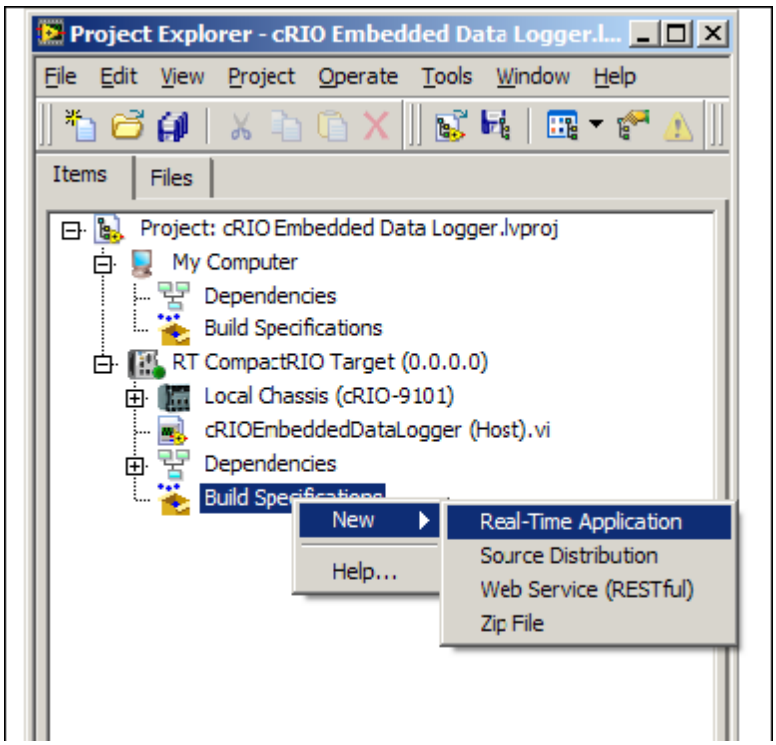


Рисунок 8.2. Создание спецификации построения приложения реального времени

После того, как вы выбрали цель **Real-Time Application**, открывается диалоговое окно свойств, в котором для построения приложения реального времени наиболее часто используются два основных раздела (Category): Information (Информация) и Source Files (Исходные файлы). Остальные разделы (Destinations, Source File Settings, Advanced, Additional Exclusions) при создании приложений реального времени применяются редко.

Раздел Information содержит следующие поля: Build specification name (имя создаваемой спецификации), Executable filename (имя исполняемого файла) и папки назначения (Destination directory) для целевого устройства реального времени и персонального компьютера. Вы можете изменить поля Build specification name и Local destination directory с учетом организации файлов (путей и имен) на вашем компьютере. Как правило, вам не нужно изменять поля Target filename и Target destination directory (папку и имя файла для целевого устройства).

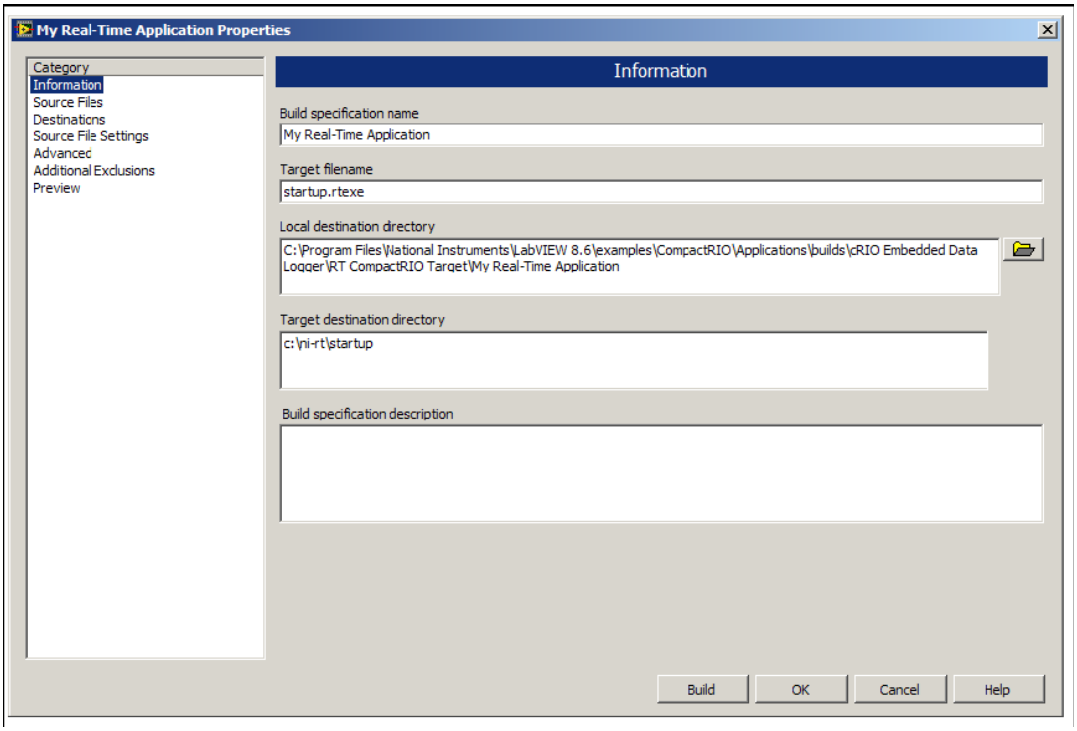


Рисунок 8.3. Раздел Information свойств приложения реального времени

Раздел Source Files используется для назначения стартового VI, а также подключения дополнительных VI или файлов поддержки. Вам необходимо выбрать среди файлов проекта (Project Files) VI самого верхнего уровня и определить его в качестве Startup VI. В большинстве приложений в качестве стартового применяется какой-то один VI. Вам не нужно подключать файл библиотеки проекта *.lvlib или делать subVI стартовыми или всегда включаемыми (в поле Always Included), кроме случая, когда они динамически вызываются из вашего приложения.

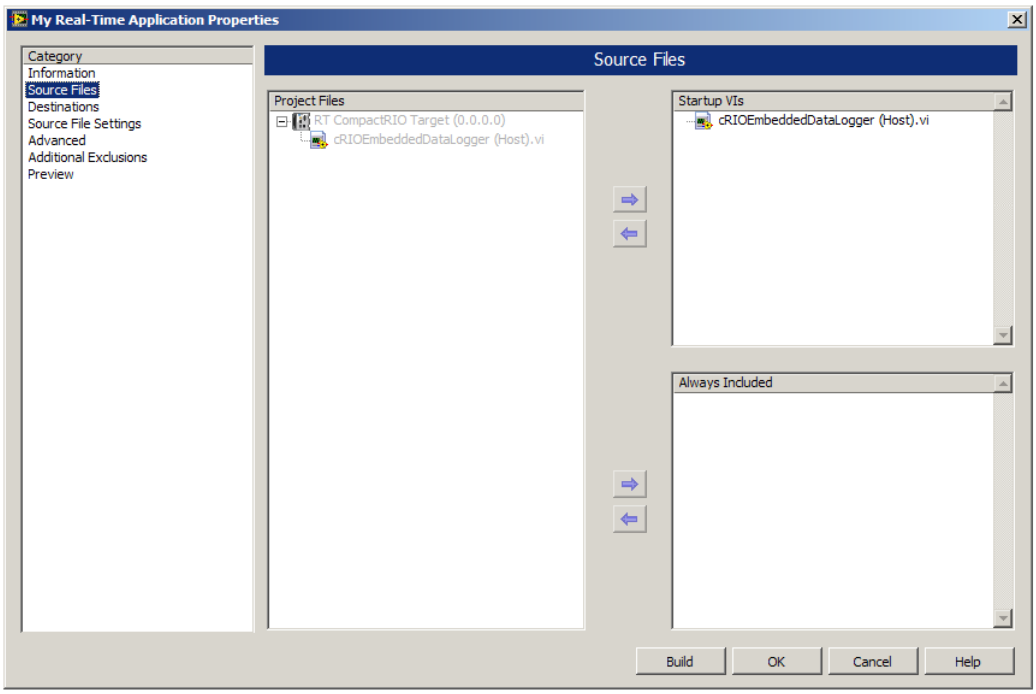


Рисунок 8.4. Раздел Source Files в окне свойств приложения реального времени

(В данном примере в качестве стартового выбран VI cRIOEmbeddedDadtLogger(Host).vi)

После заполнения всех требуемых полей следует щелкнуть по кнопке OK, чтобы сохранить спецификации построения. Вы можете также сразу запустить построитель приложения щелчком по кнопке **Build.** Кроме того, вы можете щелкнуть правой кнопкой мыши по сохраненной спецификации построения и выбрать команду **Build.**

При построении приложения создается исполняемый файл, который сохраняется на жестком диске в локальной папке назначения на вашем компьютере.

Настройка запуска исполняемого приложения реального времени при включении контроллера

Когда исполняемое приложение построено, вы можете настроить его таким образом, чтобы оно автоматически запускалось при загрузке контроллера. Для этого нужно щелкнуть правой кнопкой мыши по элементу Real-Time Application, который находится ниже элемента Build Specifications, и выбрать команду Set as startup (Сделать запускаемым). Когда вы развернете исполняемое приложение в контроллер реального времени, контроллер также настраивается на автоматический запуск приложения при его включении или перезагрузке контроллера. Чтобы запретить автоматический запуск, вы можете выбрать команду Unset as Startup.

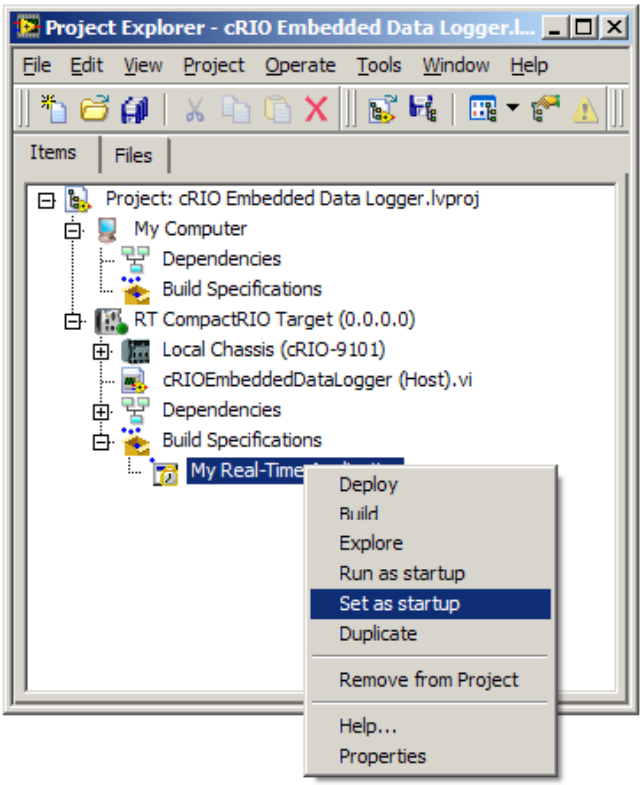


Рисунок 8.5. Конфигурирование построения спецификации на запуск приложения при загрузке контроллера

Развертывание исполняемого приложения реального времени в энергонезависимую память системы CompactRIO

После того, как вы сконфигурировали и создали исполняемое приложение, необходимо скопировать его вместе со вспомогательными файлами в энергонезависимую память системы CompactRIO и настроить контроллер таким образом, чтобы приложение запускалось при его включении. Чтобы скопировать файлы и сконфигурировать контроллер, щелкните правой кнопкой мыши по элементу Real-Time Application и выберите команду Deploy (Развернуть). LabVIEW копирует исполняемые файлы в энергонезависимую память контроллера и скрытно модифицирует файл ni-rt.ini, чтобы приложение запускалось при включении контроллера. Если вы заново построите приложение или измените его свойства (например, отменяете запуск при включении), необходимо заново развернуть приложение, чтобы сделанные изменения вступили в силу в целевом устройстве реального времени.

Возможно, в какой-то момент вам захочется удалить приложение из целевого устройства реального времени. Проще всего это можно сделать, удалив исполняемый файл посредством доступа к нему через FTP протокол. Если вы использовали настройки по умолчанию, этот файл, который имеет имя, взятое из поля Target filename раздела Information и расширение .rtexe, находится в папке NI-RT\Startup.

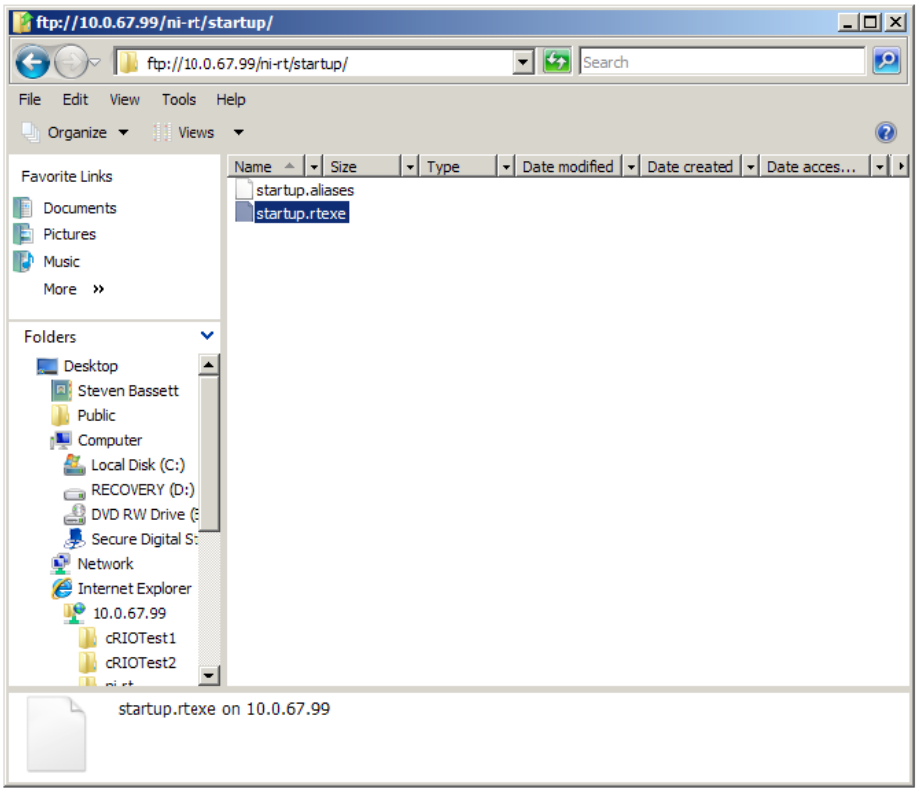


Рисунок 8.6. Удаление файла startup.rtexe из контроллера CompactRIO

Развертывание приложений в устройстве с сенсорной панелью

Конфигурирование соединения с устройством с сенсорной панелью

Несмотря на то, что построенные приложения можно вручную копировать в устройство с сенсорной панелью, рекомендуется использовать Ethernet и разрешать проекту LabVIEW автоматически загружать приложение. Все устройства с сенсорными панелями производства National Instruments поставляются с утилитой NI TPC Service, которая позволяет проекту LabVIEW загружать программный код непосредственно через Ethernet. Чтобы сконфигурировать соединение, щелкните правой кнопкой мыши по устройству с сенсорной панелью в проекте LabVIEW и выберите пункт Properties. В разделе General (Общие настройки) в качестве типа соединения в списке Connection выберите NI TPC Service и введите IP адрес устройства с сенсорной панелью в поле Touch Panel device IP address. Протестируйте соединение, чтобы убедиться в его работоспособности.

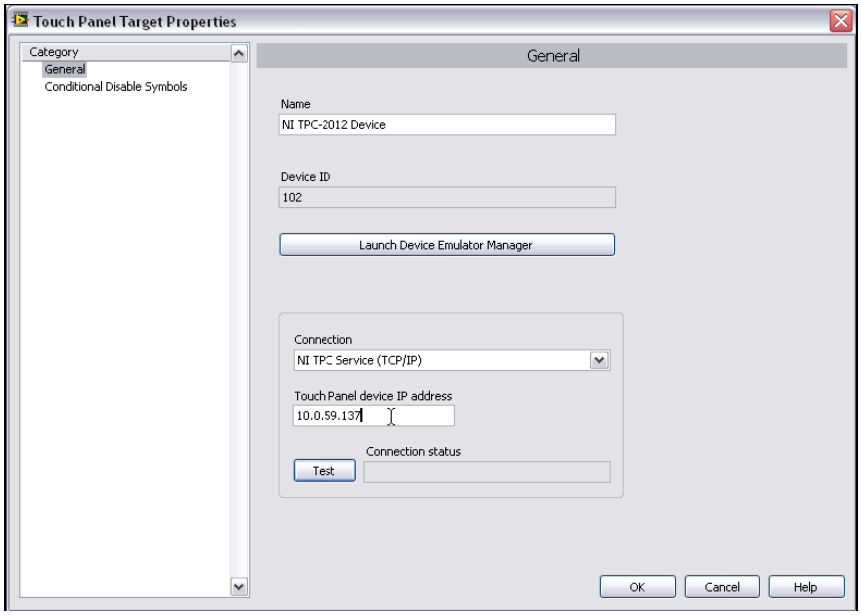


Рисунок 8.7. Соединение с устройством с сенсорной панелью через Ethernet с помощью NI TPC Service

IP адрес устройства с сенсорной панелью можно узнать, если перейти в командную строку и набрать команду ipconfig. Для этого зайдите в меню Start и выберите команду Run... Затем во всплывающем окне введите вышеупомянутую команду.

Развертывание LabVIEW VI в энергозависимую или энергонезависимую память

Развертывание приложения в устройстве с сенсорной панелью под Windows XP Embedded и под Windows CE происходит почти одинаково. Единственным отличием устройства под XP Embedded является то, что развертывать приложение можно только в энергонезависимую память. В то же время, устройство под Windows CE позволяет, в зависимости от выбора папки назначения, развертывать приложение либо в энергозависимую, либо в энергонезависимую память. Чтобы запустить VI, развернутый в энергозависимой или энергонезависимой памяти, необходимо в первую очередь создать исполняемый файл.

Построение исполняемого файла на основе VI для устройства с сенсорной панелью под XP Embedded

Проект LabVIEW дает возможность строить исполняемые приложения для устройств с сенсорной панелью на основе VI. Для этого в навигаторе проекта необходимо создать спецификацию построителя приложения под нужное устройство с сенсорной панелью. Щелкните правой кнопкой мыши по пункту Build Specifications, и у вас появляется возможность выбрать цель построения: Touch Panel Application (Приложение для устройства с сенсорной панелью), Source Distribution (Распространение исходных файлов), Zip File (Zip-архив) и т.д.

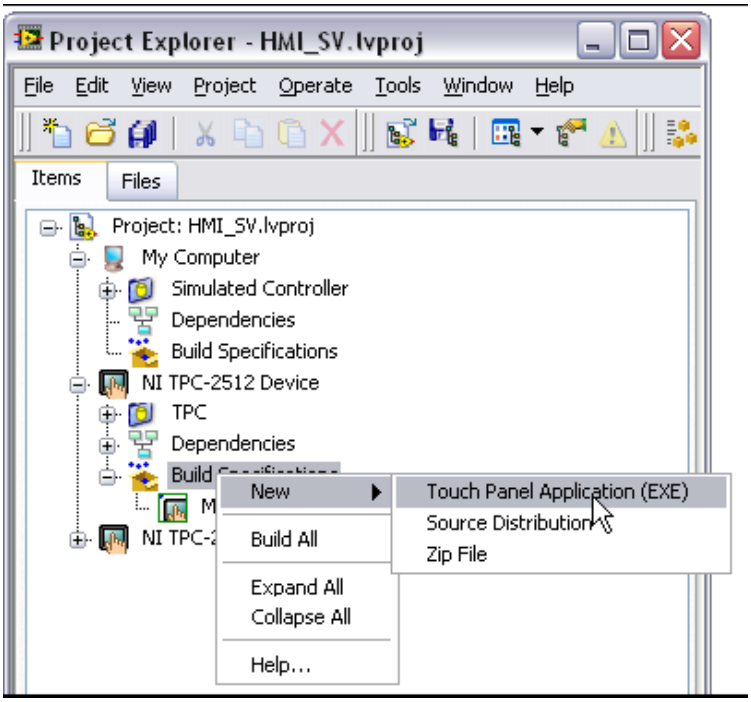


Рисунок 8.8. Создание приложения для устройства с сенсорной панелью с помощью проекта LabVIEW

После того, как вы выбрали цель **Touch Panel Application**, откроется диалоговое окно свойств. К двум основным разделам, используемым при построении приложений для устройств с сенсорной панелью, относятся Information и Source Files. Другие разделы при построении таких приложений используются редко.

Раздел Information содержит имя построения спецификации в поле Build specification name, имя исполняемого файла в поле Target filename, полное имя папки назначения на платформе с сенсорной панелью в поле Target destination directory, полное имя папки назначения на персональном компьютере в поле Local Destination Directory. Вы можете изменять поля Build specification name и Local Destination Directory с учетом организации файлов на вашем компьютере. Как правило, у вас нет необходимости изменять имя файла и папку назначения (Target filename и Target destination directory) для целевого устройства.

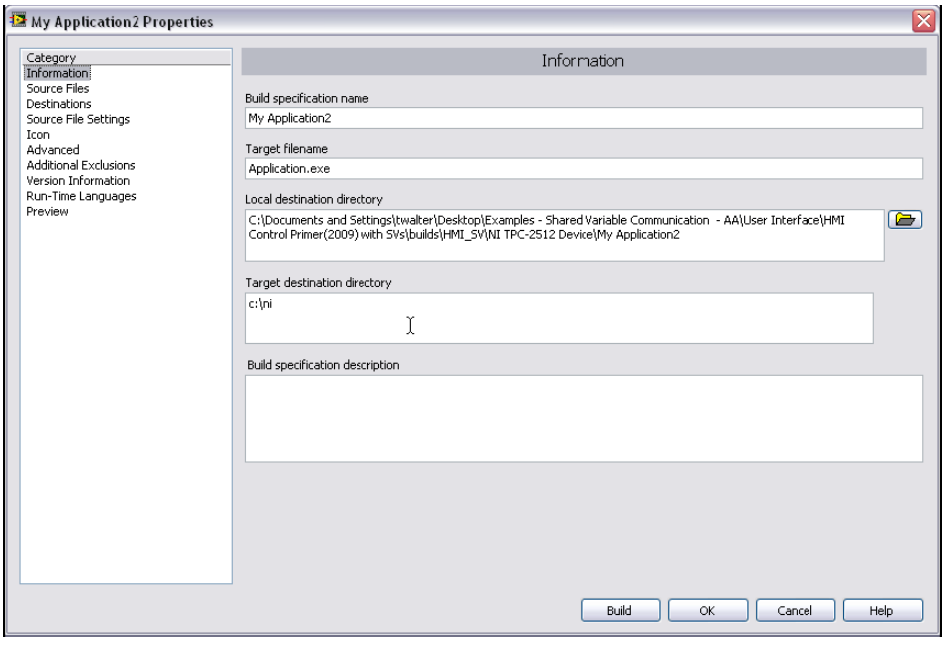


Рисунок 8.9. Раздел Information в окне свойств приложения для устройства с сенсорной панелью

Раздел Source Files используется для назначения стартового VI и включения дополнительных VI и файлов поддержки. Среди файлов проекта необходимо выбрать VI самого верхнего уровня и установить его в качестве стартового. Для большинства приложений в качестве стартового выбирается какой-либо один VI. Вам не нужно подключать файл библиотеки проекта *.lvlib или устанавливать subVI в качестве стартовых (в поле Startup VIs) или всегда включаемых (в поле Always Included) кроме случая, когда они вызываются динамически из вашего приложения.

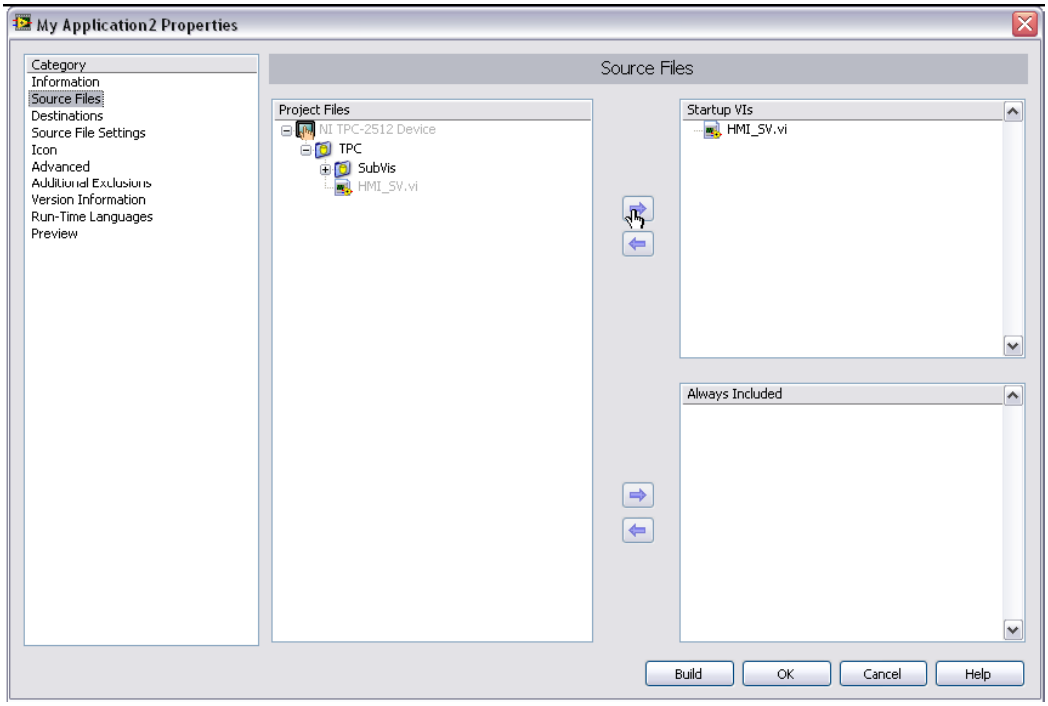


Рисунок 8.10. Раздел Source Files в окне свойств приложения на платформе с сенсорной панелью (В данном примере в качестве стартового выбран HMI_SV.vi)

После того как все требуемые поля разделов заполнены, можно щелкнуть по кнопке OK, чтобы сохранить спецификацию построителя или сразу построить приложение щелчком по кнопке Build. Вы можете также щелкнуть правой кнопкой мыши по сохраненной спецификации построителя и выбрать команду **Build.**

В процессе построения приложения создается исполняемый файл, который сохраняется в указанной папке назначения на жестком диске вашего компьютера.

Построение исполняемого файла на основе VI для устройства с сенсорной панелью под Windows CE

Проект LabVIEW предоставляет возможность построить исполняемое приложение для устройства с сенсорной панелью на основе VI. Для этого в навигаторе проекта создается спецификация построителя под соответствующее целевое устройство с сенсорной панелью. Щелчком правой кнопки мыши по пункту Build Specifications выбирается цель построения: Touch Panel Application (приложение с сенсорной панелью), Source Distribution (Распространение исходных файлов), Zip-file (Zip-архив) и т.д.

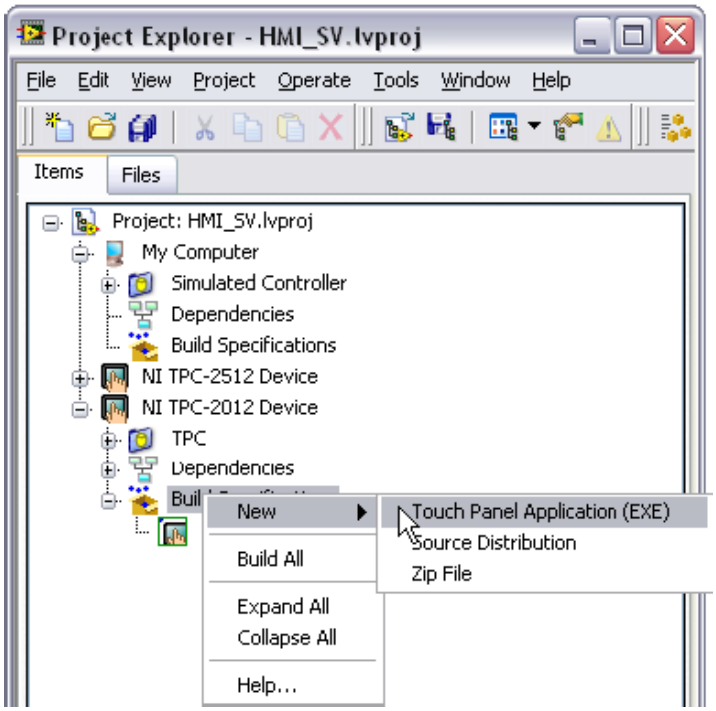


Рисунок 8.11. Создание приложения для устройства с сенсорной панелью с помощью проекта LabVIEW

После того, как вы выбрали цель **Touch Panel Application**, появится диалоговое окно свойств, в котором есть три основных раздела, чаще всего используемые при построении приложений для устройств с сенсорной панелью под Windows CE: Application Information (Сведения о приложении), Source Files (Исходные файлы) и Machine Aliases (Псевдонимы компьютеров). Остальные разделы при построении приложений для устройств с сенсорной панелью под Windows CE редко подвергаются изменениям.

Раздел Application Information содержит следующие поля: Build specification name (имя спецификации построителя), Target filename (имя исполняемого файла), Destination directory (полное имя папки назначения на персональном компьютере) и Remote path for target application (полное имя папки назначения для удаленного целевого устройства с сенсорной панелью). Вы можете изменять поля Build specification name и Destination directory с учетом организации файлов на вашем компьютере. Как правило, у вас нет необходимости изменять поля Target filename и Remote path for target application. Имя папки назначения определяет, откуда будет запускаться развертываемый исполняемый файл: из энергозависимой или из энергонезависимой памяти.

Назначение папок на устройстве под Windows CE:

- \My Documents – энергозависимая память. Если вы установите туда исполняемый файл, то при отключении/включении питания он там не сохранится.
- \HardDisk – энергонезависимая память. Если вы хотите, чтобы приложение сохранялось в памяти устройства после отключения/включения питания, в поле Remote path for target application необходимо ввести имя папки, которая начинается с \HardDisk, например, \HardDisk\Documents and Settings

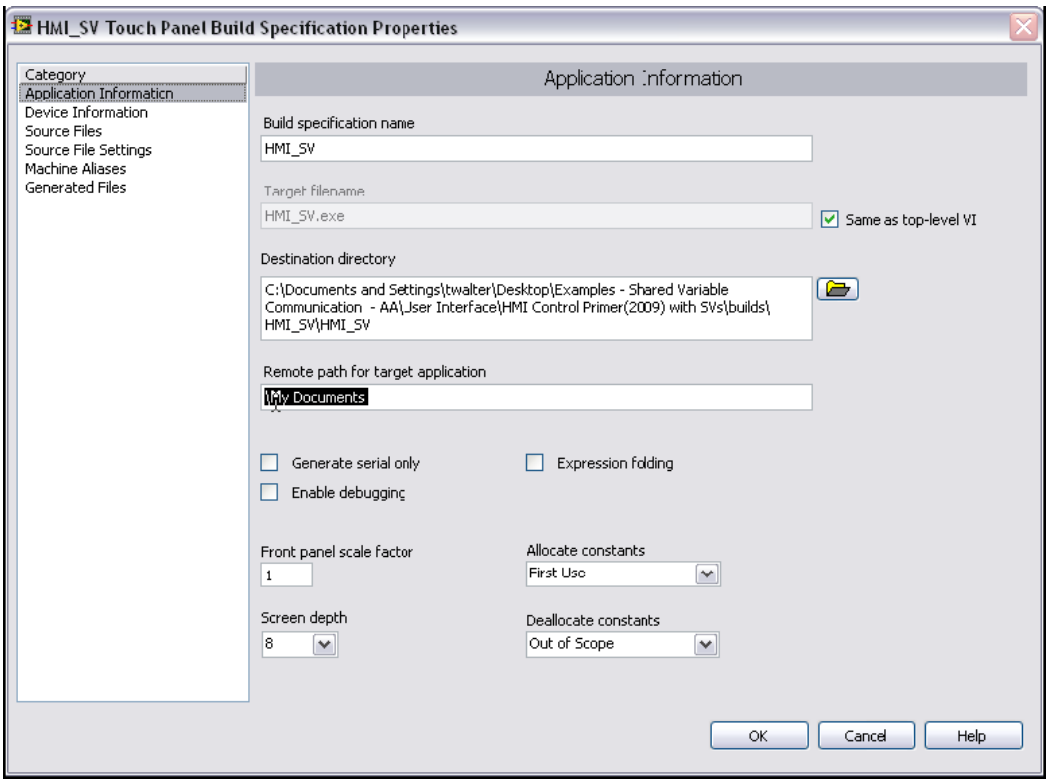


Рисунок 8.12. Раздел Information в окне свойств приложения для устройства с сенсорной панелью

Раздел Source Files используется для назначения стартового VI и включения дополнительных VI и файлов поддержки. Вам необходимо выбрать VI самого верхнего уровня вашего проекта. Этот VI и является запускаемым при включении устройства. В приложениях для устройств с сенсорной панелью под Windows CE вы можете выбрать только один VI в качестве стартового. Вам не нужно подключать файл библиотеки проекта *.lvlib или помещать subVI в поле Always included (Всегда включены).

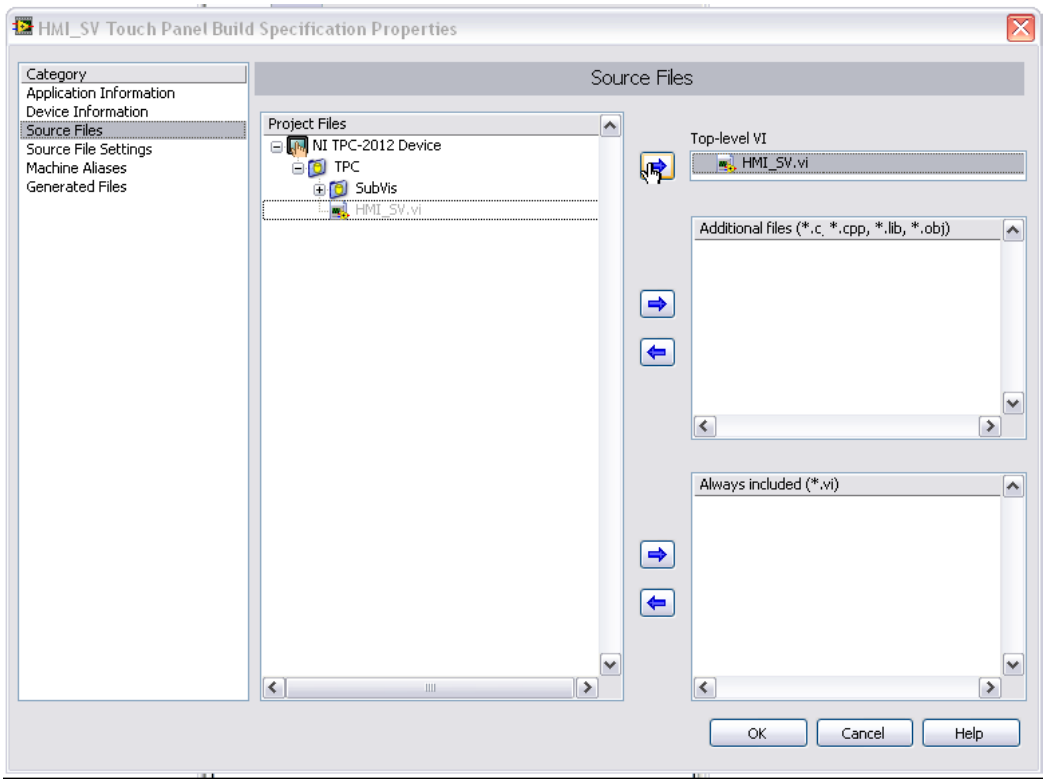


Рисунок 8.13. Раздел Source Files в окне Touch Panel Application Properties

(В этом примере в качестве стартового выбран HMI_SV.vi)

Раздел Machine Aliases предназначен для развертывания файла псевдонимов, который требуется при использовании сетевых переменных общего доступа для связи с любыми устройствами. Проверьте, установлен ли флажок "Deploy alias file" (Развертывание файла псевдонимов). Список псевдонимов должен включать в себя серверы с сетевыми сетевых переменных общего доступа и их IP адреса (обычно это контроллеры CompactRIO или персональные компьютеры под Windows). Более подробная информация о файлах псевдонимов и развертывании приложений, использующих сетевые переменные общего доступа, приведена в разделе по развертыванию в настоящем документе.

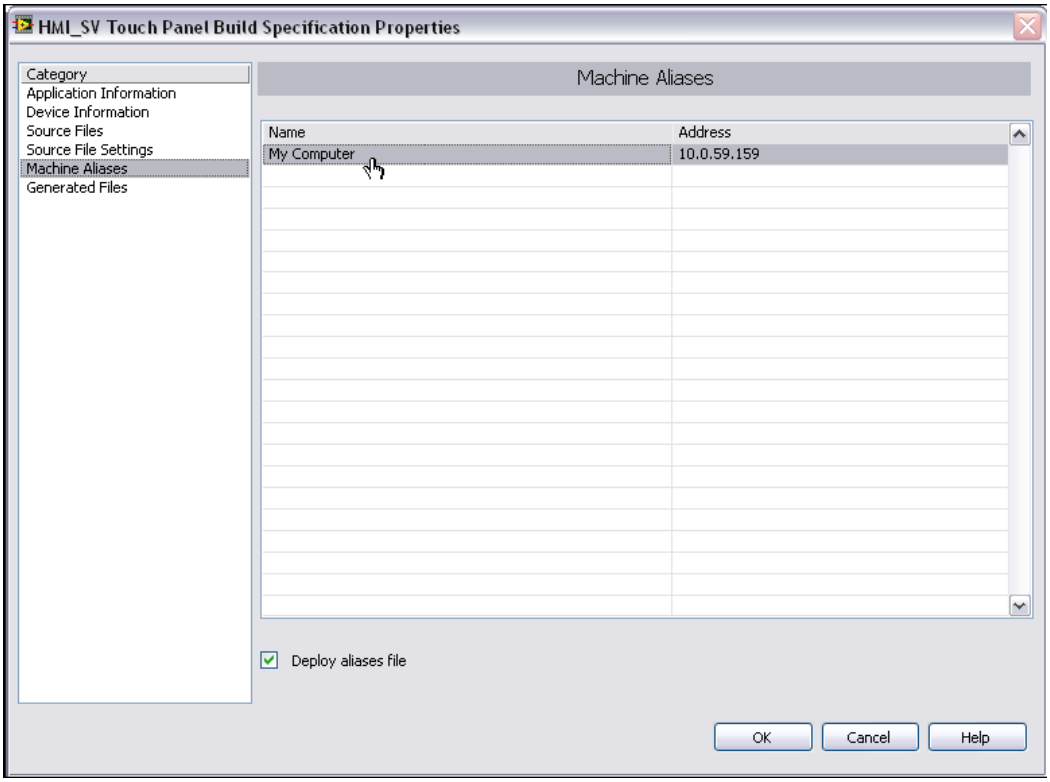


Рисунок 8.14. Раздел Machine Aliases в окне свойств приложения

(Обязательно проверьте, установлен ли флажок "Deploy alias file", если используются сетевые переменные общего доступа)

После того, как в соответствующие поля вышеупомянутых разделов введены необходимые данные, вы можете щелкнуть по кнопке OK, чтобы сохранить спецификацию построения приложения или сразу построить приложение, щелкнув по кнопке **Build.** Вы можете также щелкнуть правой кнопкой по сохраненной спецификации и выбрать команду **Build.**

Когда строится приложение, создается исполняемый файл, который сохраняется на жестком диске вашего компьютера в локальной папке назначения.

Развертывание исполняемого приложения в устройстве с сенсорной панелью под Windows CE или Windows XP Embedded Target

После конфигурирования и построения исполняемого файла необходимо его скопировать вместе со файлами поддержки в память устройства с сенсорной панелью. Для этого щелкните правой кнопкой мыши по элементу Touch Panel Application и выберите команду Deploy. LabVIEW выполнит копирование исполняемых файлов. Если вы построите приложение заново, то для того, чтобы изменения вступили в силу на платформе с сенсорной панелью, необходимо снова выполнить процедуру развертывания.

Кнопка запуска Run

Если вы щелкните мышью по кнопке Run запуска VI, предназначенного для устройства с сенсорной панелью, LabVIEW поможет вам создать спецификацию построителя (если она не создана) и развернет код программы в устройстве с сенсорной панелью.

Настройка приложения, исполняемого в устройстве с сенсорной панелью, на запуск при включении

После развертывания приложения в устройстве с сенсорной панелью, вы можете настроить исполняемый файл так, чтобы он автоматически запускался при загрузке устройства. Поскольку вы работаете под Windows, это можно сделать с помощью стандартных средств Windows. При работе под Windows XP Embedded необходимо скопировать исполняемый файл и вставить его ярлык в папку Startup меню Start. При работе под Windows CE необходимо перейти в папку STARTUP на жестком диске и модифицировать файл startup.ini таким образом, чтобы там указывался путь к исполняемому файлу (\HardDisk\Documents and Settings\HMI_SV.exe). Сконфигурировать программу, чтобы она запускалась при загрузке, можно также на закладке Misc (Разное) утилиты Configuration Utility (**Start»Programs»Utilities»Configuration Utilities**). Эта утилита модифицирует файл startup.ini.

Развертывание приложений, которые используют сетевые переменные общего доступа

Начальные сведения о сетевых переменных общего доступа

Сетевая переменная общего доступа – это объект программный, который существует в сети и позволяет обмениваться информацией между программами, приложениями, удаленными компьютерами и аппаратными средствами.

Ниже приведены сведения о трех компонентах, которые делают сетевую переменную работоспособной в LabVIEW.

Узлы сетевых переменных

С помощью узла сетевой переменной на блок-диаграмме представляются операции чтения и записи через сеть. Каждый такой узел соответствует программному объекту в сети (настоящей сетевой переменной), управляемому механизмом продвижения сетевых переменных общего доступа Shared Variable Engine. На рисунке 8.15 приведен узел сетевой переменной, сетевой путь к ней и соответствующий элемент в дереве проекта.

Рисунок 8.15. Узел сетевой переменной и сетевой путь к нему

Механизм продвижения переменных общего доступа (Shared Variable Engine)

Механизм продвижения переменных общего доступа (Shared Variable Engine) – это программный компонент, который обслуживает данные, публикуемые в сети Ethernet. Shared Variable Engine может работать на целевых устройствах реального времени или персональных компьютерах под Windows, в зависимости от того, где обслуживаются переменные общего доступа. В ОС Windows Shared Variable Engine является службой, которая запускается при загрузке самой ОС. На целевых устройствах реального времени Shared Variable Engine является драйвером, который запускается при загрузке системы.

Когда Shared Variable Engine стартует, он читает данные из энергонезависимой памяти, чтобы определить, какие переменные нужно опубликовать в сети.

Протокол PSP

Для обмена данными Shared Variable Engine использует протокол NI-PSP (Publish Subscribe Protocol – протокол публикации/подписки). Этот сетевой протокол создан на основе протокола TCP и позволяет каждому клиенту соответствующей переменной общего доступа подписаться на данные, обслуживаемые Shared Variable Engine.

Развертывание библиотек переменных общего доступа на целевом устройстве, обслуживающем эти переменные

Система CompactRIO запускает Shared Variable Engine при начальной загрузке, и этот механизм получает доступ к энергонезависимой памяти, чтобы определить, какие библиотеки нужно развернуть. Библиотеки переменных общего доступа автоматически развертываются при запуске VI, который имеет доступ к любой из переменных, и когда вы развертываете приложение, которое также имеет доступ к любым переменным. Тем не менее, возможна ситуация, когда в системе не развернуто ни одной библиотеки. В этом случае Shared Variable Engine не открывает доступ к сетевым переменным.

Вы можете выбрать один из двух методов непосредственной развертывания библиотеки переменных общего доступа в целевом устройстве:

1. Вы можете указать систему CompactRIO в качестве целевой в проекте LabVIEW, ниже устройства поместить библиотеку и развернуть ее, в результате чего она запишется в энергонезависимую память контроллера CompactRIO и заставит Shared Variable Engine создать в сети новые элементы данных.

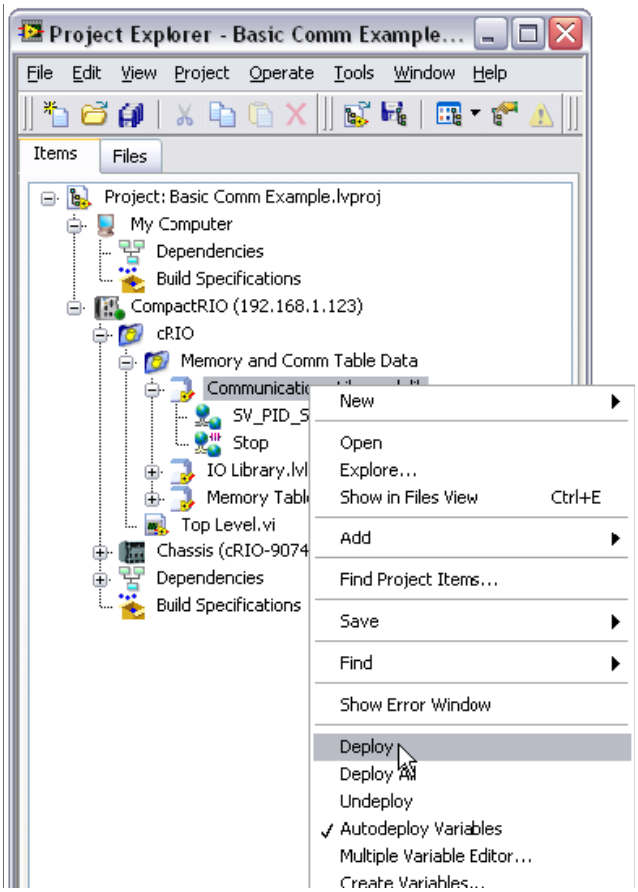


Рисунок 8.16. Развертывание библиотек в целевых устройствах реального времени путем выбора команды Deploy из контекстного меню

2. Вы можете развернуть библиотеку программным способом из приложения LabVIEW, работающего под Windows, с помощью узла Application invoke node.

- Щелкните правой кнопкой мыши по блок-диаграмме, чтобы открыть палитру функций, перейдите к субпалитре Programming»Application Control и поместите на блок-диаграмму узел Invoke Node
- Возьмите из палитры инструментов (Tools Palette) инструмента Operate Tool щелкните левой кнопки мыши по пункту Method узла Invoke Node и в появившемся меню выберите Library»Deploy Library
- Ко входу Lib Path узла подсоедините ссылку, которая указывает путь к библиотеке (библиотекам), в которых содержатся переменные общего доступа. На входе Target IP Address задайте IP адрес целевого устройства реального времени.

Рисунок 8.17. Вы можете программно развертывать библиотеки в целевых устройствах реального времени, используя узел Invoke Node на персональном компьютере

-

Удаление библиотеки переменных общего доступа

После развертывания библиотеки переменных общего доступа в Shared Variable Engine, библиотека и ее настройки будут храниться до тех пор, пока вы ее не удалите вручную. Чтобы удалить библиотеку, выполните следующие действия:

1. Запустите утилиту NI Distributed System Manager (Менеджер распределенных систем) из меню **LabVIEW Tools** или меню Start.
2. В раздел "My Systems" добавьте систему реального времени (с помощью команды Actions»Add System to My Systems)
3. Щелкните правой кнопкой мыши по библиотеке, которую вы хотите удалить и выберите команду Remove Process (Удалить процесс).

Развертывание приложений-клиентов переменных общего доступа

Запуск исполняемого файла, который является только клиентом общей переменной (не хостом) не требует каких-либо специальных действий по развертыванию библиотек. Тем не менее, контроллеру требуется способ перевода имени системы, в которой обслуживается переменная, в соответствующий IP адрес этой системы.

Рисунок 8.18. Узел сетевой переменной и сетевой путь к ней

Чтобы обеспечить масштабируемость, эта информация жестко не программируется в исполняемом файле. Вместо этого на целевом устройстве хранится файл, называемым файлом псевдонимов (alias file). Такой файл доступен для прочтения человеком и содержит логическое имя целевого устройства (CompactRIO) и его IP адрес (10.0.62.67). Когда запускается исполняемый файл, он считывает файл псевдонима и заменяет логическое имя на IP адрес. Если вы затем измените IP адреса развернутых систем, вам нужно только отредактировать файл псевдонимов для восстановления связи между двумя устройствами. Для целевых устройств реального времени под Windows XP Embedded спецификация построителя каждой развертываемой системы автоматически загружает файл псевдонимов. Для целевых устройств под Windows CE вам необходимо сконфигурировать спецификацию построителя так, чтобы файл псевдонимов загружался.

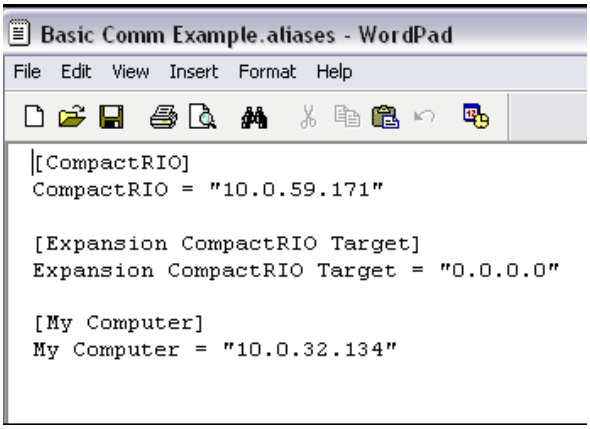


Рисунок 8.19. Файл псевдонимов, читаемый человеком, содержит список имен целевых устройств и IP адресов

Если вы развертываете системы, которые имеют динамические IP адреса, назначаемые с использованием DHCP, можете применять DNS имя вместо IP адреса. В проекте LabVIEW вы можете ввести имя DNS вместо IP адреса на странице свойств целевого устройства.

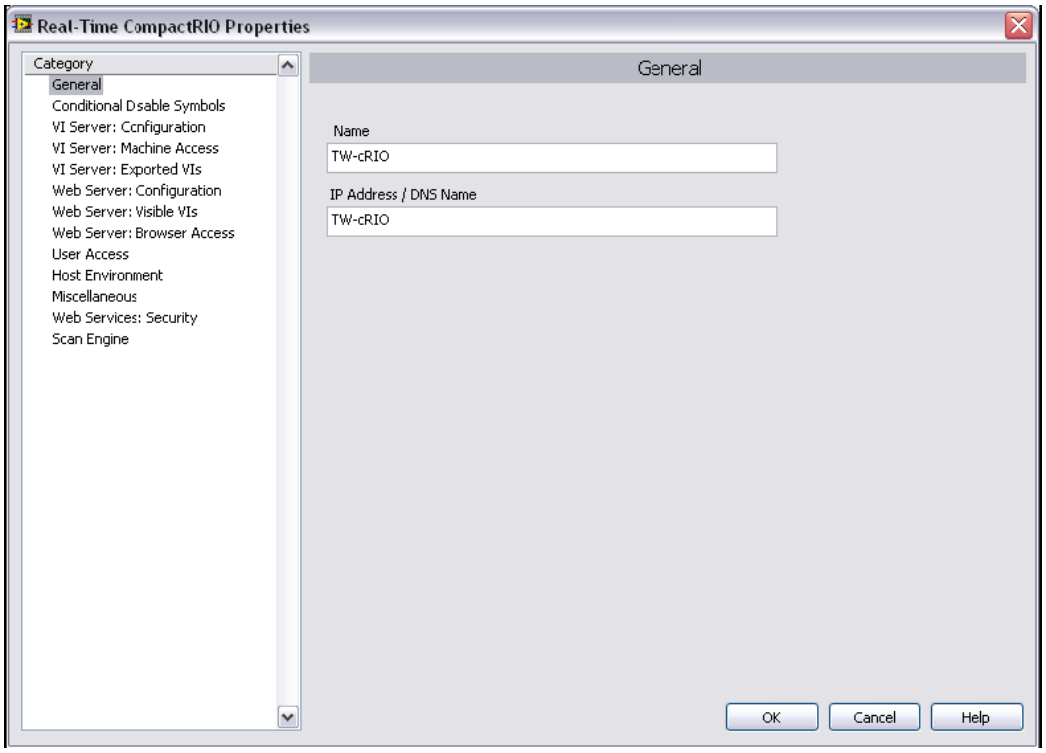


Рисунок 8.20. Для систем, использующих DHCP, можно ввести DNS имя вместо IP адреса

Если вам требуется масштабируемость, есть один хороший подход к разработке приложения на основе обобщенного целевого устройства (вы можете разрабатывать приложения для удаленных, пока не существующих платформ), который имеет имя, отражающее назначение приложения. Затем вы можете запустить исполняемый файл, как часть инсталлятора, который выдает приглашение ввести IP адреса удаленного устройства и своего компьютера или извлекает их из другого источника, например, из базы данных. Затем исполняемый файл может модифицировать файл псевдонимов, чтобы там отразились эти изменения.

Рекомендуемые стеки программ для CompactRIO

National Instruments предоставляет несколько рекомендуемых и чаще всего используемых наборов драйверов (Recommended software sets), которые могут быть инсталлированы на контроллеры CompactRIO с помощью утилиты Measurement and Automation Explorer.

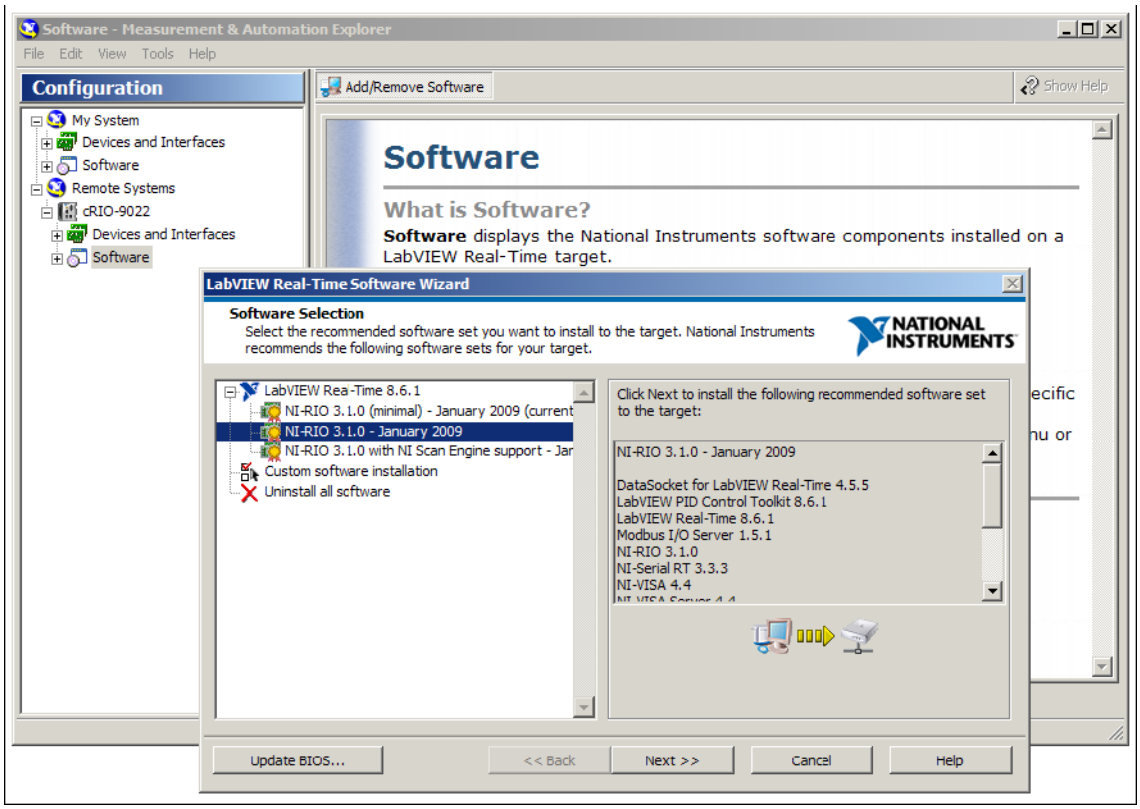


Рисунок 8.21. Рекомендуемые наборы программных средств инсталлируются на контроллер CompactRIO

Рекомендуемое производителем программное обеспечение дает гарантию, что приложение имеет одинаковые наборы основных драйверов для любой системы реального времени, которое имеет одно и то же программное обеспечение. В общем случае имеется минимально необходимый и полный набор программных средств.

Тиражирование системы

В данном разделе приведен пример кода LabVIEW

После того, как вы развернули спроектированное в LabVIEW приложение реального времени в контроллер CompactRIO, вам, возможно, захочется развернуть этот образ на другие такие же целевые устройства реального времени. Вы можете повторно разворачивать уже построенное приложение описанным выше путем с помощью LabVIEW Project и LabVIEW Application Builder. При попытке тиражирования и развертывании большого количества систем этот метод становится слишком громоздким.

Чтобы ускорить многократное развертывание, National Instruments предоставляет набор VI для тиражирования систем на целевые устройства реального времени. Вы можете использовать эти VI для копирования содержимого жесткого диска контроллера, работающего под LabVIEW Real-Time, и затем размножения этой информации на множество контроллеров. Эти средства можно применять для программной идентификации систем в сети и конфигурирования сети. Таким образом можно избавиться от необходимости использования утилиты Measurement and Automation Explorer (MAX) и FTP-клиента.

Утилиты тиражирования VI реального времени (Real Time Replication Utility VIs)

Начиная с версии LabVIEW 2009, вместе с компонентами LabVIEW Real-Time инсталлируются восемь VI для тиражирования системы.

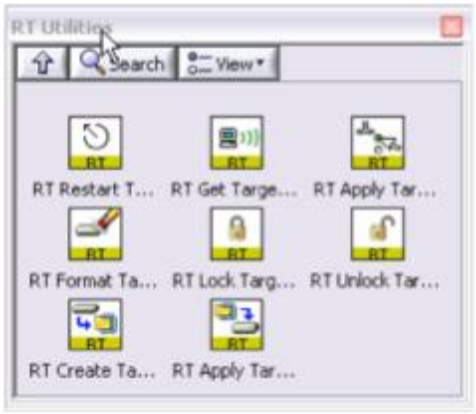


Рисунок 8.22. С помощью VI из палитры RT Utility можно программно конфигурировать контроллер и создать образ для конфигурирования

С помощью RT Get Target Information VI вы можете искать в сети устройства реального времени. Поиск можно выполнять по IP адресам и MAC адресам или искать все устройства реального времени производства NI в одной и той же подсети.

Когда RT Get Target Information VI обнаруживает в сети программируемое целевое устройство, утилита RT Create Target Disk Image устанавливает с ней соединение и перекачивает все содержимое через FTP протокол на ваш компьютер под Windows. Это содержимое сохраняется в виде zip-архива. Процесс копирования полной информации о контроллере занимает несколько минут.

Вы можете конфигурировать новые контроллеры с помощью VI RT Apply Network Settings и RT Apply Target Disk Image. Первый VI устанавливает на целевом устройстве настройки сети, а второй VI берет ранее сохраненный образ диска и загружает его содержимое в новый контроллер реального времени. Процесс копирования образа диска в контроллер занимает несколько минут.

Указанные выше VI помогают также блокировать и деблокировать FTP сервер контроллера реального времени по паролю.

Построение утилиты тиражирования

Рассмотренные VI позволяют построить в LabVIEW свою собственную утилиту тиражирования, работающую на компьютере под Windows.

В настоящее руководство включен пример заготовки тиражирующей утилиты на основе этих VI. Когда вы запускаете приложение на настольном компьютере, эта утилита предоставляет графический пользовательский интерфейс для управления контроллерами и размножения образов дисков.

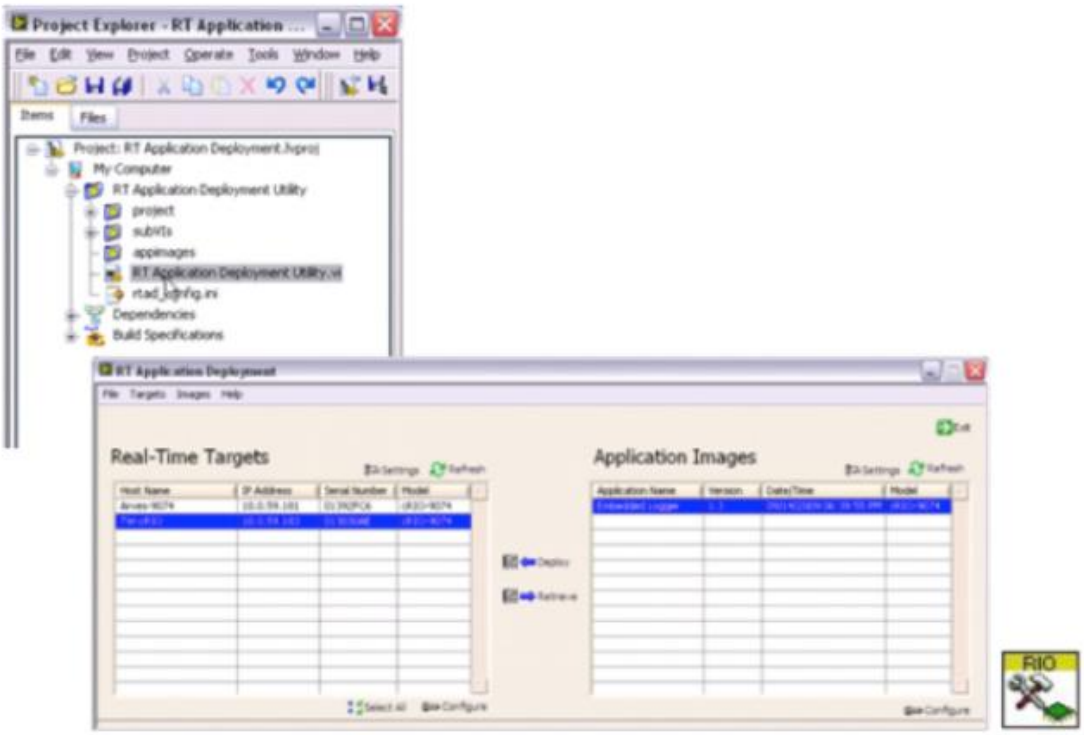


Рисунок 8.23. Вы можете создавать сложные программы тиражирования, например, как в данном примере

Пример приложения, приведенный на рисунке 8.23, автоматически сканирует сеть и ищет контроллеры реального времени, а также может сканировать папку жесткого диска устройств реального времени с сохраненными образами приложения. Вы можете извлекать образы дисков устройств реального времени, сохранять их на Windows-компьютере, а также развертывать образы приложения на целевом устройстве реального времени. Процесс развертывания или извлечения образов дисков занимает несколько минут. Более подробную информацию о работе с данным примером можно найти в файле readme.doc в проекте LabVIEW.

Средства тиражирования NI-RIO System Replication Tools

Набор средств NI-RIO System Replication Tools предлагает дополнительную поддержку рассмотренной выше палитры RT Replication Utility, предоставляя LabVIEW API, реализующих возможность программной загрузки двоичного файла FPGA во флэш-память шасси. Это полезно, если у вас есть развернутое приложение, причем программный код FPGA должен запускаться немедленно, и он не может быть загружен стандартным путем из программы реального времени. С помощью этих средств вы можете удалять и загружать двоичный файл FPGA во флэш-память и устанавливать его как VI, загружаемый из флэш-памяти. Набор средств NI-RIO System Replication Tools требует, чтобы на Host-компьютере, использующем эти VI, был установлен пакет NI-RIO.

Set RIO Device Settings.vi

VI Set RIO Device Settings предназначен для конфигурирования загрузки двоичного файла из флэш-памяти.

Download Bitfile.vi

VI Download Bitfile загружает заданный двоичный файл в одно или более FPGA целевых устройств, либо удаляет существующий файл. На вход VI подаются IP адрес Host-компьютера, ресурс целевого устройства FPGA (RIO0), путь к двоичному файлу и наименование выполняемой операции (загрузка или удаление двоичного файла).

Защита интеллектуальной собственности

В нашем случае к интеллектуальной собственности (Intellectual property – IP) относится любое уникальное программное обеспечение или алгоритм (алгоритмы) работы приложений, которые вы или ваша компания самостоятельно разработали. Этот может быть какой-то особенный алгоритм управления или полномасштабное инсталлируемое приложение. Обычно IP разрабатываются очень долгое время и дают компаниям способ выделиться в конкурентной борьбе. Поэтому охрана интеллектуальной собственности является очень важной задачей. Средства проектирования LabVIEW и CompactRIO предоставляют возможность защитить вашу интеллектуальную собственность. В общем случае есть два уровня защиты интеллектуальной собственности, которые вы можете реализовать:

Защита алгоритмов и программного кода от копирования и модифицирования

Если вы создали алгоритмы для выполнения специфических функций, например, продвинутых функций управления, специальных фильтров и т.д., возможно, вам захочется иметь возможность распространять алгоритм в виде subVI, но при этом защитить его от просмотра или изменения. Такой вариант может быть использован как для защиты интеллектуальной собственности, так и уменьшения расходов на поддержку, не давая другим участникам рынка модифицировать и взламывать ваши алгоритмы.

Привязка программного кода к определенной аппаратуре для предотвращения тиражирования интеллектуальной собственности

Применяйте данный вариант защиты, если вы хотите быть уверенными в том, что конкурент не сможет тиражировать вашу систему путем запуска вашей программы на другой системе CompactRIO или хотите, чтобы ваши клиенты обращались к вам за сервисом и поддержкой.

Защита алгоритмов и программ от копирования и модифицирования

Защита инсталлируемой программы

Среда LabVIEW приспособлена для защиты всех развертываемых программ, а также всех приложений, запускаемых при включении контроллера CompactRIO. По умолчанию эти программы заблокированы и их нельзя открыть. В отличие от других серийных контроллеров и некоторых PLC-контроллеров, где исходный текст программы хранится прямо в контроллере и защищен только паролем, системы CompactRIO не требуют, чтобы исходный текст программы хранился в контроллере.

Программа, работающая на процессоре реального времени компилируется в исполняемый файл и не может быть "рекомпилирована" обратно в код LabVIEW. Аналогично, код, скомпилированный в двоичный файл и исполняемый в FPGA, также не может быть "рекомпилирован" в код LabVIEW. Чтобы обеспечить в будущем возможность отладки и сопровождения, можно сохранить проект LabVIEW на контроллере или вызвать исходные VI из запускаемой программы, но по умолчанию любая программа, инсталлированная в контроллере реального времени защищена от копирования или модифицирования алгоритмов.

Защита отдельных VI

Иногда бывает, что вы хотите передать конечным потребителям исходный код LabVIEW для его настройки или сопровождения, но в то же время вам нужно защитить некоторые алгоритмы. LabVIEW предоставляет несколько методов, которые дают возможность делать удобные, и, в то же время, защищенные subVI внутри VI.

Метод 1. Защита программы на LabVIEW паролем

Защита VI паролем требует от пользователей ввести пароль, если они хотят редактировать или просматривать блок-диаграмму конкретного VI. Благодаря этому вы можете передавать VI кому-нибудь другому, защитив исходный код. Защита паролем subVI запрещает другим лицам редактировать VI или просматривать его блок-диаграмму без ввода пароля. Однако, если пароль утерян, нет возможности разблокировать VI. Поэтому, необходимо внимательно подумать над тем, чтобы хранить резервные копии ваших файлов без паролей в другом безопасном месте.

Чтобы защитить паролем VI, перейдите в меню **File»VI Properties**. Выберите раздел Protection. Вам предоставляются три варианта:

- Unlocked (состояние VI по умолчанию) – незаблокирован
- Locked (no password) – заблокирован без пароля
- Password-protected – защищен паролем

Пароль вступает в силу при следующем запуске LabVIEW.

Рисунок 8.24. Пароль, который защищает программу LabVIEW

Пароль в LabVIEW крайне трудно взломать, однако ни один алгоритм формирования пароля не дает 100% защиты от попытки проникновения. Если вам нужно полная гарантия, что никто не сможет получить доступ к вашему исходному коду, то следует предусмотреть удаление блок-диаграмм.

Метод 2. Удаление блок-диаграммы

Чтобы гарантировать невозможность модификации или открытия VI, можно полностью удалить блок-диаграмму. Почти, как и исполняемый файл, распространяемая вами программа более не содержит исходного редактируемого кода. Не забудьте сделать резервные копии ваших файлов, поскольку блок-диаграммы не подлежат восстановлению. Вариант удаления блок-диаграммы можно выбрать при создании комплекта распространяемых исходных файлов LabVIEW, который можно упаковать и отправить другим разработчикам. Вы можете конфигурировать настройки для отдельных VI, чтобы добавлять пароли, удалять блок-диаграммы и т.д.

Для создания комплекта распространяемых исходных файлов выполните следующие действия:

1. В проекте LabVIEW щелкните правой кнопкой мыши по пункту Build Specifications и выберите команду New»Source Distribution, чтобы отобразилось диалоговое окно Source Distribution Properties. Добавьте ваши VI список распространяемых файлов.
2. На странице Source File Settings (Настройки исходного файла) в этом же диалоговом окне снимите флажок Use default save settings (Использовать параметры сохранения по умолчанию). Затем установите флажок Remove block diagram (Удаление блок-диаграммы), чтобы удалить блок-диаграммы.

3. Постройте комплект распространяемых исходных файлов, чтобы создать копию VI без блок-диаграммы.

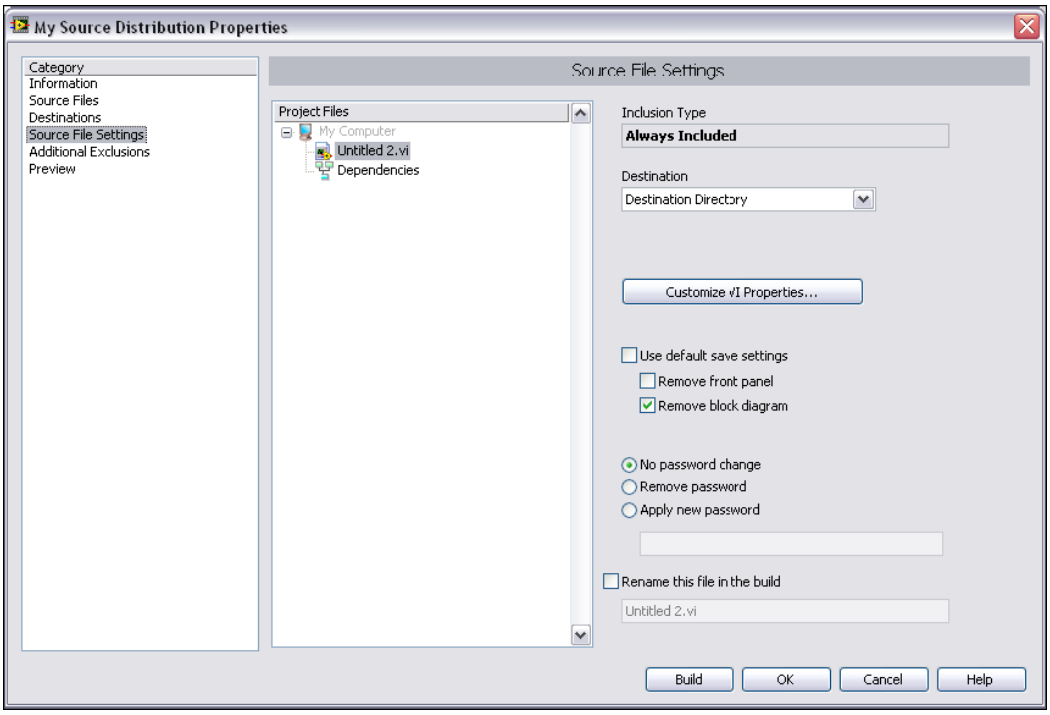


Рисунок 8.25. Удаление блок-диаграммы из LabVIEW VI

Внимание! Если вы сохраняете VI без блок-диаграмм, не перезаписывайте исходные версии этих VI. Сохраняйте такие VI в разных папках или под разными именами.

Привязка программы к аппаратуре для предотвращения тиражирования интеллектуальной собственности

Некоторые изготовители оборудования и компьютеров также хотят защитить свою интеллектуальную собственность путем привязки инсталлируемой программы к определенной системе. Чтобы облегчить тиражирование систем, инсталлируемая на контроллер CompactRIO программа по умолчанию не привязана к аппаратуре и легко может быть перемещена и исполнена на другом контроллере. Для разработчиков, которые хотят защитить свои системы от тиражирования клиентами и конкурентами, эффективным способом защиты программного кода приложения, относящегося к CompactRIO, является привязка кода к определенным аппаратным компонентам системы. Это гарантирует, клиенты не смогут извлечь программный код из системы, которую они у вас купили, и запустить приложение на другом комплекте оборудования CompactRIO. Вы можете привязать программный код приложения к различным аппаратным компонентам системы CompactRIO, в том числе, используя:

- MAC адрес контроллера реального времени
- Серийный номер контроллера реального времени
- Серийный номер шасси CompactRIO
- Серийный номер отдельных модулей
- Электронный ключ от третьей стороны

Описанные ниже действия помогут вам программно привязать любое приложение к любому из приведенных выше параметров оборудования и, следовательно, не дать пользователям размножать программный код приложения:

1. Получите информацию о параметрах устройства. Для получения более подробных сведений о программном извлечении этой информации, обратитесь к процедурам, описанным ниже.

2. Сравните полученные значения параметров устройства с предопределенными значениями, для которых разработан код приложения, с помощью функции Equal? (Равно?) из палитры Comparison (Сравнение).
3. Подсоедините выход результата сравнения ко входу селектора case-структуры.
4. Поместите программный код приложения в кадр true, а кадр false оставьте пустой.

Выполнение этих действий гарантирует, что приложение нельзя тиражировать и использовать на любых других компонентах аппаратуры CompactRIO.

Лицензионный ключ

При развертывании множества систем жесткое программирование идентификации аппаратных средств может оказаться неидеальным вариантом, поскольку оно требует вручную изменять исходный код программы и повторной компиляции для каждой устанавливаемой системы. Эта проблема может быть решена путем использования файла с лицензионным ключом, который хранится отдельно от программного кода приложения на контроллере CompactRIO. Этот файл может быть легко обновлен для каждой системы без необходимости изменения приложения. Кроме чтения MAC-адреса или серийного номера, VI может открыть лицензионный файл и проверить, действительна ли лицензия. Для безопасности лицензионный файл должен быть отдельным для каждой развертываемой системы, и ваша программа должна выполнять математические операции с лицензионным ключом и специальными свойствами аппаратуры, например, MAC адресом. Поскольку MAC адрес и серийные номера могут быть определены программным способом в Windows или в операционной системе реального времени, то вы можете разработать развертываемое приложение LabVIEW, которое автоматически запрашивает систему CompactRIO, генерирует и устанавливает файл с соответствующим лицензионным ключом.

Получение MAC адреса системы CompactRIO

В данном разделе приведен пример кода LabVIEW

Вы можете вручную получить MAC-адрес контроллеров реального времени из утилиты MAX, обратившись на закладку сетевых настроек контроллера или из командной строки Windows. Вы можете также определить MAC адрес программно.

Получение MAC адреса из Windows

Чтобы из Windows определить программным путем MAC адрес системы реального времени, выполните следующие действия:

1. Запустите VI RT Ping Controllers, который находится в палитре Real-Time»Real-Time Utilities. Этот VI возвращает сетевую информацию по всем контроллерам реального времени определенной подсети.

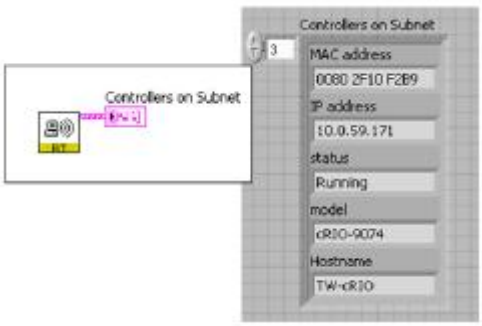


Рисунок 8.26. Получение MAC адреса сетевых платформ реального времени из Windows

2. Среди полученных данных от VI RT Ping Controllers найдите нужный контроллер по IP адресу или по имени.
3. Кластер на выходе VI RT Ping Controllers возвращает MAC адреса контроллеров в виде массива строк в шестнадцатеричном формате.

Получение MAC адреса из платформы реального времени

1. В процессе соединения с контроллером реального времени запустите VI RT Ping Controllers, который находится в палитре Real-Time»Real-Time Utilities. Присоедините ко входу system location (адрес системы) константный кластер с логической константой False и строковой константой для ввода имени компьютера. Данный VI возвращает информацию о контроллере реального времени, на котором выполняется программа.

Рисунок 8.27. Получение MAC адреса на платформе реального времени

2. Среди данных, полученных от VI, найдите MAC адреса контроллеров, которые возвращаются в виде массива строк в шестнадцатеричном формате.

Палитра CompactRIO Information Retrieval Tools

Вы можете также извлечь другую информацию о системе CompactRIO, например, серийные номера, с помощью палитры CompactRIO Information Retrieval Tools. Эти бесплатные VI можно запускать под Windows или на контроллере реального времени, чтобы восстановить информацию о локальном или удаленном контроллере CompactRIO, объединительной плате и модулях, в том числе тип и серийный номер каждого из этих компонентов.

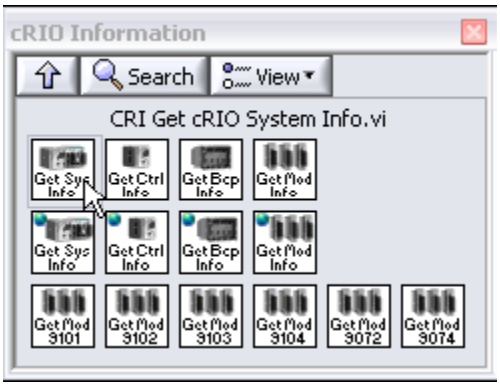


Рисунок 8.28. VI из палитры CompactRIO Information Retrieval Tools возвращают такую информацию как серийные номера

Палитра CompactRIO Information Retrieval Tools позволяет также восстанавливать серийный номер платформы реального времени программным путем.

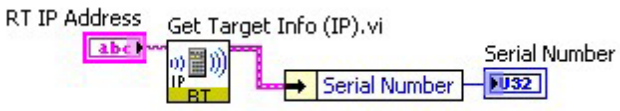


Рисунок 8.29. Получение серийного номера с помощью палитры CompactRIO Information Retrieval Tools

Перенос на другие платформы

Настоящий документ сконцентрировал внимание на архитектурах для построения встраиваемых систем управления с использованием систем CompactRIO. Те же базовые технологии и структуры работают на других управляющих платформах National Instruments, в том числе PXI и NI Single-Board RIO. Благодаря этому вы можете повторно использовать ваши алгоритмы и вашу архитектуру для других проектов, в которых требуется другое оборудование, или без проблем перемещать ваше приложение между платформами. Тем не менее, CompactRIO имеет ряд возможностей для облегчения изучения и ускорения разработки, которые недоступны на других платформах. Далее излагаются сведения, которые необходимо учитывать при перемещении между платформами, и показывается пример, как переносить приложения на платформу NI Single-Board RIO.

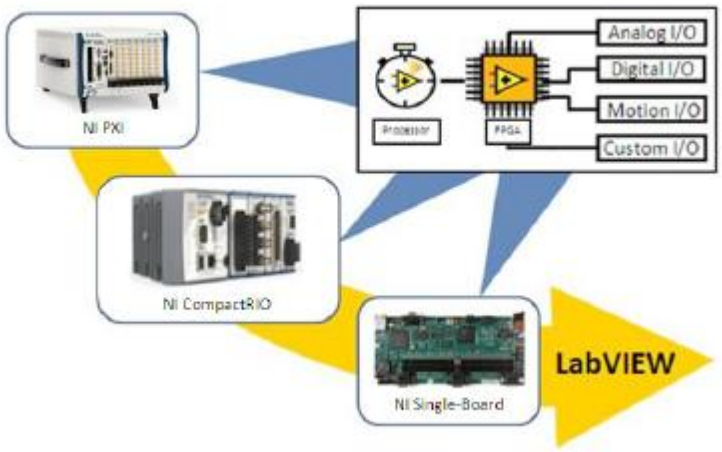


Рисунок 8.30. В LabVIEW Вы можете использовать одну и ту же архитектуру от CompactRIO до высокопроизводительной PXI и одноплатной NI Single-Board

Переносимость программы на LabVIEW

LabVIEW является межплатформенным языком программирования, который может компилировать программы для многих архитектур процессоров и операционных систем. В большинстве случаев алгоритмы, спроектированные в LabVIEW, являются переносимыми между всеми LabVIEW платформами. Действительно, вы даже можете взять программу на LabVIEW и скомпилировать ее для любого 32-разрядного процессора, что позволяет вам ее перенести на оборудование заказчика. При переносе кода между платформами наиболее часто необходимые изменения относятся к физическому уровню ввода-вывода.

При переносе кода между платформами CompactRIO все операции ввода-вывода однозначно совместимы, поскольку модули C серии поддерживаются на всех платформах CompactRIO. Если вам необходимо перенести приложение на NI Single-Board RIO, поддерживаются все модули C серии, но, в зависимости от вашего приложения, возможно, потребуется настройка программного интерфейса ввода-вывода.

NI Single-Board RIO

NI Single-Board RIO является одноплатной версией CompactRIO, предназначенной для приложений, где требуется конструктив несмонтированной платы. Поскольку такая архитектура физически по-другому устроена, в ней используется процессор и FPGA, и в нее могут входить до трех модулей C Series. NI Single-Board RIO отличается от CompactRIO тем, что подсистема ввода-вывода встроена непосредственно в плату. В состав устройства типа NI Single-Board RIO входят 110 двунаправленных линий цифрового ввода вывода (3.3 В), до 32 линий аналогового ввода, до 4 линий аналогового вывода, а также до 32 линий цифрового ввода и вывода (24 В) в зависимости от конкретной используемой модели.

Программирование FPGA на LabVIEW

В настоящее время NI Single-Board RIO не поддерживает режим мультиплексирования, из-за чего вам необходимо на LabVIEW разрабатывать программу для чтения данных ввода-вывода из FPGA и

вставки их в таблицу памяти. Ниже анализируется эффективная FPGA архитектура для одноточечного ввода-вывода аналогично режиму мультиплексирования, который рассмотрен позже, и демонстрируется, как скрыть приложение, в котором используется режим мультиплексирования.

Встраиваемый ввод-вывод и модули ввода-вывода

В зависимости от потребностей вашего приложения по вводу-выводу вы можете создавать полностью свое приложение для работы с каналами ввода-вывода, доступными на плате NI Single-Board RIO, или, возможно, вам потребуется добавить модули в приложение. Далее приведены функциональные возможности ввода-вывода доступные на плате NI Single-Board RIO и соответствующие им отдельные модули, указанные в скобках:

- 110 каналов цифрового ввода-вывода общего назначения, 3.3 В (5.5 В макс., ТТЛ совместимые) (эквивалентные модули отсутствуют)
- 32 несимметричных /16 дифференциальных каналов аналогового ввода (разрядность 16 бит, частота дискретизации 250000 $с^{-1}$ на все каналы) (NI 9205)
- 4-канальный аналоговый вывод (16 бит, 100000 $с^{-1}$ в режиме параллельного сбора данных) (NI 9263)
- 32-канальный цифровой ввод (входные цепи с общим истоком) (NI 9425)
- 32-канальный цифровой вывод (выходные цепи с общим стоком) (NI 9425)

Платформа NI Single-Board RIO может работать максимум с тремя дополнительными модулями C серии. Приложение, где требуется более трех дополнительных модулей ввода-вывода слабо подходят для платформы NI Single-Board RIO. В этом случае вам в качестве инсталляционной платформы следует рассматривать интегрированные системы CompactRIO.

Объем FPGA

FPGA-устройство самого большого объема, доступное на платформе NI Single-Board RIO – Xilinx 2M system gate Spartan-3. Платформы CompactRIO предоставляют версии, использующие как Spartan-3 и выше, так и более быстрые Virtex-5. Тестировать совместимость программного кода с оборудованием у вас нет возможности, но вы можете добавить платформу в ваш проект в LabVIEW и, поскольку разрабатываете FPGA-приложение, вы можете периодически тестировать приложение путем компиляции FPGA кода для эмулируемой RIO-платформы. Это поможет вам хорошо понять, насколько ваше приложение совместимо с FPGA Spartan-3.

Перенос приложений CompactRIO на NI Single-Board RIO или устройства R серии

Для переноса приложения CompactRIO на NI Single-Board RIO или FPGA устройства ввода-вывода PXI/PCI устройства R серии выполните следующие основные действия:

1. Постройте проект для платформы NI Single-Board RIO или устройства R серии с эквивалентными каналами ввода-вывода
2. Если вы используете CompactRIO в режиме мультиплексирования (CompactRIO Scan Mode), постройте для этого режима API интерфейс на основе LabVIEW FPGA
    a. Постройте на LabVIEW FPGA программу мультиплексирования ввода-вывода(аналоговый ввод, аналоговый вывод, цифровой ввод-вывод, специальный цифровой ввод-вывод)
    b. Преобразуйте имена-псевдонимы переменных ввода-вывода в общие переменные состояния отдельных процессов с помощью разрешенной FIFO-очереди реального времени
    c. Постройте мультиплексирование ввода-вывода в реальном времени с помощью масштабирования и таблицы текущих значений переменных состояний процессов
3. Скомпилируйте VI LabVIEW FPGA для новой платформы
4. Протестируйте и подтвердите достоверность обновленного приложения реального времени и FPGA приложения

На первом шаге при переносе приложения с CompactRIO на NI Single-Board RIO или FPGA-устройство R серии происходит поиск эквивалентных типов ввода-вывода на вашей платформе. Для типов ввода-вывода, которые не могут быть перенесены на NI Single-Board RIO или платформу R серии, вы можете добавлять модули C серии. Все модули C серии для CompactRIO совместимы как с NI Single-Board RIO так и с устройствами R серии. Чтобы добавить модули ввода-вывода C серии к устройству сбора данных (DAQ-устройству) R серии, необходимо использовать шасси расширения R серии NI 9151.

Второй шаг необходим, только если переносимое приложение было изначально написано с учетом использования режима мультиплексирования. Если вам нужно заменить часть приложения, связанную с мультиплексированием, на метод ввода-вывода, поддерживаемый NI Single-Board RIO или устройством R серии, приведенный ниже пример вам поможет пройти весь процесс.

Если в приложении, которое вы перемещаете в NI Single-Board RIO, не используется режим мультиплексирования, можно считать, что процесс переноса почти завершен. В этом случае пропустите шаг 2 и добавьте ваш исходный код для FPGA и платформы реального времени к вашему новому проекту под NI Single-Board RIO, снова скомпилируйте VI FPGA. Теперь вы готовы запустить приложение и проверить его функциональность. Поскольку CompactRIO и NI Single-Board RIO обе базируются на RIO архитектуре и многократно используемых модулях ввода-вывода C серии, перенос приложений между этими платформами является очень простым.

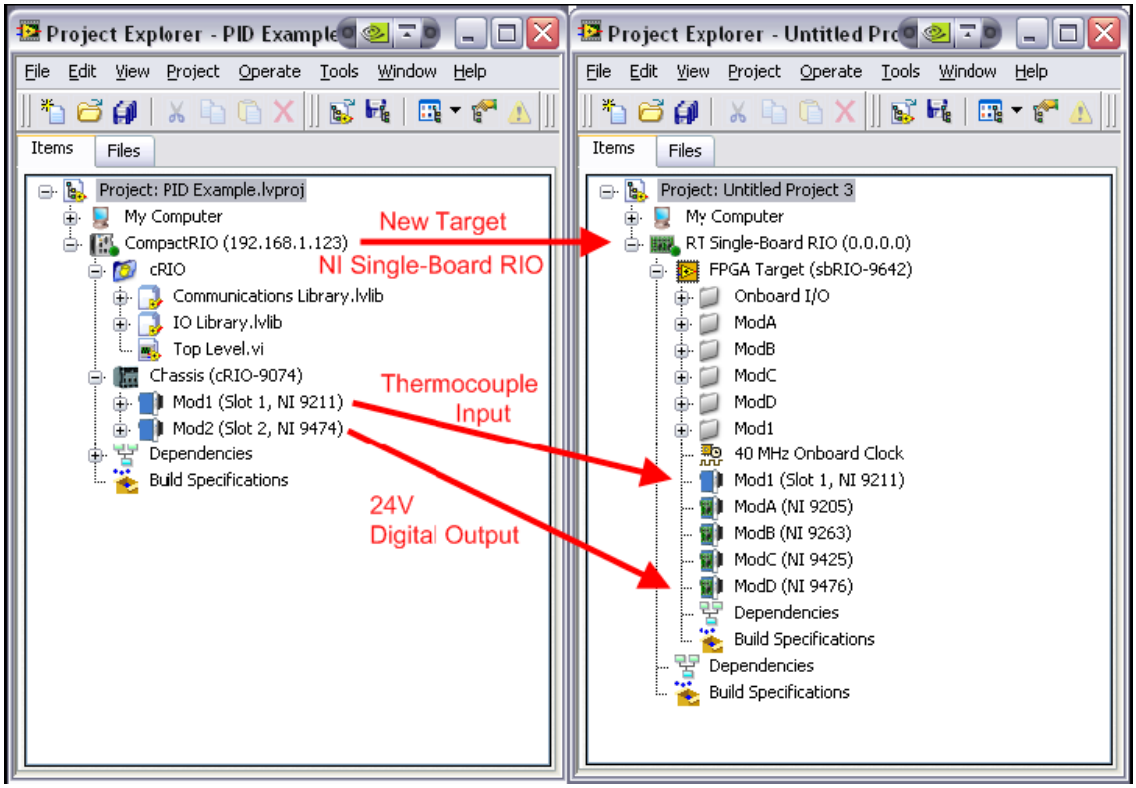


Рисунок 8.31. Первый шаг при переносе приложения с CompactRIO на другую платформу – поиск замены модулей ввода-вывода на будущей платформе

Пример переноса приложения CompactRIO с мультиплексированием на платформу NI Single-Board RIO

В данном разделе приведен пример кода LabVIEW

Если вы в исходном приложении используете режим мультиплексирования, то вам следует создать на основе FPGA упрощенную версию режима мультиплексирования, поскольку устройства типа NI Single-Board RIO и DAQ-устройства R серии этот режим не поддерживают. Построение мультиплексора на FPGA очень похоже на метод вставки одноточечных данных из FPGA в процесс мультиплексирования в реальном времени, который рассматривается в параграфе

"Программирование в LabVIEW FPGA". Чтобы заменить режим мультиплексирования похожим мультиплексором на основе FPGA, необходимо выполнить три действия:

1. Постройте мультиплексор ввода-вывода с помощью LabVIEW FPGA
2. Замените переменные ввода-вывода общими переменными состояния отдельных процессов.
3. В LabVIEW Real-Time вставьте данные из FPGA в таблицу текущих значений, созданную на основе общих переменных

В первую очередь создайте VI LabVIEW FPGA, который дискретизирует и обновляет все каналы аналогового ввода и вывода с частотой, заданной в конфигурации вашего мультиплексора. Для восстановления специальных цифровых устройств, таких, как счетчики, ШИМ модулятор и импульсный датчик положения, вы можете использовать IP блоки.

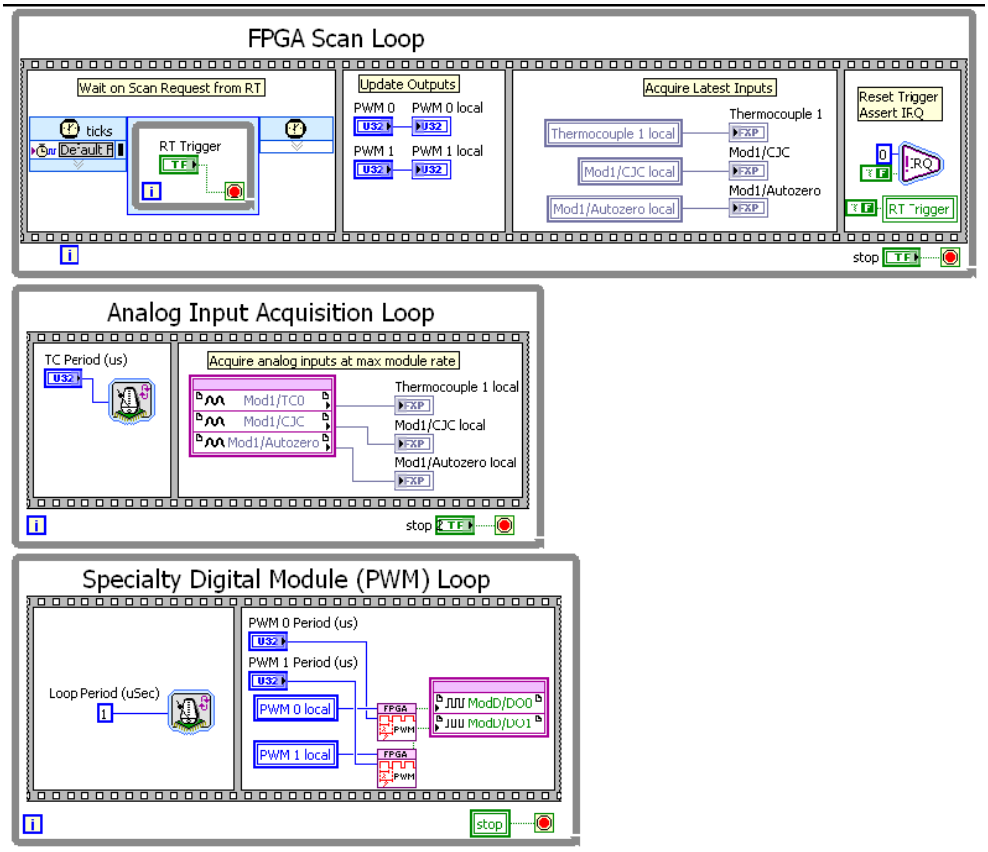


Рисунок 8.32. Разработка простого FPGA приложения, которое работает как FPGA мультиплексор.

После того, как вы реализовали простой мультиплексор в FPGA, настало время переносить часть приложения, которая работает в реальном времени, чтобы обмениваться информацией с отдельным FPGA мультиплексором, который лучше чем текущее значение таблицы, встроенной в режим мультиплексирования. Чтобы завершить данное действие, вам нужно в первую очередь преобразовать все псевдонимы переменных ввода-вывода в общие переменные состояния процессов, для которых разрешена FIFO очередь реального времени. Главное различие между двумя типами переменных заключается в том, что пока переменные ввода-вывода автоматически обновляются драйвером, чтобы отразить состояние канала ввода или вывода, общие переменные состояния процессов драйвером не обновляются. Вы можете изменить тип переменной, если зайдете на страницу свойств каждого псевдонима переменной ввода-вывода и измените тип single-process (переменная состояния отдельного процесса).

Указание: Если у вас есть множество переменных для преобразования, вы можете легко преобразовать библиотеку псевдонимов переменных ввода-вывода в общие переменные путем экспорта в текстовый редактор и изменения свойств. Убедиться в том, что вы правильно извлекаете свойства, проще всего, если вы в первый раз создаете первую временную общую переменную состояния процесса с FIFO очередью реального времени из одного элемента, разрешенную в библиотеке, и затем экспортируете библиотеку в редактор электронных таблиц. В

редакторе электронных таблиц следует удалить столбцы кроме переменных ввода-вывода и скопировать данные кроме общих переменных в строки переменных ввода-вывода. Затем импортируйте модифицированную библиотеку в ваш новый проект. Псевдонимы переменных ввода-вывода импортируются в виде общих переменных состояния отдельных процессов. Поскольку LabVIEW ссылается на общие переменные и псевдонимы переменных ввода-вывода по имени библиотеки и имени переменной, все экземпляры псевдонимов переменных ввода-вывода в вашем VI автоматически обновляются. Наконец, удалите временную общую переменную, котороая была создана перед процессом перемещения.

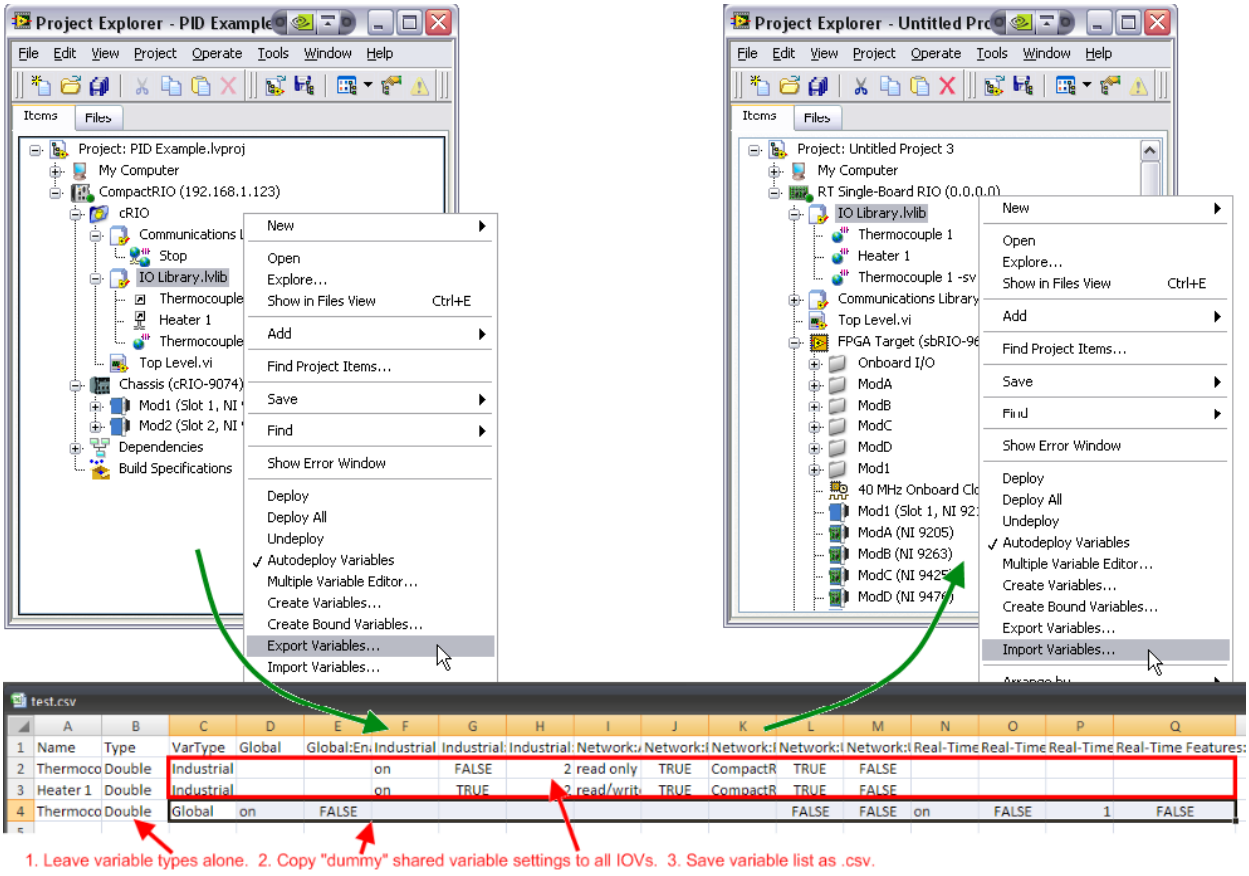


Рисунок 8.33. Вы можете легко преобразовать библиотеку псевдонимов переменных ввода-вывода IOV Alias Library в общие переменные путем экспорта переменных в электронную таблицу, путем изменения параметров и импорта на вашу новую платформу.

Заключительным этапом реализации FPGA мультиплексора и таблицы текущих значений общих переменных является построение задачи реального времени, которая считывает данные из FPGA и постоянно обновляет таблицу текущих значений. FPGA ввод-вывод, который добавляется в таблицу текущих значений, является детерминированным. Следовательно, вы снова используете метод, описанный в параграфе "Программирование в LabVIEW FPGA" за исключением настоящего момента, когда вы создаете фрагмент реального времени для данной программы.

Чтобы считать данные из FPGA мультиплексора, создайте задачу с циклом заданной длительности, для которой установлена нужная частота мультиплексирования вашего RT VI самого верхнего уровня. Этот цикл является детерминированным циклом ввода-вывода. Таким образом, для него должен быть установлен самый высокий приоритет. Чтобы подобрать скорость цикла управления вашего предыдущего приложения с мультиплексированием, установите период данного цикла равным ранее установленному периоду для режима мультиплексирования. Любые другие циклы задач в вашем приложении, которые ранее синхронизировались с мультиплексором, также требуют изменения источника синхронизации на генератор, выдающие сигнал с частотой 1 кГц, и устанавливают такую же частоту дискретизации как частоту повторения цикла ввода-вывода.

Цикл мультиплексирования ввода-вывода помещает новые данные в FPGA и затем вытягивает оттуда обновленные входные данные. Специальные VI для чтения и записи также несут ответственность за масштабирование и калибровку аналогового и специального цифрового ввода-вывода.

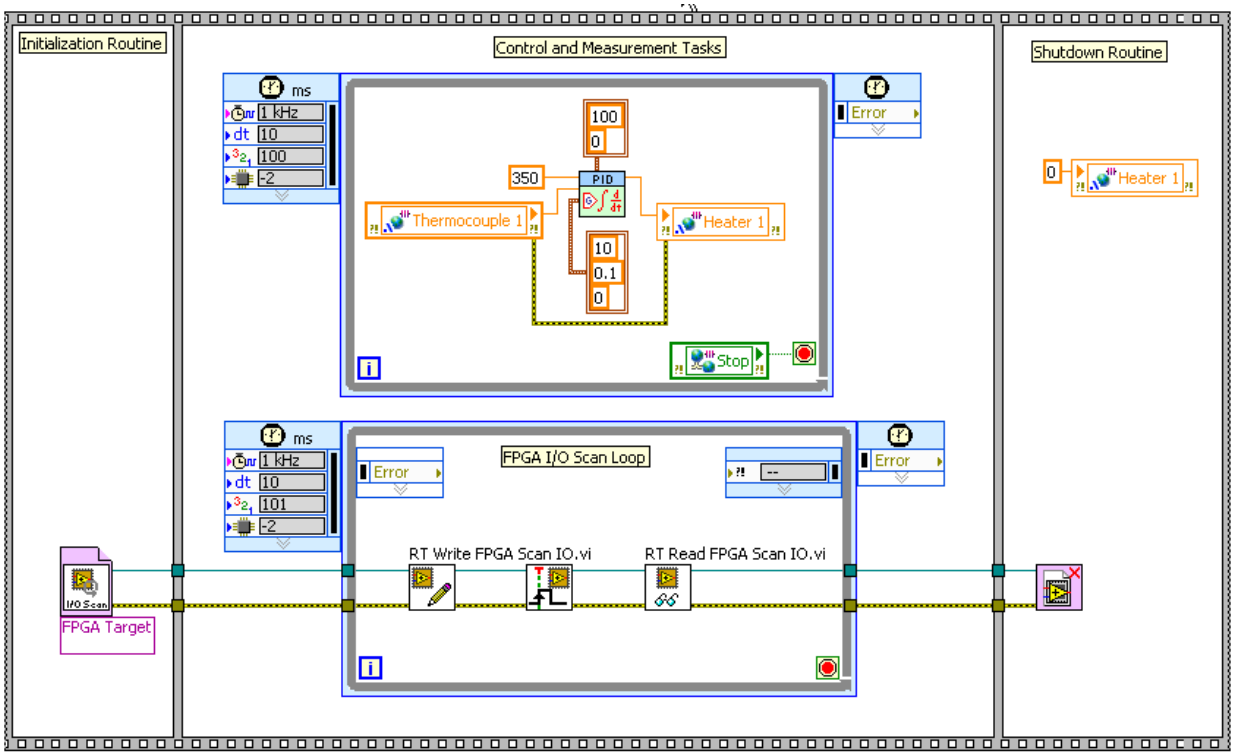


Рисунок 8.34. Цикл сканирования каналов ввода-вывода FPGA имитирует режим мультиплексирования CompactRIO путем детерминированной передачи самых новых значений в каналы ввода-вывода или из каналов ввода-вывода FPGA и вставки данных в таблицу памяти

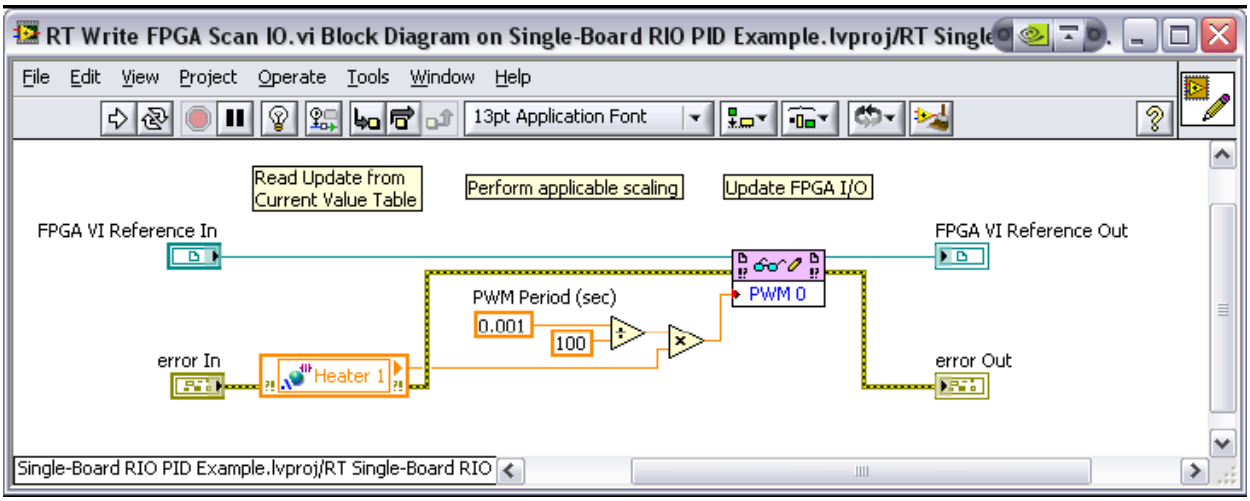


Рисунок 8.35. VI RT Write FPGA Scan IO извлекает данные из таблицы памяти с помощью общей переменной состояния процесса реального времени, находящейся в FIFO очереди, масштабирует их с помощью соответствующего преобразования для VI FPGA Scan и помещает их в VI FPGA.

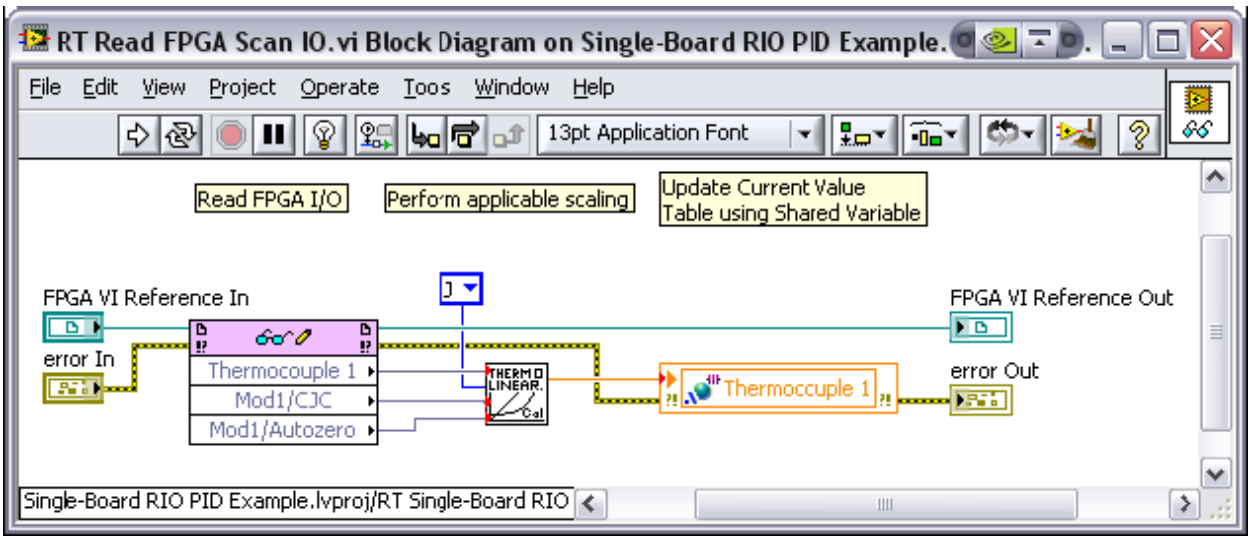


Рисунок 8.36. VI RT Read FPGA Scan IO извлекает все обновленные отсчеты из очередного цикла сканирования FPGA, выполняет подходящие преобразования и масштабирование, и помещает данные в таблицу памяти с помощью общей переменной состояния процесса реального времени, находящейся в FIFO очереди

После построения интерфейсной части, находящейся на компьютере и относящейся к индивидуальному мультиплексированию каналов ввода-вывода, которая заменяет режим мультиплексирования, вы готовы тестировать и подтверждать перенос приложения на новую платформу. Убедитесь, что VI FPGA скомпилирован и что платформы реального времени и FPGA, которые есть в проекте, правильно сконфигурированы, т.е. им назначен корректный IP адрес и имя RIO ресурса. После того, как VI FPGA скомпилирован, соедините компьютер с платформой реального времени и запустите приложение.

Поскольку RIO архитектура является общей для NI Single-Board RIO, CompactRIO, устройств ввода-вывода типа FPGA R серии, программный код на LabVIEW для каждой из этих платформ является легко переносимым с одной платформы на другую. Как продемонстрировано в данном параграфе, при правильном планировании вы можете перемещать приложения между всеми платформами без какого-либо изменения программы. Когда вы используете специализированные свойства одной платформы, например, режим мультиплексирования CompactRIO Scan Mode, процесс переноса является более сложным. Тем не менее, в этом случае необходимо изменить только фрагменты программы, связанные с вводом-выводом. В обеих ситуациях, все алгоритмы обработки и управления на LabVIEW являются полностью переносимыми и многократно используемыми по всем аппаратным платформам RIO.

ПРИЛОЖЕНИЕ А
Начало работы с CompactRIO

Начало работы с учебным пособием по CompactRIO

В настоящем учебном пособии показано, как распаковать новую систему CompactRIO и создать ваш первый проект. В нем объясняется, каким образом сконфигурировать аппаратные средства, инсталлировать необходимые программные компоненты и как создать простой проект, чтобы развернуть его в систему CompactRIO.

Система CompactRIO и ее компоненты

Система CompactRIO состоит из следующих пяти основных компонентов:

1. Шасси – Chassis
2. Контроллер – Controller (он может быть интегрирован в шасси в таких устройствах как cRIO-907x)
3. Модули – Modules
4. Вспомогательные элементы системы – Accessories
5. Программное обеспечение – Software

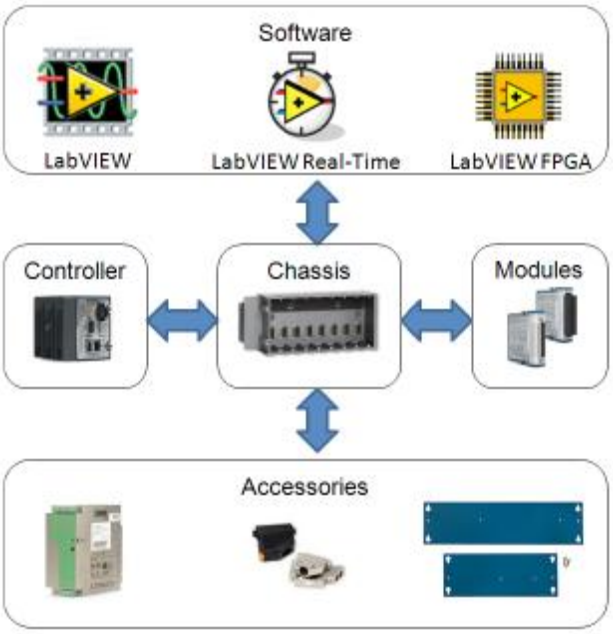


Рисунок 9.1. Компоненты системы CompactRIO

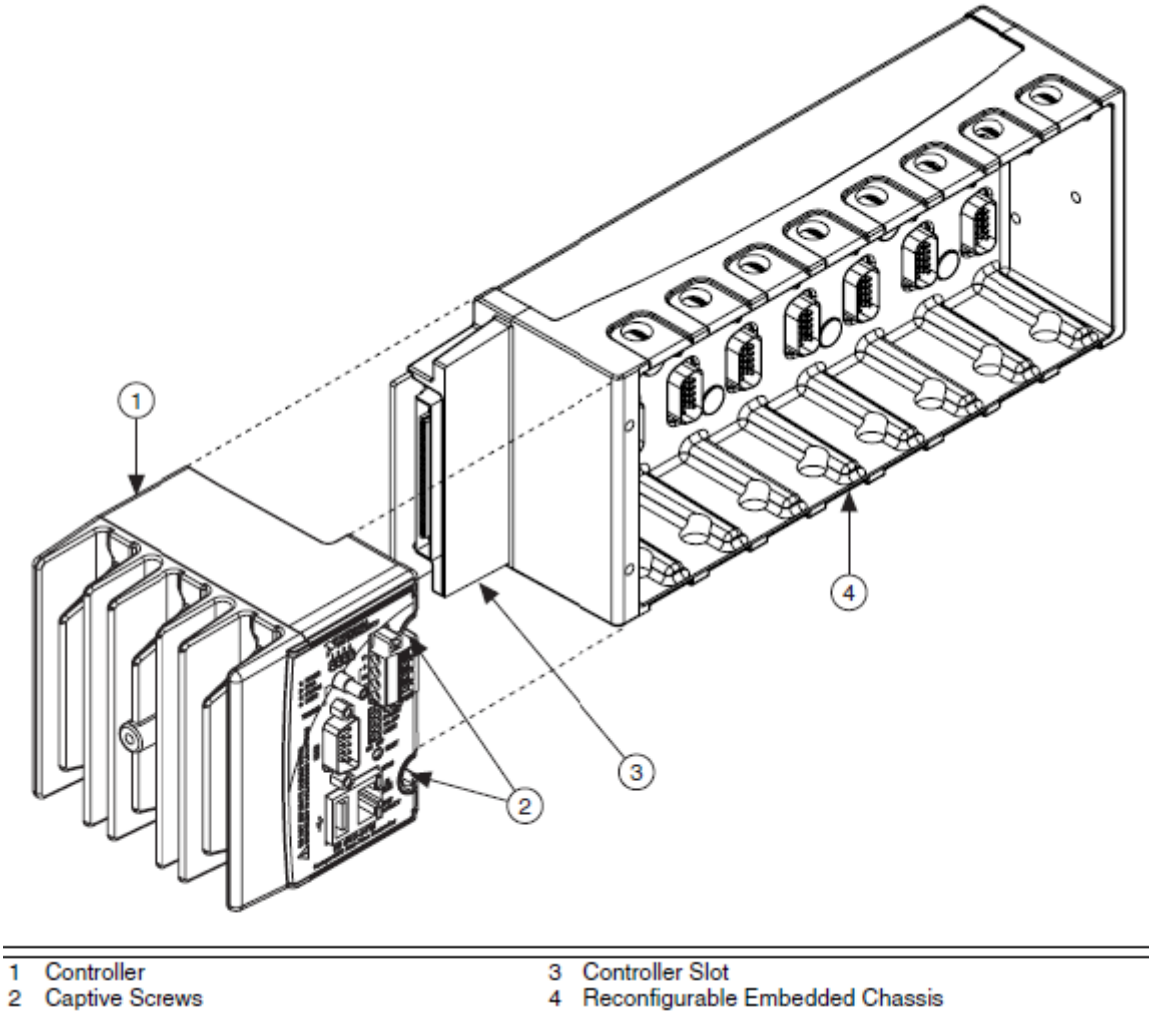
Сборка и конфигурирование аппаратуры

После извлечения компонентов системы CompactRIO из коробки, выполните следующие действия, чтобы подготовить систему для инсталляции программного обеспечения и программирования.

Соединение контроллера, шасси и модулей

Если у вас есть контроллер модульного типа, присоедините его к реконфигурируемому FPGA шасси. Серия cRIO-907x состоит из интегрированного шасси с контроллером, т.е. никакой сборки не требуется. Чтобы присоединить контроллер к шасси, выполните следующие действия:

- Убедитесь в том, что питание не подключено к контроллеру и совместите его с шасси.
- Вставьте контроллер в соответствующий слот шасси согласно рисунку 9.2 и плотно надавите, чтобы убедиться в том, что контроллер и шасси соединены. Затяните два невыпадающих винта на передней панели контроллера с помощью отвертки типа Phillips No. 2.



1	Controller	3	Controller Slot
2	Captive Screws	4	Reconfigurable Embedded Chassis

Рисунок 9.2. Присоединение контроллера к шасси

1 – контроллер, 2 – невыпадающие винты, 3 – слот для контроллера, 4 – встраиваемое реконфигурируемое шасси

Подключение питания к контроллеру

Чтобы подать питание, подключите к системе соответствующий источник питания постоянного тока, например, напряжением 24 В. Подсоедините положительный вывод источника питания к клемме V1, и отрицательный вывод – к любой из С клемм. Некоторые контроллеры имеет дублированные входы источника питания, второй источник питания можно подключить к клемме V2.

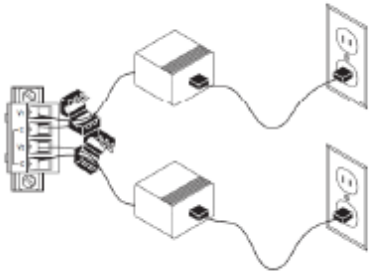


Рисунок 9.3. Подключение CompactRIO к источнику питания

Конфигурирование Dip- переключателей

Контроллер CompactRIO имеет пять Dip-переключателей– Safe Mode – Безопасный режим, Console Out – Консольный выход, IP Reset – сброс IP адреса, No App – без приложения, User1 – Пользовательский 1. National Instruments поставляет контроллер с переключателями в положении OFF (отключено). Переключатели должны оставаться в положении OFF при конфигурировании контроллера.

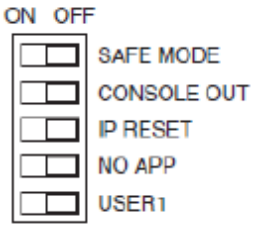


Рисунок 9.4. При конфигурировании контроллера CompactRIO оставляйте переключатели в положении OFF

Отдельные Dip-переключатели и их назначение рассматриваются в руководствах по эксплуатации и характеристиках CompactRIO.

Подключение системы CompactRIO к персональному компьютеру

Есть два способа подключения контроллера CompactRIO к вашему компьютеру:

1. Установить контроллер в ту же подсеть, что и компьютер, путем присоединения его к маршрутизатору или концентратору
2. Подсоединить систему CompactRIO непосредственно к компьютеру с помощью кабеля с перекрестными соединениями контактов (кроссовер)

Конфигурирование системы CompactRIO с помощью утилиты MAX

После того, как вы соединили все аппаратные средства, можно конфигурировать вашу систему CompactRIO с помощью утилиты Measurement & Automation Explorer (**Start»Programs»National Instruments»Measurement & Automation Explorer**), это необходимо для выбора целевого устройства и программирования в LabVIEW. Перед тем, как конфигурировать CompactRIO в MAX, необходимо инсталлировать на ваш главный (Host) компьютер LabVIEW Real-Time Module, NI-VISA и NI-RIO. Убедитесь в том, что вы устанавливаете последнюю версию NI-RIO.

При включении системы CompactRIO можно раскрыть раздел Remote systems (Удаленные системы) в MAX. Первоначально система CompactRIO появляется там с именем по умолчанию cRIO-[номер контроллера]. Вы можете изменить это имя в разделе настроек Identification (Идентификация). Когда вы выбираете систему в дереве навигации, на панели детализации отображаются настройки IP адреса для вашей системы CompactRIO. По умолчанию IP адрес – 0.0.0.0.

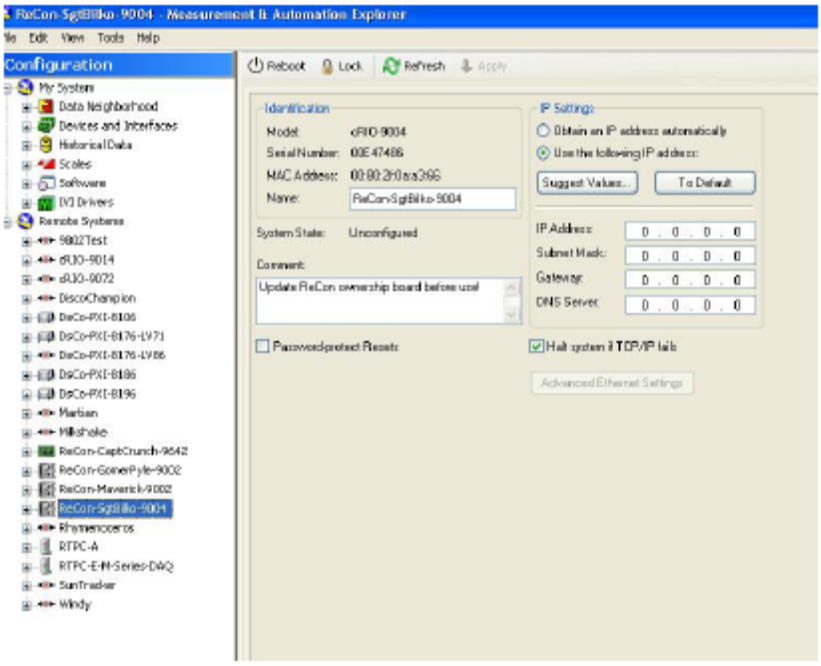


Рисунок 9.5. Настройки IP адреса по умолчанию – 0.0.0.0.

Если вы подключаете свою систему CompactRIO с помощью сетевого маршрутизатора, который поддерживает DHCP, то можете настроить систему так, чтобы она получала IP адрес автоматически. При использовании кабеля с перекрестным соединением контактов, вам необходимо присвоить вашему главному компьютеру статический IP адрес. Комитет по цифровым адресам в Интернете (Internet Assigned Numbers Authority – IANA) зарезервировал следующие зоны пространства IP адресов для частных объединенных сетей:

- 10.0.0.0 - 10.255.255.255
- 172.16.0.0 - 172.31.255.255
- 192.168.0.0 - 192.168.255.255
- 169.254.0.0 – 169.254.255.255 (локальные адреса для использования кабелей с перекрестными соединениями контактов)

Вы должны назначить уникальный IP адрес каждому компьютеру в вашей локальной сети. Например, вы можете назначить адрес 192.168.0.1 главному компьютеру (Host) и 192.168.0.3 – удаленной системе. Первоначально вам следует поместить главный компьютер и систему CompactRIO в одну и ту же подсеть до тех пор, пока вы не сконфигурируете их с помощью маски подсети. Чтобы найти маску подсети для вашего главного компьютера, наберите команду "**ipconfig**" в командной строке: **Start»Run»cmd**.

Чтобы сконфигурировать настройки TCP/IP для Host компьютера, выполните следующие действия:

1. Откройте сетевые соединения (доступные через панель управления).
2. Щелкните правой кнопкой мыши по значку **Подключение по локальной сети** и выберите команду **Свойства**. На закладке **Общие** в разделе **Компоненты, используемые этим подключением**: щелкните мышью по строке **Протокол Интернета (TCP/IP)**, а затем щелкните мышью по кнопке **Свойства**.
3. Чтобы задать IP адрес, щелкните по кнопке **Использовать следующий IP адрес** и заполните поля **IP-адрес** и **Маска подсети**.

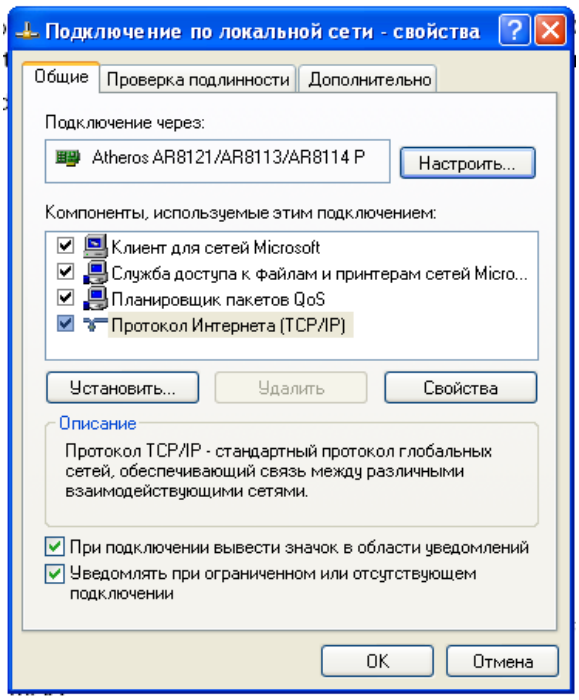


Рисунок 9.6. Конфигурирование настроек TCP/IP для Host компьютера

Этапы конфигурирования в MAX

Чтобы завершить конфигурирование системы CompactRIO с помощью утилиты MAX выполните следующие действия:

1. В разделе **IP Settings** (Настройки IP адреса) установите IP адрес:
   - Если соединение устанавливается с помощью маршрутизатора, который поддерживает DHCP, выберите **Obtain IP Address automatically** (Получить IP адрес автоматически).
   - Если используется кабель с перекрестным соединением контактов, выберите **Use the following IP address** (Использовать следующий IP адрес) и назначьте IP адрес системе CompactRIO с теми же настройками подсети и IP, что и ваш персональный компьютер.
2. Если необходимо введите новое имя системы CompactRIO в разделе Identification в поле **Name**. Чтобы изменения вступили в силу, щелкните по кнопке Apply. Если вы не ввели IP адрес в поле Gateway (Шлюз), то можете получить предупреждение о том, что система CompactRIO и шлюз находятся в разных подсетях. Чтобы игнорировать это неопасное предупреждение, щелкните мышью по кнопке Yes. Если же вы ввели адрес в поле Gateway, убедитесь, что он соответствует допустимому значению адреса шлюза по умолчанию и не равен 0.0.0.0. Равенство этого адреса 0.0.0.0 в MAX приводит к потере связи. Чтобы найти адрес шлюза по умолчанию для вашей сети, в командной строке наберите команду **ipconfig/all**.
3. Инсталлируйте программное обеспечение системы CompactRIO. При раскрытии CompactRIO в разделе Remote Systems становятся видимыми разделы Devices and Interfaces и Software. Щелкните правой кнопкой по разделу Software и выберите команду Add/Remove Software.

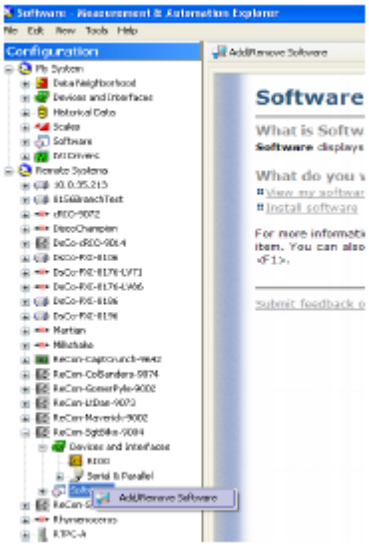


Рисунок 9.7. Инсталляция программного обеспечения на систему CopmactRIO

4. Затем вы увидите мастер с рекомендуемым набором программного обеспечения. Щелкните по кнопке **Next** и следуйте указаниям по инсталляции программного обеспечения. Системе CompactRIO требуется то же самое программное обеспечение, что и на вашем главном компьютере, с небольшим количеством дополнительных компонентов. Версия программного обеспечения на главном компьютере и системе CompactRIO должна быть одна и та же. После установки программного обеспечения система CompactRIO автоматически перезагружается.

Теперь вы готовы добавить вашу систему CompactRIO в проект LabVIEW.

Включение системы CompactRIO в проект LabVIEW

Чтобы создать проект с использованием RIO Scan Interface, выполните следующие действия:

1. В окне LabVIEW Getting Started откройте новый проект (Empty Project).

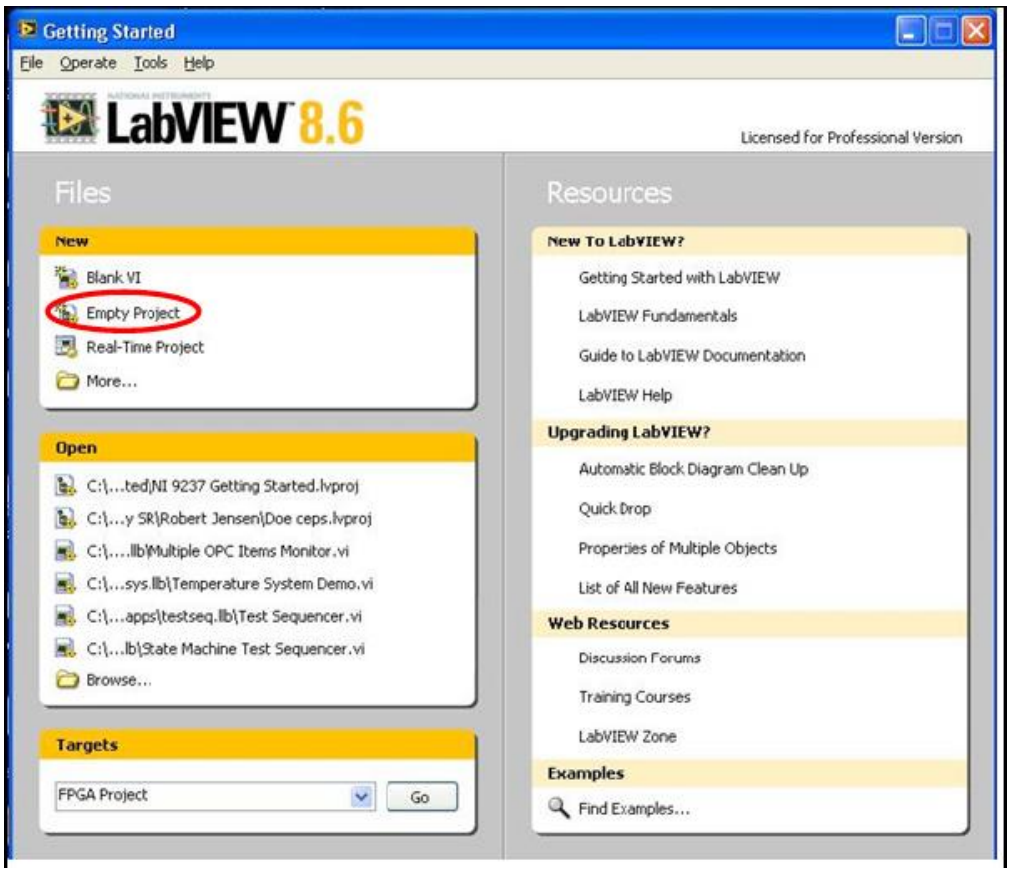


Рисунок 9.8. Открытие нового проекта (Empty Project) в окне LabVIEW Getting Started

2. Сохраните проект с определенным именем в какой-нибудь папке. Теперь щелкните правой кнопкой мыши по имени проекта и выберите команду **New»Targets and Devices**. Вы увидите всплывающее окно, где можете раскрыть раздел Real-Time CompactRIO для выбора вашего удаленного целевого устройства и добавить его в проект. Если у вас нет пока аппаратных средств, то можете выбрать пункт "New target or device" и вручную выбрать тип оборудования.

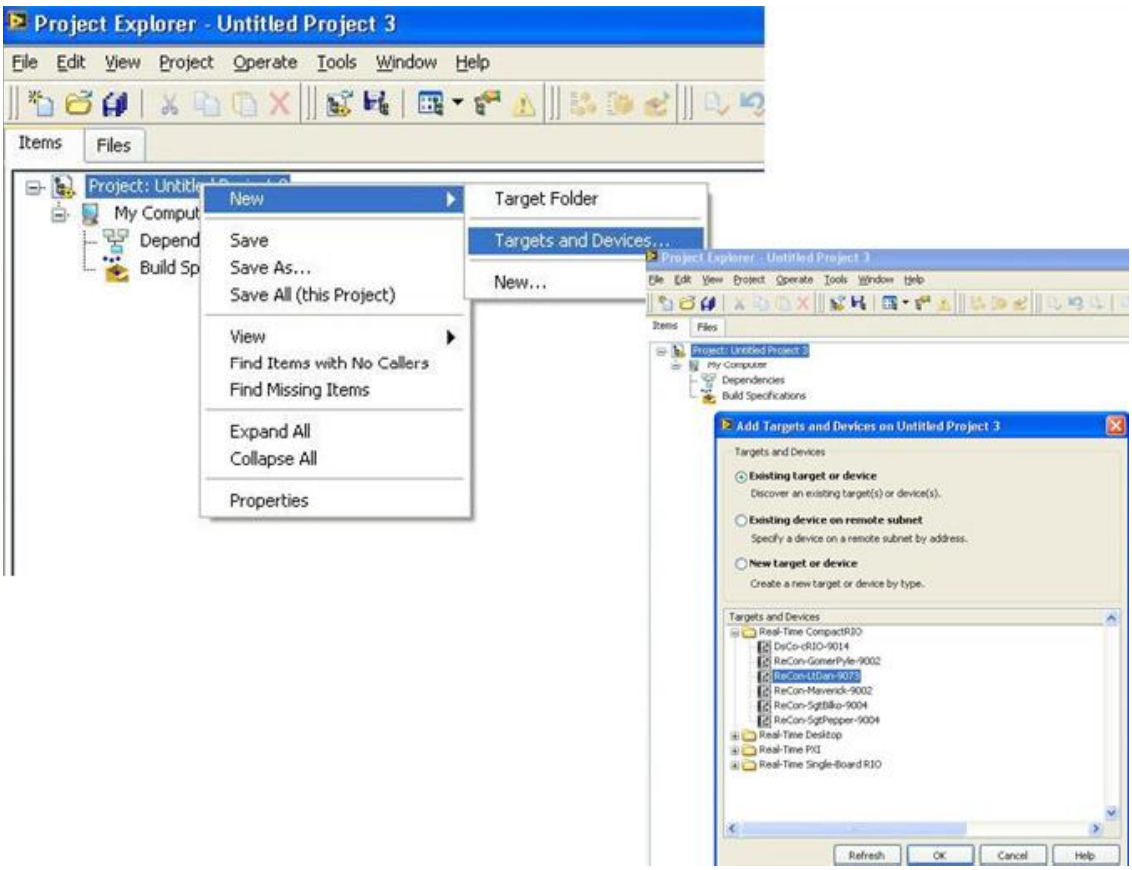


Рисунок 9.9. Выбор удаленной платформы и добавление ее в проект

3. Если ваш контроллер CompactRIO содержит модули, вы увидите диалоговое окно с вопросом, хотите ли вы добавить модули, использующие режим сканирования (CompactRIO Scan Mode) или режим FPGA Interface. Если вы выбираете режим Scan Mode, то шасси и модули добавляются в проект автоматически.

Теперь вы готовы проектировать программное приложение управления в реальном времени или начать с открытия примеров программ.

Изменение существующего проекта LabVIEW

Если вы решаете адаптировать пример программы таким образом, чтобы он работал с вашим удаленным целевым устройством, вы можете щелкнуть правой кнопкой мыши по целевым устройствам реального времени в примерах обозревателя проектов (LabVIEW Project Explorer) и выбрать команду Properties. На закладке General измените IP адрес в поле IP Address/DNS Name, чтобы он соответствовал вашей системе CompactRIO. Всегда сохраняйте копию примера путем выбора команды меню **File»Save As»Duplicate.lvproj file and contents** и сохранения всего содержимого проекта в отдельной папке.

ПРИЛОЖЕНИЕ В
Отладочные средства LabVIEW

Отладка приложений в процессе их разработки в LabVIEW

В LabVIEW есть множество средств, которые помогают разобраться в работе пользовательской программы в процессе ее разработки и редактирования. Эти средства спроектированы для использования в процессе исполнения программы, поэтому они могут повлиять на детерминизм работы приложения во время его отладки. В линейке инструментов блок-диаграммы можно найти следующие средства, которые являются хорошей отправной точкой при отладке приложений.

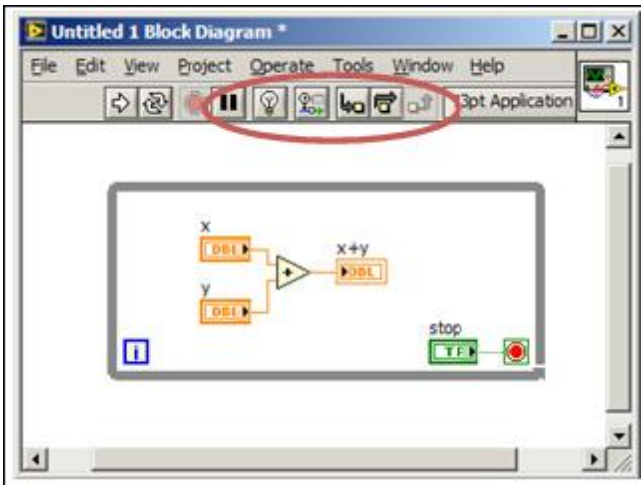


Рисунок 10.1. В LabVIEW есть разнообразные отладочные средства

Исполнение с подсветкой 

Исполнение с подсветкой оживляет блок-диаграмму – отслеживается поток данных, позволяя наблюдать промежуточные значения. Чтобы пронаблюдать исполнение блок-диаграммы с подсветкой, щелкните мышью по кнопке Highlight Execution  на панели инструментов. Затем щелкните мышью по белой стрелке запуска  , подсветка выполнения покажет перемещение данных на блок-диаграмме от одного узла к другому с помощью кружка, который двигается по проводникам. Подсветку можно использовать одновременно с пошаговым исполнением программы, чтобы видеть, как значения данных перемещаются от одного узла к другому через VI.

Пошаговое выполнение 

Пошаговое выполнение VI применяется для того, чтобы на блок-диаграмме просмотреть каждое действие VI в процессе его работы. Кнопки пошагового выполнения влияют на работу VI, только если VI или subVI исполняются в пошаговом режиме.

Задайте пошаговый режим щелчком мыши по кнопке Step Into или Step Over на линейке инструментов блок-диаграммы. Наводите курсор на кнопки Step Into (Шаг в …), Step Over (Шаг через …) или Step Out (Шаг из …), чтобы увидеть подсказку, которая описывает следующий шаг при щелчке на данную кнопку. Вы можете исполнять subVI в пошаговом или в обычном режиме.

При пошаговом выполнении VI мигание узлов показывает, что они готовы к исполнению. Если VI выполняется в пошаговом режиме с подсветкой, визуальный признак исполнения (кружок) появляется на иконках subVI, которые работают в текущий момент.

Точки прерывания 

Инструмент для работы с контрольными точками (Breakpoint tool – точка прерывания) служит для размещения контрольных точек на VI, узлах, проводниках и приостанавливает в этих местах

работу программы. Когда вы устанавливаете точку прерывания на проводнике, выполнение программы приостанавливается после того, как данные проходят через проводник, и кнопка паузы становится красной. Поместите контрольную точку на блок-диаграмму, чтобы приостановить выполнение после того, как все узлы на блок-диаграмме выполнятся. В этом случае граница блок-диаграммы становится красной и мигает, чтобы показать расположение точки прерывания.

Когда VI приостанавливается на контрольной точке, LabVIEW выводит блок-диаграмму на передний план и подсвечивает узел, проводник или строку скрипта, в которых находится эта точка. При наведении курсора на существующую контрольную точку черная область курсора, связанного с инструментом для работы с контрольными точками, становится белой.

Когда в процессе исполнения программы вы достигаете контрольную точку, VI приостанавливается и кнопка паузы становится красной. Кроме того, фон и граница VI начинают мигать. Теперь вы можете выполнить следующие действия:

- Выполнить VI в пошаговом режиме с помощью кнопок пошаговой отладки.
- Проверить промежуточные значения на пробниках, размещаемых на проводниках перед запуском VI.
- Проверить промежуточные значения на пробниках, размещаемых после запуска VI, если вы включили режим Retain Wire Values (Сохранение значений данных в проводниках).
- Проверить значения индикаторов и органов управления на лицевой панели.
- Щелкнуть мышью по кнопке паузы, чтобы продолжить исполнение VI до следующей точки прерывания или до тех пор, пока VI не закончит работу.

Разрешать, запрещать, удалять или искать существующие контрольные точки (точки прерывания) можно с помощью окна Breakpoint Manager, которое открывается с помощью команды меню View»Breakpoint Manager или щелчком правой кнопки мыши по объекту на блок-диаграмме и выбором команды контекстного меню Breakpoint»Breakpoint manager. Вы можете удалять точки прерывания по отдельности или все сразу по всей иерархии VI.

Пробник

Для контроля промежуточных значений в проводнике в процессе работы VI служит пробник. Этот инструмент следует использовать в случае, если у вас сложная блок-диаграмма, где выполняется последовательность операций, любая из которых может вернуть некорректные данные. Чтобы определить, есть ли некорректные данные и где они находятся, инструмент для работы с пробниками следует применять в сочетании с подсветкой выполнения, пошаговым выполнением и точками прерывания. Если данные доступны, пробник немедленно обновляется и отображает данные в процессе выполнения с подсветкой, в режиме пошагового выполнения или когда VI приостанавливается в контрольной точке. Когда выполнение приостанавливается в узле на точке прерывания или в процессе пошагового выполнения, вы также можете установить пробник на проводник, который только что отработал, чтобы посмотреть, какое значение прошло через этот проводник.

Запрет исполнения программного кода

Разработчики имеют возможность запрещать исполнение целых фрагментов графического программного кода аналогично "закомментированию" в каком-нибудь текстовом языке программирования. Запрет выполнения фрагментов программного кода без лишения работоспособности всей программы является полезной отладочной технологией для работы с подозрительным или ошибочным программным кодом. В LabVIEW такая возможность обеспечивается с помощью структуры Diagram Disable Structure (Структура запрета блок-диаграммы – рисунок 10.2). Каждая такая структура содержит два или более кадров – как минимум один из кадров содержит запрещенный код, который полностью игнорируется компилятором LabVIEW и не выполняется в процессе работы VI. Один кадр может содержать код, который, наоборот, выполняется в процессе работы VI.

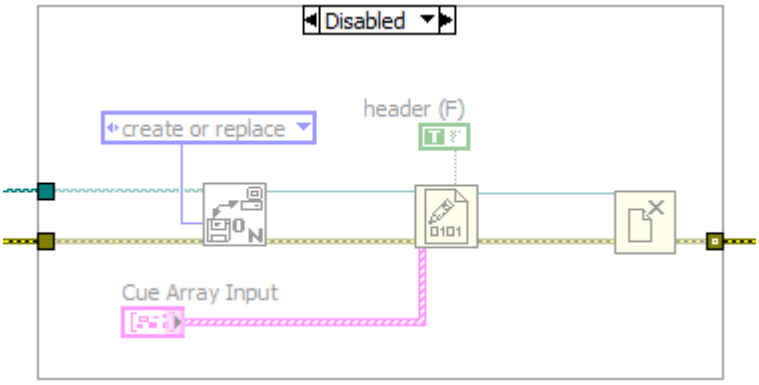


Рисунок 10.2. Структура Diagram Disable Structure в LabVIEW

Отладка развертываемых приложений

Диалоговое окно отладки приложений

С помощью LabVIEW можно отлаживать развертываемые встраиваемые приложения. Для отладки автономных приложений реального времени, работающих на целевых устройствах реального времени, используется диалоговое окно Debug Application or Shared Library (Отладка приложения или библиотеки общего пользования). С помощью данного диалогового окна вы можете просмотреть блок-диаграмму приложения, запустить приложение в режиме подсветки, пронаблюдать с помощью пробников состояния входов и выходов, а также использовать другие отладочные средства. Вы можете открыть окно отладки приложения или библиотеки общего пользования из окна Project Explorer, выбрав команду **Operate»Debug Application or Shared Library**.

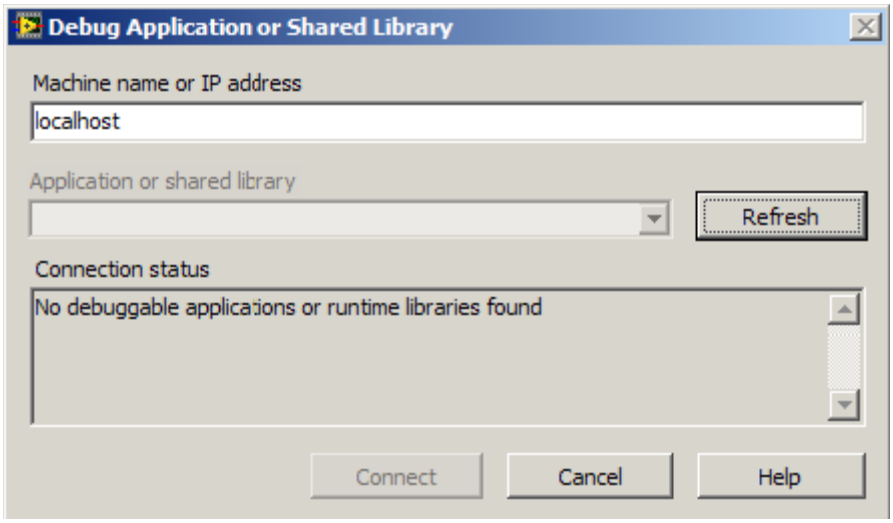


Рисунок 10.3. Окно Dialog Application or Shared Library

Чтобы разрешить отладку приложения, необходимо изменить характеристики построителя приложений внутри LabVIEW Application Builder. Выполните следующие действия, чтобы отладить приложение реального времени:

1. В построителе автономного приложения реального времени разрешите отладку, установив флажок Enable debugging на странице Advanced в диалоговом окне Real-Time Application Properties.
2. Щелкните правой кнопкой по спецификации построителя автономного приложения реального времени и выберите команду Build из контекстного меню для создания приложение.
3. Перезагрузите целевое устройство реального времени, чтобы запустить автономное приложение реального времени.
4. В окне Project Explorer выберите команду **Operate»Debug Application or Shared Library** для вызова диалогового окна Debug Application or Shared Library,.
5. Введите IP адрес целевого устройства реального времени в текстовое поле Machine name or IP address. Щелкните мышью по кнопке Refresh, чтобы просмотреть список автономных приложений реального времени, которые можно отлаживать в целевом устройстве реального времени.
6. Выберите автономное приложение реального времени, которое хотите отлаживать.
7. Щелкните мышью по кнопке Connect, чтобы открыть лицевую панель стартового VI для отладки. Используйте блок-диаграмму этого VI для отладки приложения с помощью пробников, точек прерывания и других отладочных технологий.
8. По окончании отладки закройте стартовый VI, который закрывает также автономное приложение реального времени на целевом устройстве реального времени и затем

перезагружает его, чтобы перезапустить приложение. Если вы хотите отключиться от целевого устройства, не закрывая VI, щелкните правой кнопкой мыши по его лицевой панели и выберите из контекстного меню команду **Debugging»Quit Debug Session** (Отладка»Выход из сеанса отладки).

NI Distributed System manager

NI Distributed System Manager (Менеджер распределенных систем) – это главное средство мониторинга систем в сети, просмотра и устранения ошибок, а также управления данными, публикуемыми в сети. NI Distributed System Manager является автономным средством, которое не требует, чтобы на этом же компьютере была установлена среда LabVIEW. NI Distributed System Manager можно открыть из стартового меню Windows или щелчком правой кнопки мыши по целевому устройству реального времени в окне Project Explorer, выбрав команду **Tools»Distributed System Manager.**

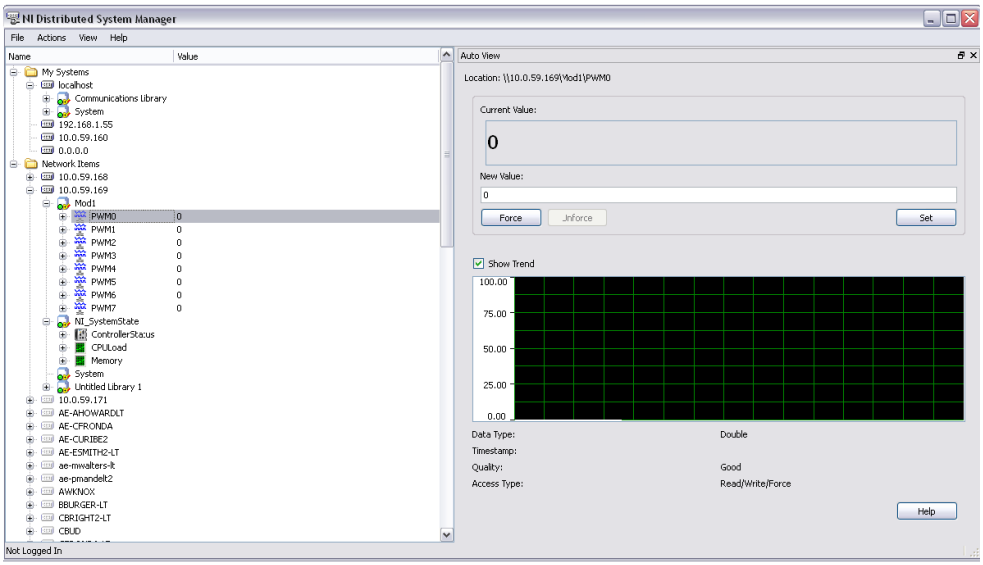


Рисунок 10.4. NI Distributed System Manager обеспечивает просмотр состояний подсистем ввода-вывода и контроллера через сеть

NI Distributed System Manager предоставляет тестовые панели для модулей CompactRIO, работающих в режиме сканирования. Как только ваша система становится доступной в сети, появляется доступ к значениям ввода-вывода в реальном времени и к предыдущим значениям за некоторый период. Таким образом вы можете быстро проверить подключения и целостность сигналов. Кроме тестовых панелей, данная утилита дает возможность контролировать использование памяти и загрузку процессора контроллеров CompactRIO. Вы можете также использовать NI Distributed System Manager для мониторинга и управления режимами механизма сканирования и ошибок в его работе.

Управляемые пользователем светодиодные индикаторы

Большинство контроллеров реального времени имеют один или более светодиодных индикаторов, которыми можно управлять из пользовательского приложения. Разработчик может использовать эти индикаторы для отображения состояния работающего приложения. Обычно на практике для отображения состояния приложения или текущего действия, которое оно выполняет, индикаторы мигают с различными частотами. В LabVIEW Real-Time есть VI RT LEDs, который предназначен для взаимодействия с пользовательским светодиодными индикаторами на целевых устройствах реального времени.

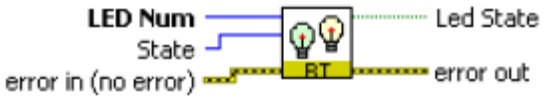


Рисунок 10.5. Светодиодные индикаторы позволяют в простой форме наблюдать за состоянием программы

Пользователь может изменять состояние индикаторов путем выполнения VI RT LEDs на блок-диаграмме и установки на входе State (Состояние) следующих значений:

0	Включение индикатора
1	Установление по умолчанию цвета №1
2	Установление по умолчанию цвета №2
3	Переключение между отключенным состоянием и цветом по умолчанию №1

Таблица 10.1. Значения на входе State у VI RT LEDs

Кроме того, некоторые контроллеры реального времени предоставляют доступ к светодиодным индикаторам из FPGA приложения. В этом случае LabVIEW FPGA приложение может быть спроектировано таким образом, чтобы использовался узел LabVIEW FPGA I/O, который имеет имя по умолчанию FPGA LED. Когда в этот узел производится запись, состояние индикатора изменятся в соответствии со значением элемента управления булевского типа.

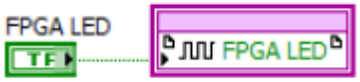


Рисунок 10.6. Вы можете получить доступ к светодиодным индикаторам из FPGA

Переключатель Console Out у контроллера CompactRIO

У контроллеров CompactRIO есть переключатель Console Out, который обеспечивает доступ к полезной диагностической информации через порт COM1 контроллера, который соединен с последовательным портом компьютера. Эта функциональная возможность особенно полезна для диагностирования систем в следующих вариантах действий:

- Отображение информации с контроллера на текстовой консоли в формате, принятом для оператора printf.
- Отображение текущей версии программного обеспечения контроллер и его IP адреса.
- Диагностирование контроллера, который ни на что не реагирует, или у которого светодиодные индикаторы показывают наличие ошибок.
- Диагностирование контроллера, который не обнаруживается в MAX.
- Диагностирование контроллера инженерами NI.

Чтобы просмотреть данные с консольного выхода Console Out на контроллере CompactRIO, выполните следующие действия:

1. Выключите подозреваемый на неисправность контроллер.
2. Подсоедините контроллер к персональному компьютеру с помощью нуль-модемным кабелем (контроллеры производства NI являются терминальными устройствами типа DTE) .
3. Примените следующие настройки для последовательного порта компьютера:
    a. Скорость (бит/с): 9600
    b. Биты данных: 8
    c. Четность: Нет
    d. Стоповые биты: 1
    e. Управление потоком: Нет
4. На контроллере установите DIP переключатель Console Out в положение ON (Включено).

5. Включите контроллер и посмотрите, что выводится из контроллера в терминальное окно компьютера.

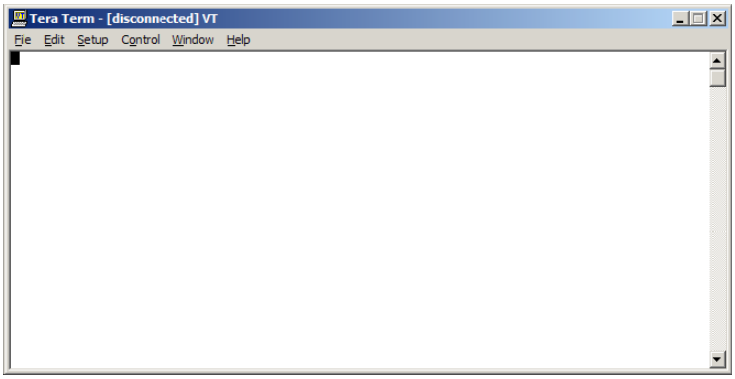


Рисунок 10.7. Выход Console Out обеспечивает простой последовательный интерфейс с контроллером

6. По завершении диагностики верните переключатель Console Out в положение OFF (Отключено).

Чтобы отправить свою собственную отладочную информацию через консольный выход, в LabVIEW Real-Time есть VI RT Debug String, который предназначен для передачи текстовой информации из контроллера в окно терминала на главном компьютере.

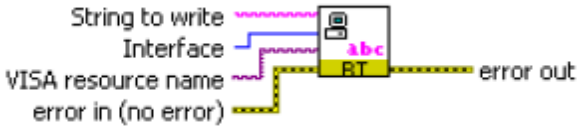


Рисунок 10.8. Вы можете настроить свое приложение таким образом, чтобы через выход Console Out в компьютер направлялись дополнительные данные

Регистрация ошибок

Поскольку у приложений CompactRIO чаще всего нет пользовательского интерфейса, они не могут сигнализировать о появлении ошибки. Поэтому важно, чтобы эти ошибки сохранялись в log-файле на контроллере, который позднее можно оттуда извлечь. Вы можете использовать log-файл для сохранения информации об ошибках LabVIEW и/или появления недопустимых данных. Регистрацию ошибок можно организовать в отдельном цикле низшего приоритета или сделать частью процедуры завершения работы приложения. Ниже показан простой вариант регистрации ошибок.

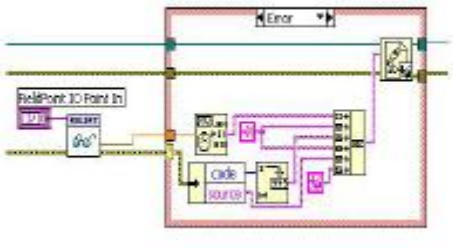


Рисунок 10.9. Простой вариант регистрации ошибок

Кроме того, контроллеры реального времени автоматически генерируют файл ошибок при отказе контроллера. Вы можете получить доступ к этому файлу щелчком правой кнопки мыши по имени контроллера в MAX в разделе Remote Systems, выбрав команду View Error Log.

Файл регистрации ошибок хранится на контроллере в папке /ni-rt/system/errlog.txt.

Средства анализа VI в LabVIEW

При построении приложения рекомендуется анализировать VI для вычисления времени их выполнения и выявления нежелательного распределения памяти или переключения потоков.

Анализ VI и разделы программного кода

Оценка быстродействия VI в реальном времени

Оценка производительности операции заключается в фиксации времени начала операции, выполнения операции и фиксации времени окончания операции. Поскольку в детерминированных приложениях очень важно выдерживать временные соотношения, в LabVIEW Real-Time имеется VI Real-Time Benchmarking, с помощью которого проверяется правильность работы приложения во времени.

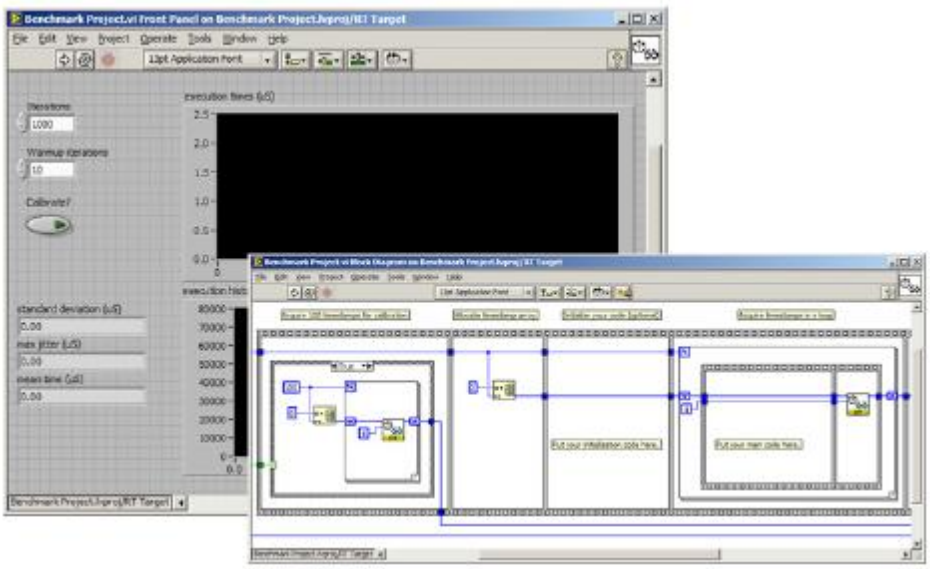


Рисунок 10.10. VI Real-Time Benchmark помогает оценить время выполнения subVI

В проекте оценки быстродействия VI и разделов VI, исполняемых в целевых устройствах реального времени используются VI RT GetTimestamp и VI Timestamp Analysis. Вы можете использовать информацию о производительности для оптимизации структуры VI, работающих в целевых устройствах реального времени. Найти примеры VI для оценки производительности в реальном времени можно с помощью навигатора примеров NI Example Finder.

Окно Profile Performance and Memory

Окно Profile Performance and Memory является средством статистического анализа расходования приложением времени выполнения и памяти. Это окно можно использовать для отображения информации о производительности всех VI и subVI, которые есть в памяти. Эта информация может помочь оптимизировать производительность VI путем обнаружения потенциально узких мест. Например, если вы заметили, что какой-то subVI выполняется слишком долго, то вы можете повысить его производительность.

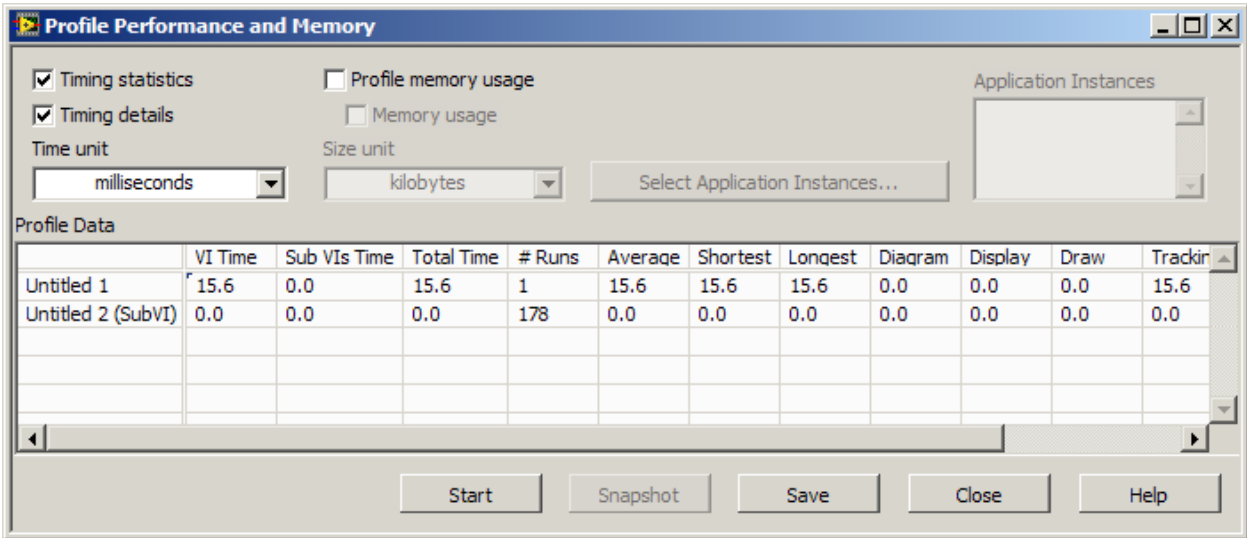


Рисунок 10.11. Окно Profile Performance and Memory дает детальную информацию о времени выполнения и объеме памяти для приложений и subVI

Чтобы запустить профайлер Profile Performance and Memory, выберите команду **Tools»Profile»Performance and Memory** (Инструменты»Профилирование» Производительность и память).

Мониторинг ресурсов целевого устройства в реальном времени

NI Distributed System Manager и Real-Time System Manager

Полезным является просмотр памяти и ресурсов процессора на контроллере, выполняющем развертываемый код. В некоторых случаях, сбои временных соотношений приложения обусловлены недостаточным объемом памяти или ресурсов процессора целевого устройства реального времени. Путем мониторинга ресурсов целевого устройства можно определить, происходит ли со временем "утечка памяти" (обычно она вызвана повторными назначениями памяти в цикле), или сколько времени выполняются различные циклы. NI Distributed System Manager отображает коэффициенты загрузки процессора и памяти, как показано на рисунке 10.12.

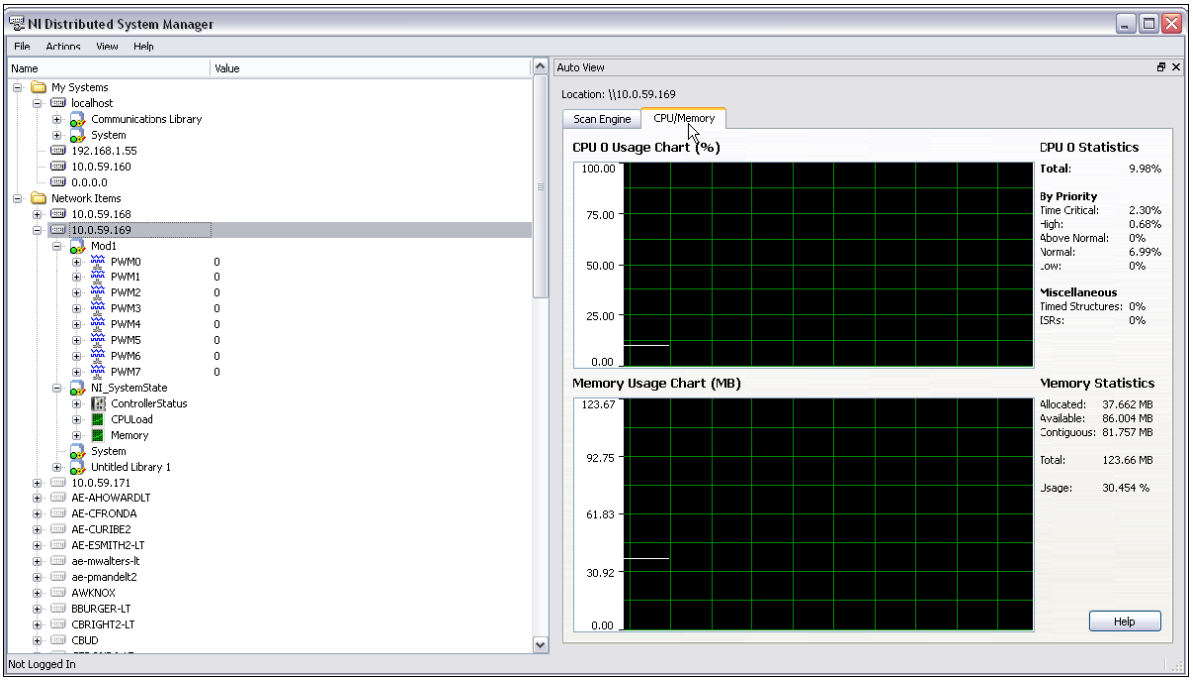


Рисунок 10.12. NI Distributed System Manager показывает коэффициенты загрузки процессора и памяти контроллеров в сети

Чтобы просматривать, как используется процессор и память, вы можете также использовать инструмент, называемый NI Real-Time System Manager (Менеджер систем реального времени).

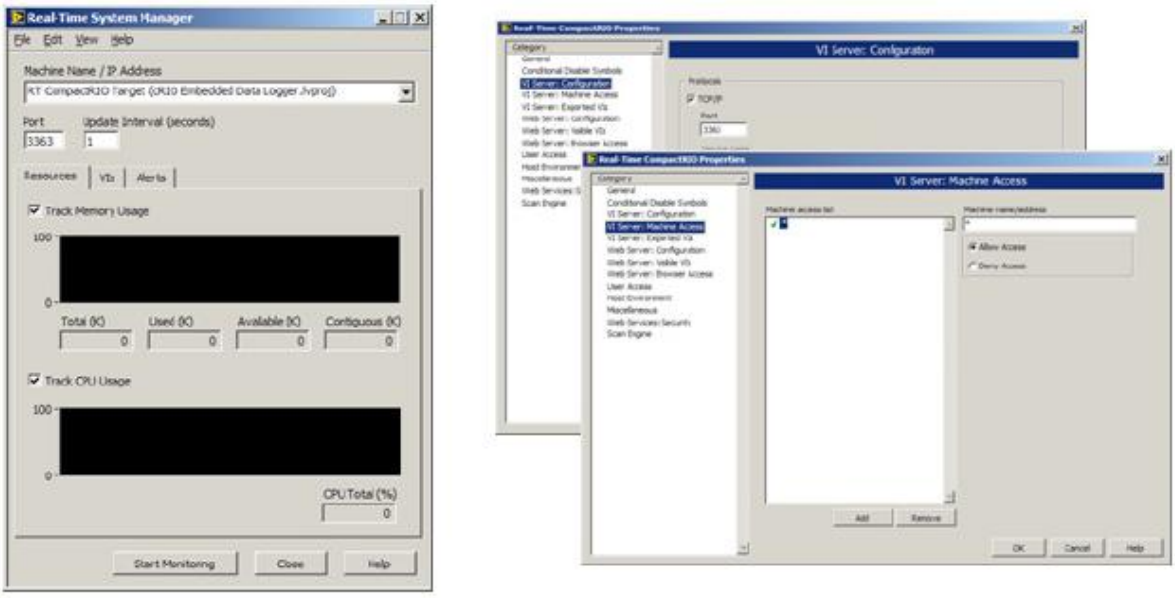


Рисунок 10.13. NI Real-Time System Manager также показывает, как загружены процессор и память

Вы можете запустить NI Real-Time system Manager (RTSM) из меню Tools»Real-Time Module. Чтобы сконфигурировать RTSM, необходимо сначала сконфигурировать утилиту VI Server на целевом устройстве. Для этого щелкните правой кнопкой по целевому устройству реального времени и выберите свойства, чтобы открыть страницу Real-Time Properties. Выделите пункт VI Server: Configuration и убедитесь, что флажок TCP/IP установлен и что флажки в разделе Accessible Server Resources (Доступные ресурсы сервера) также установлены. Убедитесь, что в разделе VI Server Machine Access указан IP адрес Host-компьютера или знак *.

Инструментальные средства Real-Time Execution Toolkit

Одним из лучших отладочных средств является Execution Trace Tool. Это инструмент трассировки выполнения программы и событий, который можно использовать для фиксации и отображения данных о временных соотношениях и событиях в VI, а также в потоках событий для приложений LabVIEW реального времени, выполняемых на целевых устройствах реального времени.

При минимальных изменениях встраиваемого программного кода эти средства отображают в графической форме ход выполнения многопотокового программного кода с высвечиванием переключения и блокировки потоков, а также распределения памяти. Используя эту информацию, можно оптимизировать программный код реального времени таким образом, чтобы он работал более детерминировано и циклы управления были более быстродействующими.

Чтобы воспользоваться описываемым инструментом, поместите Execution Trace Tool VI в приложение, работающее на целевом устройстве реального времени. Как только трассировка закончилась, эти VI посылают информацию обратно в пользовательский интерфейс Execution Trace Tool, который работает на Host-компьютере или может сохранить эту информацию на диске для последующего просмотра. Ниже приведен пример трассировки, где VI Trace Tool Start Trace располагается перед отлаживаемым программным кодом, а VI Trace Tool Stop Trace and Send посылает информацию Host-компьютере после того, как этот программный код отработает.

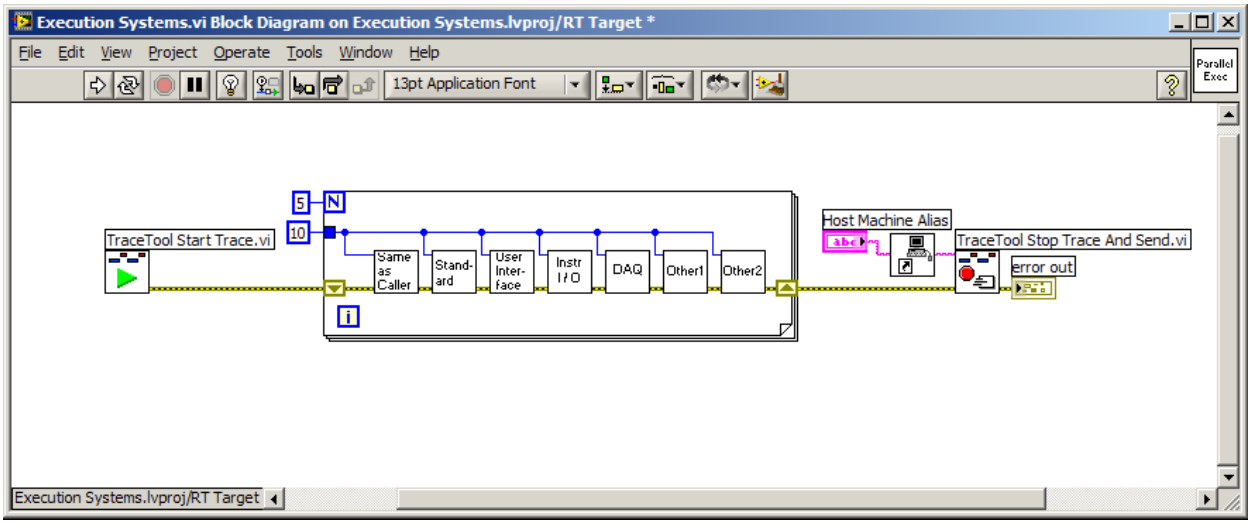


Рисунок 10.14. Изменяя программу таким образом, чтобы она начала сохранять трассировочные данные в файл регистрации и посылать его в компьютер, можно пронаблюдать детали низкого уровня при выполнении программы

Как только пользовательский интерфейс Execution Trace Tool получает информацию, вы видите одновременно два окна сессии трассировки. Окно потоков показывает, когда каждый из потоков активен. Окно VI отображает каждый VI, который находится в памяти и точно показывает, когда он работал.

Окно потоков показывает все потоки, которые запускаются в системе, включая потоков за пределами LabVIEW, такие, как хронирование в реальном времени и коммуникационные потоки. Для установления приоритетов VI в окне просмотра VI потоки LabVIEW, отображаемые в виде списка, кодируются цветами и обозначаются в соответствии с системой исполнения LabVIEW приложения, в которой они находятся. Срди отображаемых потоков можно наблюдать потоки операционной системы, синхронизируемый цикл и пользовательские флаги, которые, возможно, вы сконфигурировали.

Как упоминалось ранее, в окне VI отображается каждый VI, который находится в памяти и точно показывается, когда он исполнялся. Однако, два VI могут перекрываться в окне VI, но они не могут выполняться строго одновременно, поскольку один VI захватывает ресурсы. Один VI имеет более высокий приоритет, т.е. он останавливает VI более низкого приоритета, чтобы немедленно воспользоваться ресурсами процессора. Такое переключение происходит так быстро, что VI более низкого приоритета не может зарегистрировать свою остановку. Вы можете проверить окно потоков, чтобы определить, какой поток является активным и, следовательно, какой VI является активным.

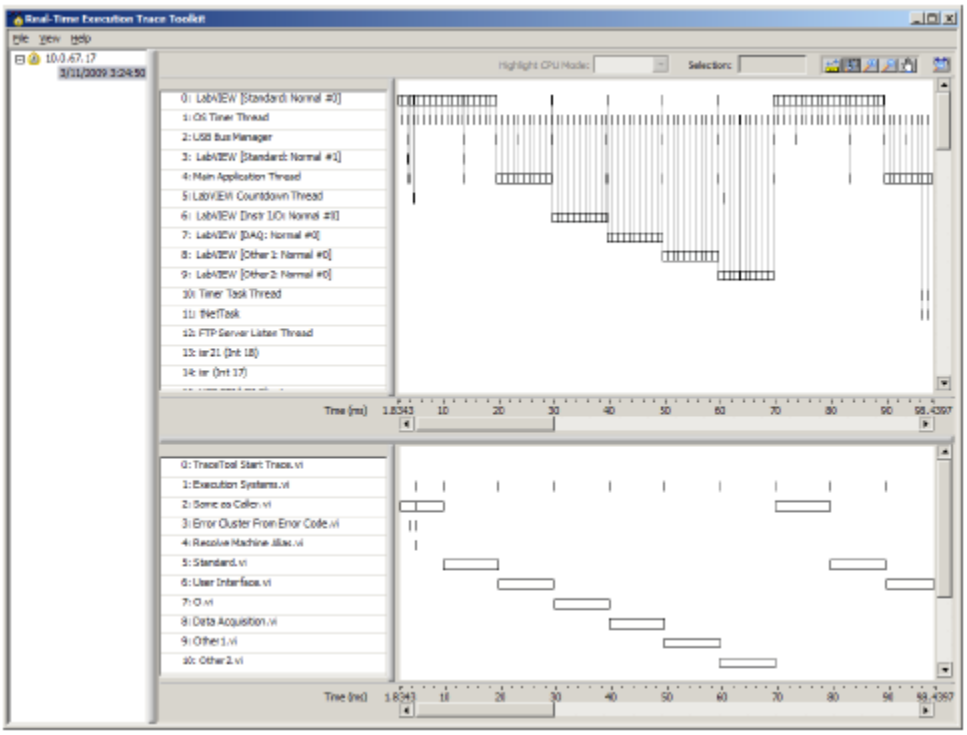


Рисунок 10.15. Execution Trace Tool обеспечивает низкоуровневую детализацию выполнения приложения, показывая выполнение потоков команд, а также захваты ресурсов и распределение памяти

Часто возникает желание увидеть какие-то другие события, кроме запуска и остановки VI. Эти события могут включать в себя доступ к менеджеру памяти, переход в режим ожидания, итерации синхронизируемого цикла (Timed Loop) и т.д. Чтобы обнаруживать такие события, можно использовать Real-Time Execution Trace Toolkit для конфигурирования флагов событий с учетом типа событий. Для этого в Execution Trace Tool нужно выполнить команду View»Configure Flags. За более детальной информацией обратитесь к справочной системе LabVIEW.

Отладка LabVIEW FPGA

Разработка приложения с помощью LabVIEW FPGA очень напоминает разработку приложения с помощью LabVIEW Real-Time. Одни и те же графические программные конструкции подходят для всех платформ. Функции LabVIEW FPGA совместимы с большей частью подмножества функций, которые есть в LabVIEW Real-Time. Тем не менее, чтобы оптимизировать LabVIEW FPGA приложение, можно применять различные технологии, усиливающие роль таких понятий, относящихся к аппаратным средствам, как регистры, конвейеризация, такты синхронизации и обмен информацией между асинхронными модулями.

Чем отличается разработка FPGA приложения от разработки приложения реального времени в LabVIEW Real-Time

Существует три главных различия технологий проектирования между аппаратными средствами FPGA и компьютерным программным обеспечением.

Время компиляции

Создание исполняемого кода для FPGA требует выполнения сложной компиляции, которая осуществляет оптимизацию верхнего уровня, в ходе которой согласуются все логические схемы на кристалле. В зависимости от сложности приложения, компиляция занимает большое количество времени.

Отсутствие в оборудовании типичных отладочных средств LabVIEW

С момента, когда программный код начинает работать внутри оборудования, традиционные программные отладочные средства, такие, как пробник, пошаговое выполнение, выполнение с высвечиванием, установка точек прерываний, более недоступны. Для упрощения отладки и разработки программы для FPGA приложения вы можете использовать технологии моделирования.

Устройства FPGA работают быстро и надежно

Устройства FPGA выбираются для приложений, где требуется высокое быстродействие и надежное функционирование (без прерываний). Впоследствии разработчиков часто интересует точная информация о том, что происходит на каждом такте синхронизации FPGA, как для оценки чистой производительности, так и для определения степени синхронизации между отдельными задачами.

Поскольку компиляция и работа программы в любом случае может занять от пяти минут до нескольких часов, метод программирования, заключающийся в написании кода и исправлении ошибок, является неприемлемым для практики. Чтобы стать более опытным программистом FPGA устройств, важно понять, как можно повысить эффективность проектирования. В нижеследующих параграфах рассматривается, как можно использовать поведенческие технологии моделирования наряду с отладкой после компиляции, чтобы создать как можно лучше протестированную и как можно более быстродействующую FPGA систему.

Технологии отладки LabVIEW FPGA приложения после компиляции

Применение для отладки органов управления/индикаторов

Когда программа работает на FPGA, хорошим отладочным механизмом являются органы управления и индикаторы. Поскольку вы не можете использовать пробники, которые есть в LabVIEW, для получения текущего значения интересующего проводника применяйте индикатор. Кроме того, в процессе отладки используйте вместо константы орган управления, поскольку изменение значения константы требует повторной компиляции приложения. С помощью органа управления ищите правильное значение, и если вы уверены в его корректности, заменяйте его на константу. Тем не менее, органы управления и индикаторы отнимают значительные объемы ресурсов FPGA, поскольку они создают интерфейс связи с VI на главном компьютере. Удаление ненужных органов управления и индикаторов является первым шагом оптимизации ресурсов. Дополнительные органы управления и индикаторы следует использовать только в процессе отладки, по завершении которой необходимо предусмотреть их удаление.

Фиксация ошибок

Часто бывает, что ошибки (сбойные ситуации) возникают только на короткое время. Наихудшим является случай, когда ошибка имеет место только в течение одного такта синхронизации. В

процессе отладки такие сбои происходят настолько быстро, что их нельзя выловить через интерфейс связи. Наилучшим способом обнаружения ошибки является фиксация условия, когда она возникает, и затем разрешение сброса, если необходимо. После того, как все стало работать правильно, можно убрать отладочную фиксацию и для инсталлированной системы обрабатывать ошибку другим путем. Для данного случая классическим примером является отладка периодического тайм-аута от очереди типа FIFO.

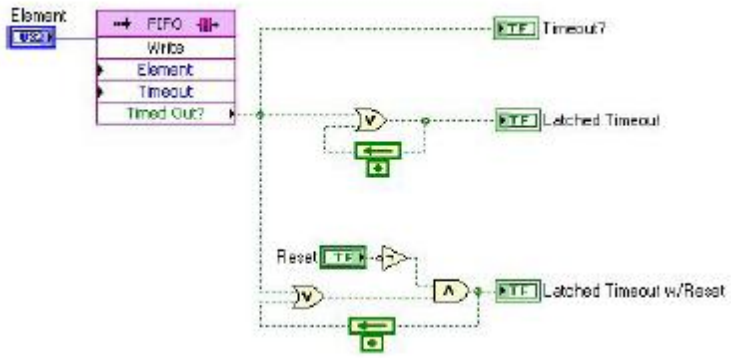


Рисунок 10.16. Данный VI демонстрирует фиксацию условия возникновения тайм-аута при наличии и при отсутствии сброса.

Регистрация данных в контрольной точке с помощью ПДП

Иногда вам хочется что-нибудь проверить пробником внутри FPGA, однако текущего значения органа управления для отладки недостаточно. В этом случае вы можете использовать для отладки канал ПДП, чтобы фактически захватить весь сигнал на основе данных в контрольной точке.

Подача сигнала с контрольной точки на устройство ввода-вывода

Независимо от того, сигнал в контрольной точке является аналоговым или цифровым, вы всегда можете в LabVIEW соединить проводник с узлом ввода-вывода. Как только сигнал из FPGA оказывается выведенным на узел ввода-вывода, с ним могут работать любые средства измерений производства NI такие как осциллограф, плата цифрового ввода-вывода, плата сбора данных (DAQ-плата) или даже еще одно FPGA устройство, запрограммированное для тестирования.

Тестирование нескольких вариантов реализации приложения с помощью Case-структуры

В некоторых случаях вам нужно протестировать несколько вариантов реализации задачи в FPGA. Вы можете скомпилировать каждый вариант, чтобы посмотреть, какой из них работает лучше. Однако, чтобы лишний раз ничего не компилировать, поместите варианты решения задачи в разные кадры case-структуры и выбирайте их на ходу. Тогда Вы можете мгновенно получить результат сравнения между вариантами и сэкономить время компиляции.

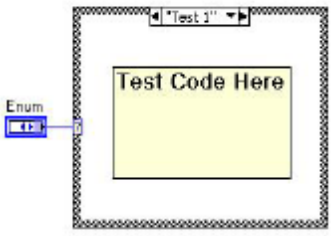


Рисунок 10.17. Поместите на блок-диаграмму case-структуры с различными вариантами программного кода, который Вы хотите испытать на FPGA

Поведенческое моделирование FPGA на компьютере разработчика

Поскольку вы используете программный код LabVIEW только при формировании логической структуры FPGA, всегда можно запустить ваши VI на главном компьютере. Это означает, что вы

можете использовать все отладочные средства LabVIEW, и вам не нужно каждый раз ждать окончания компиляции, когда вам нужно протестировать некоторую логическую схему. Кроме того, вы можете создать испытательный VI, чтобы определить входы, которые обычно соединены с внешним миром с помощью пользовательского устройства ввода-вывода, и захватить выходы для анализа и верификации. Наконец, вы можете запустить программу на главном компьютере одновременно с программой на FPGA, включая моделируемые регистры и буферы памяти типа FIFO с ПДП.

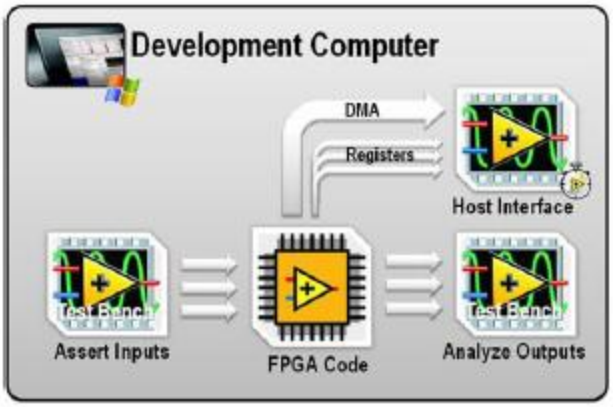


Рисунок 10.18. Концептуальная диаграмма моделирующей системы с испытательным VI, который сопрягает пользовательские входы и захваченные выходы с одновременно работающим с ним интерфейсом главного компьютера.

Чтобы заставить FPGA работать на компьютере разработчика, щелкните правой кнопкой мыши по строчке FPGA в проекте и выберите "Properties…". В разделе отладки у вас есть три возможных варианта.

1. **Execute VI on FPGA target (Выполнить VI на платформе FPGA)** – Если этот вариант оставить выбранным, процесс компиляции начнется по нажатию на кнопку "Run"  у FPGA VI.
2. **Execute VI on the development computer with simulated I/O** – Данный вариант настраивает FPGA VI на запуск на персональном компьютере по нажатию на кнопку  . В выпадающем списке вы можете выбрать как использование случайных данных (что преобладало в LabVIEW 8.5 и ранее) так и пользовательского ввода-вывода. Таким образом, вы можете разработать испытательный VI, который определяет входы и захватывает выходы.
3. **Execute VI on the development computer with real I/O** – На этот раз, данный вариант доступен только для встраиваемых устройств R серии. В этом случае VI запускается на компьютере и загружается его фиксированная версия в устройство R серии, чтобы дискретизировать ввод-вывод, когда программа исполняет узел ввода-вывода. Это является полезным для раннего тестирования и прототипирования, однако, имейте ввиду, что ввод-вывод осуществляет дискретизацию с программной синхронизацией и, скорее всего, не обеспечивает синхронизацию, которую вы действительно хотите применить для вашего VI.

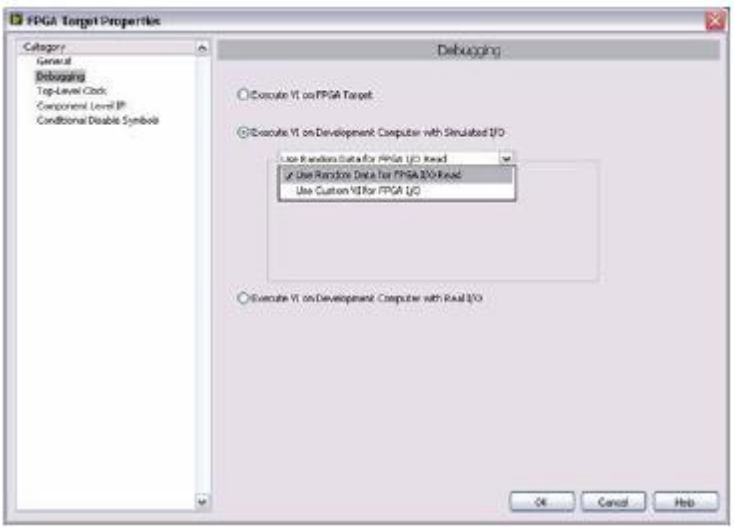


Рисунок 10.19. Свойства Windows для FPGA, которые показывают три опции для запуска программы для FPGA и типа ввода-вывода

ПРИЛОЖЕНИЕ С
Практический опыт разработки больших приложений

Рекомендации по разработке больших приложений

Правильные практические приемы проектирования программного обеспечения являются важными для успешной разработки управляющего приложения CompactRIO. Любой проект должен начинаться с четко определенных и документированных требований, на основе которых выбирается архитектура и затем начинается сама разработка. Далее проверяется правильность программного кода в отношении предъявленных требований и, наконец, этот программный код развертывается и сопровождается.

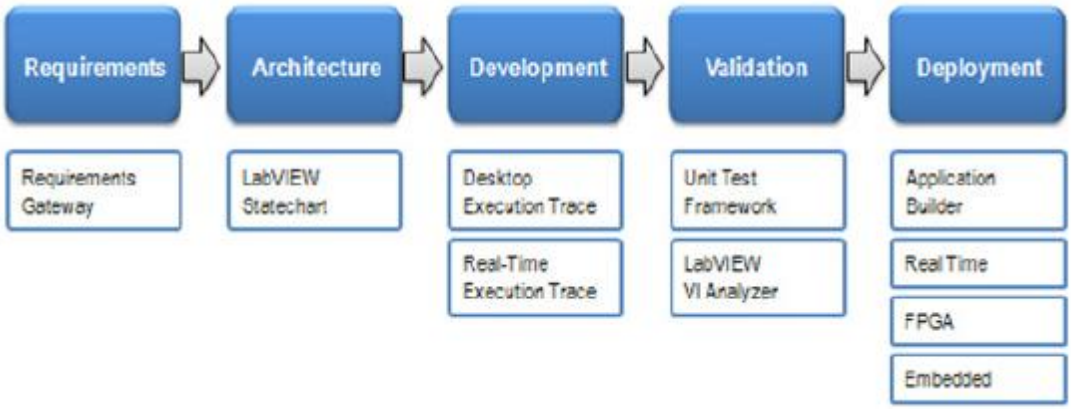


Рисунок 11.1. Правильные практические приемы проектирования программного обеспечения являются важными для успешного завершения проекта.
National Instruments располагает средствами, который могут оказаться полезными на каждом этапе проектирования

Requirements – требования, Architecture – архитектура, Validation – проверка правильности кода (приемочные испытания), Deployment – развертывание

Поскольку процесс создания приложений в LabVIEW весьма прост, многие инженеры сосредотачивают свое внимание только на этапе разработки. Они сразу начинают разрабатывать VI, уделяя сравнительно мало внимания планированию. Для простых приложений, таких, как легко реализуемые лабораторные испытания или приложения, мониторинга, такой подход может оказаться приемлемым. Однако для более сложных проектов хорошее планирование жизненно важно. В этом разделе дается представление о фундаментальных методах, используемых для успешной разработки программного обеспечения в LabVIEW. Чтобы получить более подробную информацию, почитайте руководство LabVIEW Development Guidelines Manual.

Применение структурного подхода к проектированию программ

Программисты, создающие ответственные приложения, в том числе встраиваемые системы управления, приложения промышленного мониторинга, высокопроизводительные испытательные системы, не могут позволить себе допускать ошибки или неопределенности в разрабатываемых ими системах. Для этих видов приложений процесс программирования должен быть хорошо структурированным и иногда подвергаться внешней сертификации, чтобы обеспечить качество и воспроизводимость. Вообще говоря, есть следующие ситуации, когда инженерам нужен структурный подход к программированию:

- Построение главным образом больших приложений

- Командная работа над программой, которая требует структурного подхода, чтобы обеспечить надежную и эффективную работу членов команды в различных частях проекта без конфликтных ситуаций
- Работа в промышленности или прикладной области, где требуется, чтобы процессы написания программного кода были сертифицированы правительственным агентством (в США – FDA (управление по контролю за продуктами и лекарствами), FAA (Федеральное авиационное агентство) и т.д.) или заказчиком (например, производителем автомобилей, работающим с поставщиками)

Многие инженеры могут подумать, что в таких случаях для разработки программного кода нужен традиционный текстовый язык программирования. В действительности, требование структурирования процесса разработки программного кода не зависит от используемого языка программирования или инструментальных средств. В данном параграфе кратко даются общие рекомендации, описываются встроенные средства и интеграция с внешними программными продуктами и демонстрируется, как инженеры могут построить крупномасштабные системы или системы для решения важных задач, которые требуют применения структурных подходов к программированию.

Сбор требований и управление

Ключевым шагом при разработке любой большой системы является сбор предварительных требований к системе до того, как началась сколько-нибудь серьезная разработка. Разработчики систем используют много различных методов сбора и документирования требований к системе, начиная с использования простых документов Microsoft Word или Excel, и заканчивая специальными средствами управления требованиями, такими как Telelogic DOORS или IBM Rational RequisitePro. Ключом к управлению требованиями является четкое документирование и установление связей между ними и программным кодом, который реализует каждое из них.

Чтобы содействовать этому процессу, в National Instruments создали продукт NI Requirements Gateway (Управление требованиями), с помощью которого инженеры могут ассоциировать свои требования с отдельными VI LabVIEW, чтобы отслеживать их реализацию. С помощью этого продукта инженеры могут:

- Документировать требования в текстовых файлах, а также файлах Microsoft Word, Excel или Project; **Telelogic DOORS**; **IBM RequisitePro** и других средств. Инженеры могут также пользоваться новым менеджером требований, который поставляется вместе с рассматриваемым набором инструментальных средств.
- Ассоциировать каждое требование с отдельными VI или разделами программного кода, чтобы было легко отслеживать, как каждое требование реализуется и тестируется.
- Отслеживать требования по мере их изменения. С помощью NI Requirement Gateway инженеры могут быстро увидеть изменение требования и получить оповещение о том, с какими VI ассоциируется это требование, а также, должна ли быть выполнена проверка, по-прежнему ли удовлетворяют эти VI данному требованию.

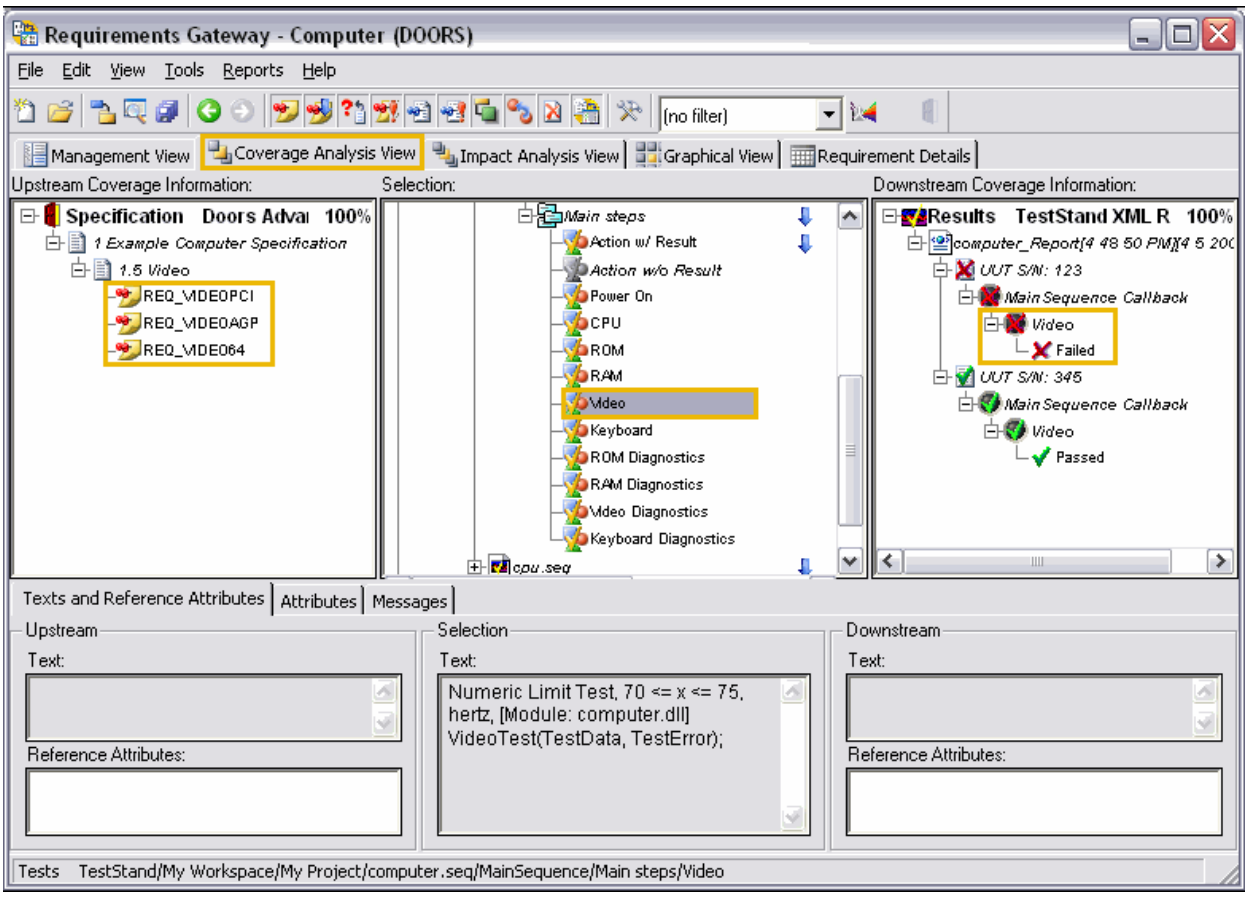


Рисунок 11.2. NI Requirements Gateway

Планирование архитектуры и проектирование

В настоящем документе представлена масштабируемая архитектура, которая подходит для многих управляющих приложений. Однако для успешной реализации архитектуры необходимо разработать структуру программного обеспечения таким образом, чтобы она удовлетворяла заданным требованиям. Ваш план должен определять компоненты системы и взаимодействие между ними. Вы должны потратить достаточное время на планирование того, как оптимально преобразовать требования к вашей системе в программный код.

Разработка больших приложений в LabVIEW, которая ничем не отличается от разработки приложения на любом другом языке программирования, влечет за собой деление приложения на логически завершенные фрагменты обозримого размера. Графическое программирование в LabVIEW может сделать нисходящее проектирование приложения простым и интуитивно понятным. Кроме графического программирования LabVIEW предоставляет средства создания приложений, которые включают в себя:

- Диаграммы состояний
- Диаграммы моделирования

Разработка

Организация файлов на диске

Файлы на диске должен быть своевременно структурирован. Плохое планирование больших приложений ведет к дополнительным затратам времени на перемещение и переименование файлов в процессе разработки. Поскольку каждый LabVIEW VI связан со своим именем и путем, то когда вы перемещаете или переименовываете subVI, то можете разорвать эти связи, которые затем должны быть вручную восстановлены. Правильное управление файлами на диске ослабляет риск перемещения большого количества файлов на более позднем этапе разработки приложения и может помочь разработчикам легко находить файлы и определять, куда сохранять новые файлы.

У многих разработчиков программ уже есть практические приемы и методы определения места, куда файлы должны быть сохранены. Хотя есть много признанных практических приемов и структур размещения файлов, приведенные ниже простые приемы показали себя подходящими для больших приложений:

- Сохранение всех файлов проекта в одной папке
- Создание папок внутри этой папки для группировки файлов по логическим признакам
- Группировка файлов в одной области памяти в соответствии с предварительно определенным критерием
- Разделение приложения на логические фрагменты разумного размера
- Использование соглашений при назначении логических и описательных имен программных объектов
- Отделение VI верхнего уровня от остального исходного программного кода

Папки на жестком диске обычно используются для группировки или разделения файлов по категориям. Таким образом subVI отделяются от вызывающих программ. Критерий, как правило, применяемый для группировки этих файлов, является сочетанием функционального назначения файла, его типа и иерархического уровня в приложении. Действительно, организация файлов на диске должна быть физическим отражением отношений между файлами и программным кодом приложения.

Управление файлами с помощью проекта LabVIEW

Проект LabVIEW предоставляет средства, которые помогают разработчикам управлять файлами. По мере разрастания приложения, разработчикам требуется управлять многочисленными файлами, связанными с приложением, такими как VI, файлы документации, библиотеки сторонних производителей, файлы данных и файлы настроек конфигурации аппаратных средств. Для управления этими файлами инженеры могут воспользоваться обозревателем проекта LabVIEW Project Explorer.

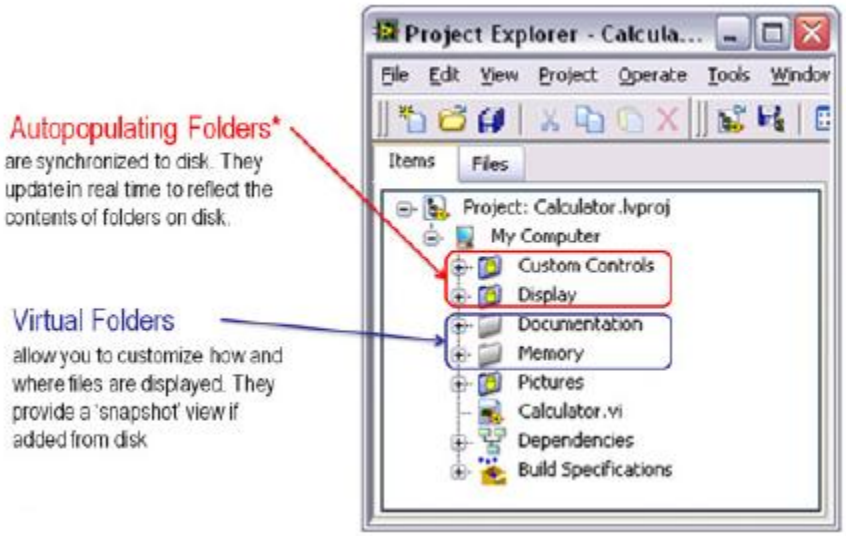


Рисунок 11.3. Проект LabVIEW

Autopopulating Folders - автоматически заполняющиеся папки синхронизированы с диском. Они обновляются в реальном времени, отражая содержимое папок на диске

Virtual Folders - виртуальные папки позволяют определять, как и где будут отображаться файлы. Эти папки дают возможность получить мгновенный обзор файлов, если они добавляются с диска.

В проекте LabVIEW разработчики могут использовать папки для организации всех файлов, из которых состоят разрабатываемые ими приложения. По умолчанию папки проекта являются виртуальными, однако разработчики могут синхронизировать их с физическими папками файловой системы. Добавив в некоторый момент папку в проект LabVIEW, разработчики могут сделать виртуальную папку автоматически заполняющейся (или наоборот), чтобы достичь максимальной гибкости организации и управления файлами. Автоматически заполняющиеся папки

синхронизированы с некоторой физической папкой, что означает, что в этих папках отражаются любые изменения, сделанные в физической папке за пределами среды проектирования LabVIEW. Автоматически заполняющиеся папки устанавливают связь между организацией файлов на диске и их логической структурой в проекте. Эти папки следует использовать всякий раз тогда, когда можно сохранить дисковую иерархию внутри LabVIEW Project Explorer.

Разработка приложений в команде

1. Часто несколько разработчиков совместно работают над одним и тем же проектом в LabVIEW. В таких случаях важно четко определить интерфейсы программного кода и установить стандарты программирования, чтобы гарантировать, что различные компоненты кода будут доступны для понимания и хорошо сработаются. Разработчики могут достичь этого за счет модульного программирования и предоставления стандартных интерфейсов.
2. Сохраняйте копии основных файлов проекта, в том числе VI, на одном сервере и организуйте управление исходным программным кодом.
3. Внедряйте стандарты программирования внутри группы разработчиков или компании

Модульное программирование в LabVIEW

Большие LabVIEW приложения требуют модульного подхода к программированию, при котором вся система подразделяется на логические компоненты. Несколько разработчиков, работающих над одним проектом, нуждаются в этом подходе, т.к. им нужно четко распределить ответственность за разработку программы. По определению, LabVIEW является языком модульного программирования. Каждый VI в LabVIEW (или модуль кода) может работать как автономное приложение или вызываться другим VI в качестве subVI. SubVI соответствует вызываемой подпрограмме в текстовых языках программирования или функции/функциональному блоку стандарта программирования 1131. Использование модульных subVI как части более крупных приложений упрощает блок-диаграмму VI верхнего уровня, помогает управлять изменениями и отлаживать системы.

Совместный доступ к программному коду с помощью программных средств управления исходным кодом

В начале работы над проектом по разработке программного обеспечения инженеры должны организовать процесс работы над изменениями проекта и разделить задания. Этот процесс важен для проектов, над которыми совместно работает много разработчиков. Средства управления исходным кодом являются наилучшим решением проблемы совместного доступа к программному коду и управления доступом, чтобы избежать случайных потерь данных. Эти средства отслеживают изменения программных модулей и помогают организовать совместный доступ к файлу разным пользователям в разных программных проектах. Кроме поддержки исходного кода типа VI, средства управления исходным кодом могут управлять и другими составляющими программного проекта, такими как спецификации свойств и другие документы.

LabVIEW поддерживает интеграцию со многими стандартными промышленными системами управления исходным кодом, такими как Microsoft Visual SourceSave, Perforce, IBM Rational ClearCase, PVCS Version Manager, MKS Source Integrity, а также бесплатными средствами с открытым исходным кодом типа CVS. Используя подобную интеграцию разработчики могут проверять файлы, как при наличии, так и отсутствии контроля исходного кода из среды LabVIEW, а также получать историю версий файлов. Чтобы проверить VI внутри LabVIEW или за пределами LabVIEW после того, как провайдер системы контроля исходного кода был сконфигурирован, разработчик просто щелкает правой кнопкой мыши по файлу в LabVIEW Project Explorer и выбирает из контекстного меню соответствующее действие.

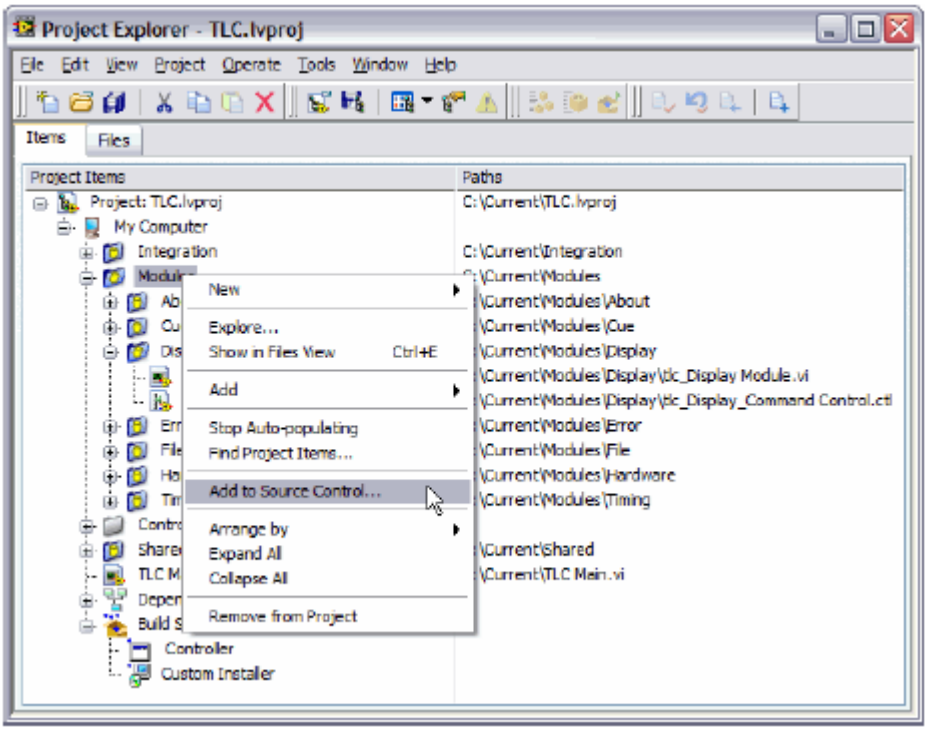


Рисунок 11.4. Опции контроля исходного кода в проекте LabVIEW

Отладка

Возможности отладки являются ключевыми в среде разработки приложений. В LabVIEW есть ряд отладочных средств, в том числе:

- Выполнение с подсветкой, при котором показывается движение данных на блок-диаграмме от одного узла к другому с помощью кружков, перемещаемых по проводам
- Пошаговое выполнение, при котором VI выполняется в пошаговом режиме, и на блок-диаграмме показывается каждое действие, которое выполняет VI в процессе работы
- Контрольные точки (точки прерывания), которые можно поместить на VI, узле или проводнике на блок-диаграмме, чтобы в этом месте приостановить выполнение программы
- Пробники, которые отображают промежуточные значения проводника в процессе работы VI

Более детальному рассмотрению методов и средств отладки приложений LabVIEW Real Time и LabVIEW FPGA посвящен отдельный параграф настоящего документа.

Проверка и подтверждение правильности программного кода

У разработчиков, которые используют LabVIEW для создания больших приложений, поощряется предоставление разработанного программного кода для дружеской проверки. Когда разработчик принимается за проверку программного кода, он сопровождает проверяющего по основному пути выполнения программы и отвечает на любые вопросы. Темами для обсуждения могут быть, насколько легко при существующей программной архитектуре добавлять новое свойство или вносить изменения, каким образом выводятся сообщения об ошибках и как они обрабатываются или достаточно ли хорошо программа разбита на модули.

Чтобы подготовиться к проверке программного кода, разработчикам следует воспользоваться преимуществом средств, которые могут помочь автоматизировать проверку программы и найти улучшения. Одним из примеров таких средств является **LabVIEW Analyzer,** который анализирует программный код LabVIEW и в шаг за шагом ведет пользователя через обнаруженные в процессе тестирования ошибки. VI Analyzer содержит более 60 тестов, которые касаются широкого спектра нарушений стиля и занижения производительности. С помощью VI Analyzer разработчики могут улучшить производительность VI, удобство работы с ним и его сопровождения. Кроме того, VI

Analyzer генерирует отчеты, которые позволяют разработчикам контролировать совершенствование программы в течение времени.

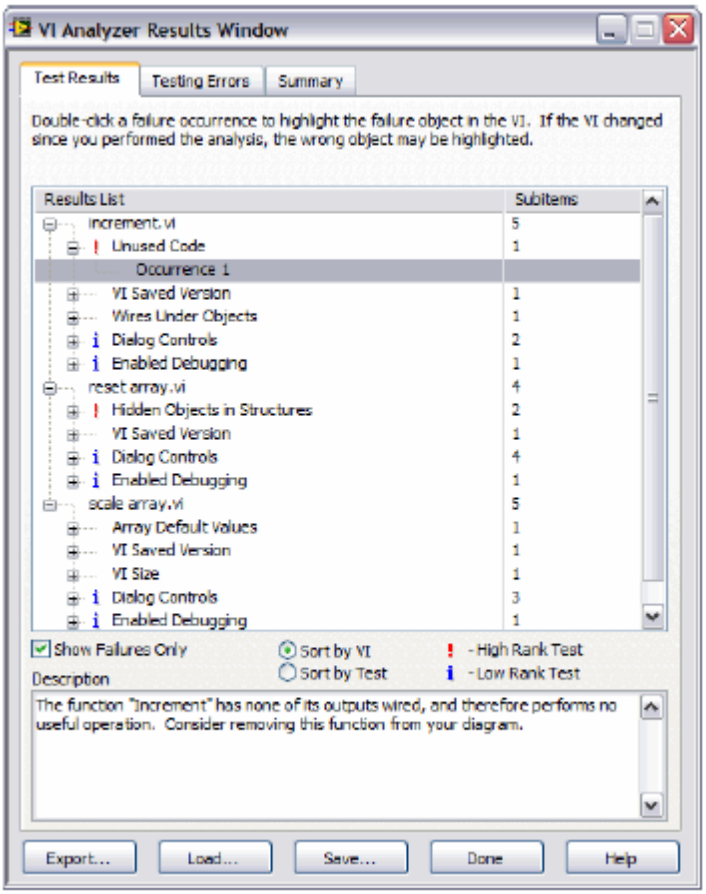


Рисунок 11.5. LabVIEW VI Analyzer

Развертывание

Последним этапом разработки управляющего приложения является развертывание его во встраиваемой системе и настройка его на автоматический запуск. Чтобы сделать LabVIEW VI запускаемым автоматически, создайте exe файл для целевого устройства реального времени. Это делается в проекте LabVIEW с помощью спецификации построителя, которая включает в себя всю информацию об исполняемом файле, а также любых вспомогательных файлах, которые вы хотите экспортировать. Спецификация построения облегчает восстановление или изменение режима построения исполняемого файла. В LabVIEW также есть средства для тиражирования системы, которые можно использовать для копирования встраиваемого в контроллер кода и тиражирования его на другую систему.