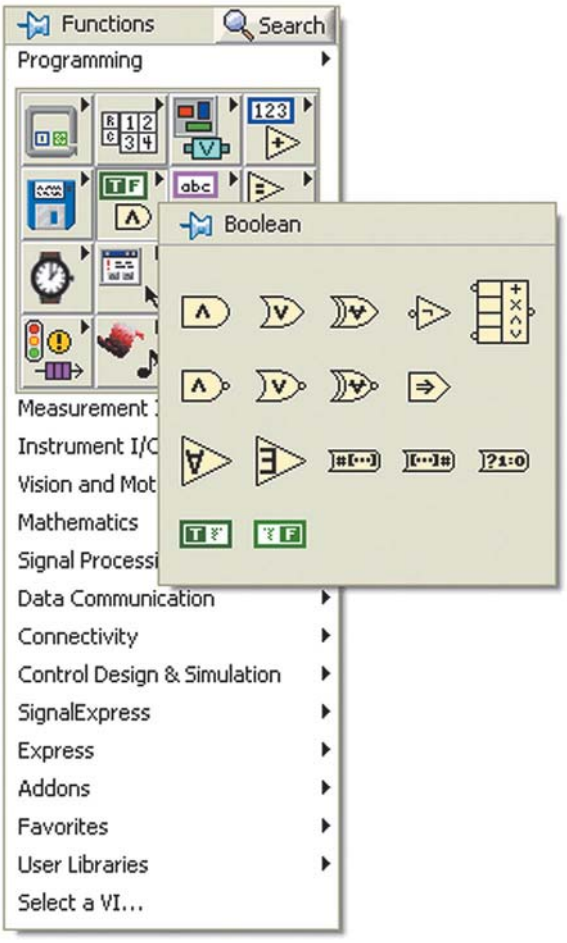
Учебный практикум по LabVIEW

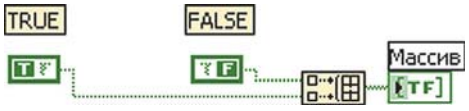
Ранее неоднократно фигурировало описание данных (констант, переменных) булевого типа. Помните соединительные проводники зеленого цвета на блок-диаграмме и индикаторы-"лампочки" на передней панели? Давайте-ка рассмотрим все это структурировано и более подробно. Скажем так, применяя системный подход. И так – булева логика, основные концепции и представление в LabVIEW, а также интересные примеры.

Как и во всех текстовых языках программирования (да и не только в них), в LabVIEW данные булевого типа могут принимать лишь значения TRUE либо FALSE (истина либо ложь, логические - 0 или 1).

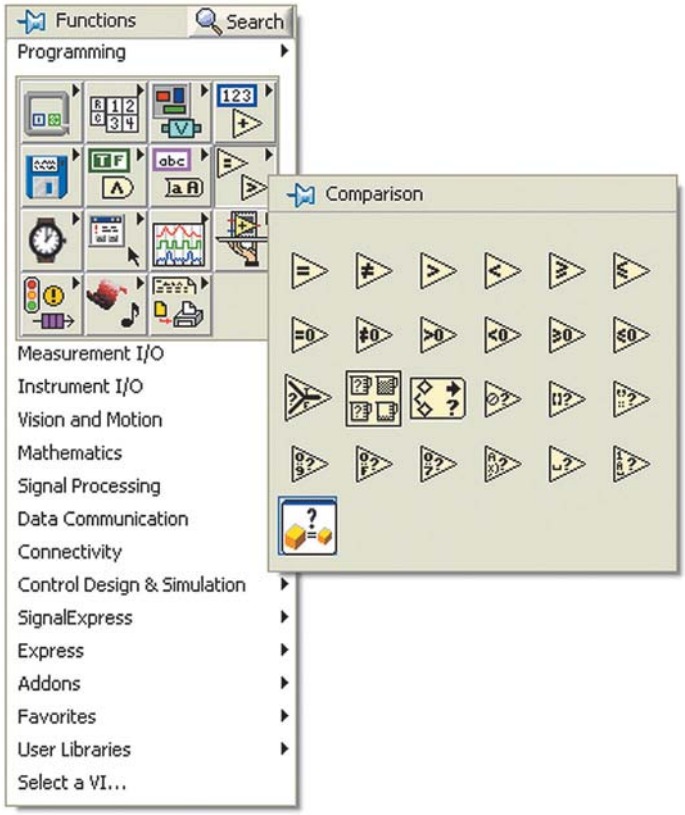
Подпалитра ***Логические функции (Boolean)*** имеет следующий вид:



На блок-диаграмме проще всего задавать булевские значения константами:



Если Вы по ошибке выбрали из подпалитры "не ту" константу, нет надобности возвращаться в ***Логические функции***, достаточно просто перевести указатель мыши в режим ***Управление (Operate value)*** (используя давно Вам знакомую палитру ***Инструменты (Toolbox)***) и нажать на уже присутствующую на блок-диаграмме пиктрограммку константы. Убедитесь самостоятельно в том, что ее значение меняется при этом на противоположное. Собственно, оперирование только лишь константами встречается в программировании относительно редко. Чаще булевские данные получают косвенно, например, анализ попадания значения переменной в интервал, либо выход за его границы. Именно для этих целей предназначены элементы подпалитры ***Сравнение (Comparison)***:

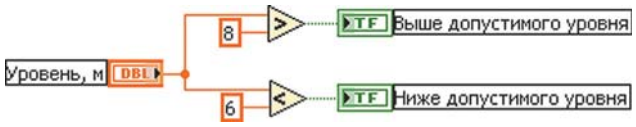


Как правило, выходы функций данной палитры имеют тип ***Boolean***. Исключения будут рассмотрены отдельно.

Функции, представленные в верхних двух рядах подпалитры ***(Equal?, Not equal?, Greater?, Less? и др.)***

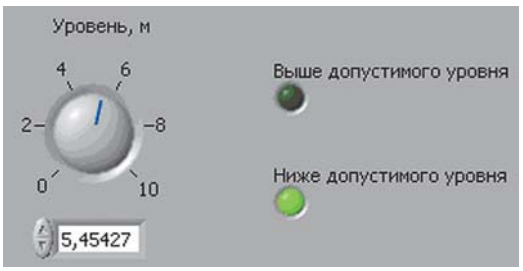
позволяют проверять разнообразные равенства либо неравенства между двумя однотипными константами либо переменными. Тип проверяемого условия понятен из изображения самой функции.

Рассмотрим такой пример. Предположим, в рамках работы некоторой системы мониторинга необходимо проверить, попадает ли уровень жидкости в баке в некоторый заданный интервал, заданный соответствующими уставками. Необходимо сигнализировать о выходе значения уровня жидкости за какую-либо из границ диапазона включением соответствующего светового индикатора. Эту процедуру весьма несложно реализовать с помощью следующего фрагмента кода LabVIEW:



Переменная ***Уровень*** содержит значение, полученное с соответствующего датчика. Допустимый диапазон уровней - от 6-ти до 8-ми.

Ну, а представление внешнего вида модели на интерфейсной панели может выглядеть примерно так:



Заметим, что также возможно выполнять и сравнение массивов. При этом режим сравнения выбирается с помощью пункта ***Comparison mode*** из выпадающего при нажатии на компаратор правой клавишей манипулятора меню:

Comparison elements - происходит поэлементное сравнение и на выходе функции, получаем массив соответствующих булевских значений;

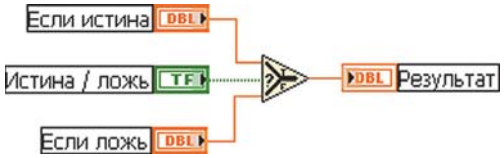
Comparison aggregates - происходит сравнение совокупности, на выходе - одиночное значение.

Также с помощью представленных функций можно сравнивать пути ***(Path)***, ссылки ***(Refnum)***, матрицы. Эти упражнения можно порекомендовать для самостоятельной проработки.

Функция ***Максимум и Минимум (Max & min)*** возвращает не булевские значения, а численные. И тем не менее, она представлена в подпалитре ***Сравнение***. Причина этого становится понятной сразу же после изучения ее принципа действия.

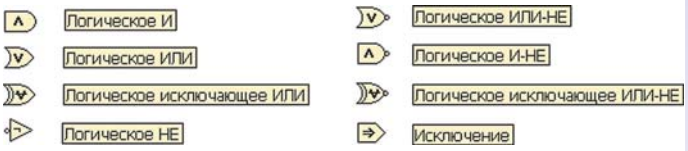
На входы функции необходимо подать два значения: ***x*** и ***y***. Функция возвращает максимальное и минимальное из них по каждому из выходов. Если на входы подавать массивы, а не скалярные переменные, выходы также будут содержать массивы из минимальных и максимальных элементов (в результате попарного их сравнения).

Функция ***Выбор (Select)*** работает по принципу ключа для двух состояний - на ее выход с именем ***Результат*** выдается значение одного из входов ***Если истина***, ***Если ложь*** в зависимости от состояния управляющего входа ***Истина / ложь***. Согласитесь, некоторое сходство с реле или транзисторным ключом явно присутствует!



После получения данных булевого типа с использованием функций из подпалитры ***Сравнение***, либо их задания с помощью констант, рассмотрим основные правила их обработки. Для этого используются элементы подпалитры ***Логические функции***. Их входы и выходы в большинстве случаев содержат данные именно такого типа.

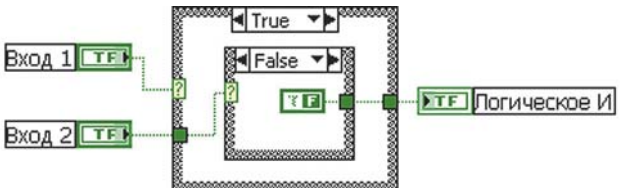
Простейшие логические функции (***И***, ***ИЛИ***, ***Исключающее ИЛИ***, ***НЕ***), а также некоторые их комбинации (***И-НЕ***, ***ИЛИ-НЕ***, ***Исключающее ИЛИ-НЕ***) представлены соответствующими функциями в LabVIEW:



За исключением функции ***Логического НЕ***, каждая из них имеет два входа и выход, содержащий результат в соответствии с таблицей истинности для данной функции.

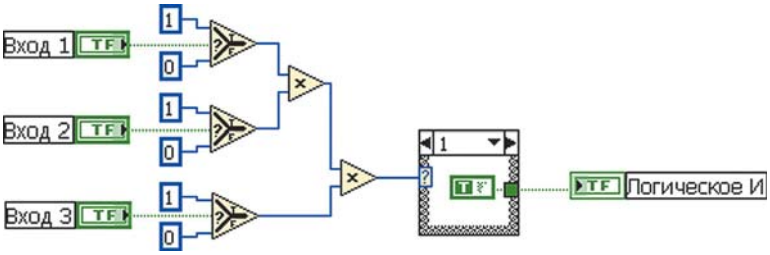
Функция ***Логическое НЕ*** имеет один вход и один выход - она производит простую инверсию поступивших данных.

В принципе, каждую из представленных здесь функций можно реализовать с использованием распространенных компонентов блок-диаграммы LabVIEW. Например, реализацию ***Логического И*** можно осуществить следующим образом:



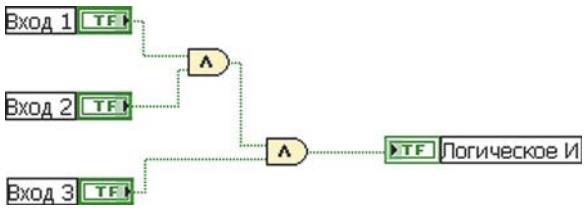
*! Значение констант, находящихся во внутреннем **Варианте** необходимо подбирать исходя из таблицы истинности реализуемой функции.*

Если входных переменных более чем две, то объем таблицы истинности резко возрастает, соответственно возрастет и количество приведенных на последнем рисунке констант. В этом случае удобнее пользоваться числовыми функциями. Например, ***Логическое И***, содержащее три входа, будет выглядеть примерно следующим образом:

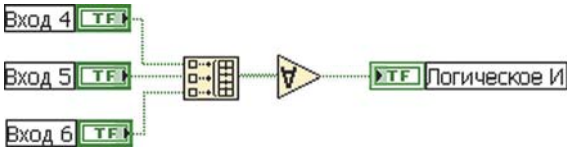


Два приведенных примера свидетельствуют лишь о том, что возможности использования стандартных функций в среде LabVIEW являются практически безграничными.

Если Вам необходимо реализовать функцию ***Логическое И*** на три входа, это можно сделать либо используя стандартные "двухвходовки", например:



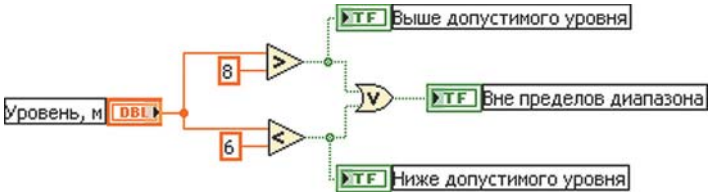
либо специальную функцию ***Логическое И для элементов массива (And array elements)***:



Преимущества данной функции становятся отчетливо видны при дальнейшем увеличении количества входов. Их достаточно просто предварительно "собрать" в массив.

Логическая функция ИЛИ для элементов массива (Or array elements) работает аналогичным образом:

Продолжая рассмотрение системы мониторинга жидкости, немного усложним поставленную ранее задачу. Предположим, необходимо сигнализировать про выход уровня жидкости за любую из установленных границ, т.е. реализовать общий случай сигнализации. Осуществим поставленную задачу. Используем из предыдущего программного кода две булевские переменные, содержащие данные о выходе уровня жидкости за каждую из границ. Очевидно, что для получения результата над ними необходимо провести операцию "логическое ИЛИ". Вот как это будет выглядеть:



Индикатору ***"Вне пределов диапазона"*** на лицевой панели придадим несколько более внушительный вид для улучшения визуализации работы. Ведь в общем случае система должна "позвать" оператора на помощь именно при выходе уровня за любую из границ:

Кому-то данный пример покажется тривиальным. Но совершенно очевидно, что закрывать глаза на использование разнообразных логических функций при построении подобных систем, да и при решении других инженерных задач, не следует, даже наоборот. Критерий необходимости их использования - количество сэкономленного при этом времени.

Наконец, подпалитра ***Логические функции*** содержит три функции перехода от булевских данных к арифметическим и наоборот.

Функция ***Число в логический массив (Number to boolean array)*** фактически выполняет преобразование беззнакового числа в его двоичное представление:

При этом возможна подача на вход функции не только 32-битного числа, как показано на диаграмме, но и чисел меньших размеров (16 и 8 бит). Как Вы наверняка догадались, размер выходного ***Логического массива*** будет соответствующим - 32, 16 и 8 элементов. Следует запомнить, что в начале полученного массива всегда расположен младший бит двоичного представления. Один из самых простых примеров применения данной функции (по крайней мере, то, что сразу приходит на ум) - проверка чисел на четность. Очевидно, что достаточно "извлечь" из полученного массива этот младший бит (что осуществляется с помощью функции ***Индексировать массив (Index array)*** подпалитры ***Массив (Array)***) - он и будет "показателем"! Если же проверять на четность непосредственно данные в числовом виде, то будет необходимо либо производить их деление на 2 с последующей оценкой полученного результата, либо использовать какие-нибудь экзотические функции, что, конечно, не является упрощением ситуации.

Функция ***Логический массив в число (Boolean array to number)*** фактически выполняет преобразование, обратное описанному:

Тип числа, получаемого на ее выходе - беззнаковое 32-битное. Первый бит ***Логического массива*** (т.е. нулевой его элемент) "расценивается" как младший в двоичном представлении числа.

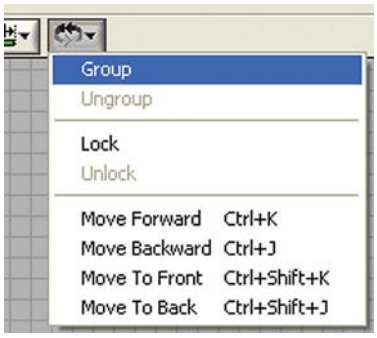
Функция ***Логическое значение в число (Boolean to (0,1)***) выполняет преобразование логических данных побитно в арифметические:

Как ни странно, тип данных на ее выходе (0,1) - 16-битное целое (т.е. с учетом знака), хотя достаточно было бы и 8-битного беззнакового числа (учитывая диапазон принимаемых значений). Следует отметить, что подача массива логических данных на вход функции также возможна, при этом на выходе будем получать соответствующий массив численных значений 0,1.

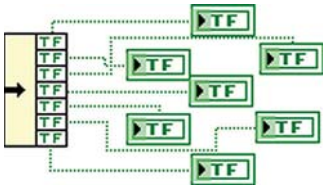
Для закрепления всего вышеизложенного, а также для упрощения создания последующих программ в LabVIEW рассмотрим следующий пример. При создании виртуальных приборов на основе модулей сбора данных и моделирования работы существующих реальных приборов с целью большей реалистичности часто применяются семисегментные индикаторы.

Поскольку среди инструментария LabVIEW "готовые" семисегментные индикаторы отсутствуют, одним из возможных решений является простой "набор" индикатора из восьми обычных графических индикаторов. При этом каждому из них необходимо придать соответствующие размеры и форму, что достигается "растяжкой" его краев с помощью манипулятора. Сегмент-запятую очень просто создать, например, с помощью круглого индикатора.

Было бы неплохо после формирования индикатора провести группировку составляющих его элементов. Для этого достаточно "обвести" их, нажав правую кнопку мыши, после чего выбрать пункт ***Сгруппировать (Group)*** в стандартом меню ***Reorder***. Это позволит впоследствии перемещать созданный индикатор с намного большим удобством и исключить повторную группировку его элементов при перемещении, например, одного или части из них.



После "набора" на передней панели, для последующего удобства настоятельно рекомендуем скомпоновать их аналогичным образом на блок-диаграмме. Это значительно облегчит Вашу будущую работу по подключению соединительных проводников к нужным индикаторам.

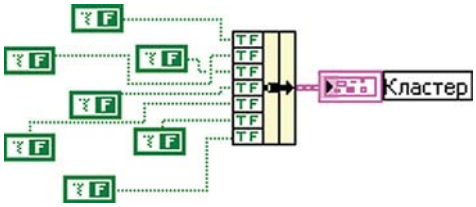


В данном случае сегмент-запятая на рисунке не представлен. Обычно пиктограммы индикаторов-запятых на блок-диаграмме размещаются отдельно от "стандартных" сегментов. Это объясняется тем, что управление запятыми зачастую осуществляется по специальным алгоритмам, и весь этот фрагмент программы размещают на блок-диаграмме отдельно.

Интуитивно понятно, что работать с такой структурированной группой пиктограмм на блок-диаграмме значительно удобнее, нежели с хаотическими их нагромождением. Цвет символа, выводимого на индикаторе, можно задать следующим образом. Необходимо для каждого из простых индикаторов установить цвет их включенного состояния, т.е. их цвет при подаче на них значения TRUE. Это выполняется путем выбора соответствующих цветов из выпадающей палитры:

Данное подменю вызывается указанием опции ***Properties*** после щелчка правой кнопкой мыши на индикаторе (на лицевой панели).

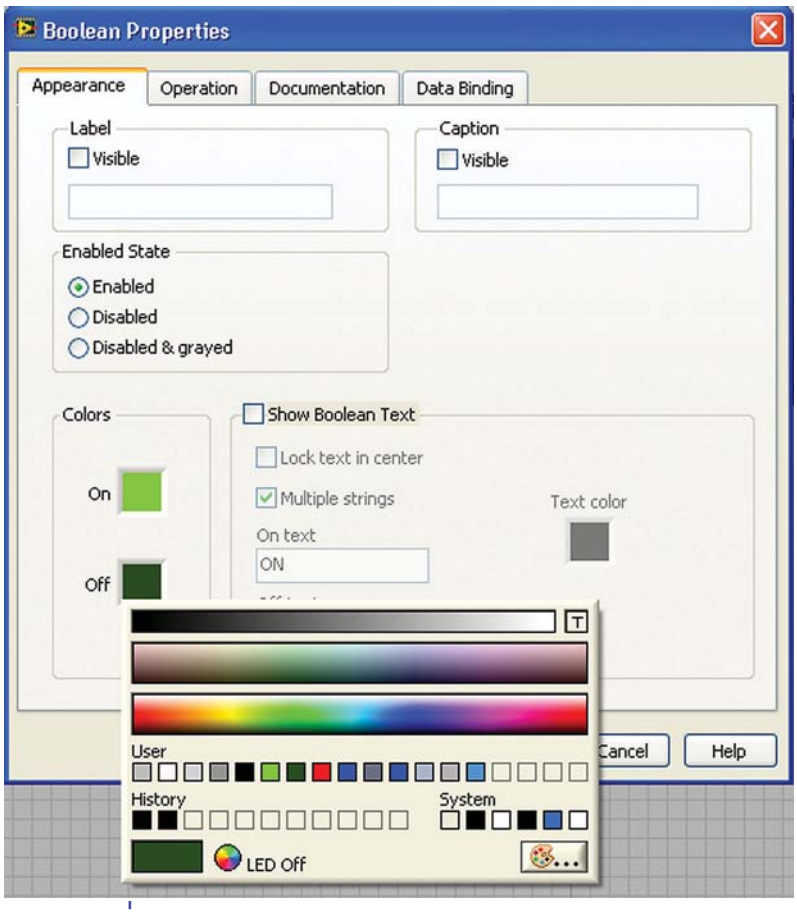
Далее, после "сборки" самого индикатора, необходимо реализовать для него соответствующий дешифратор. Он служит для перевода значения кода, задаваемого пользователем (т.е. символа, который необходимо вывести на индикатор) в семисегментный код. Рассмотрим пример, когда пользователь задает какую-либо цифру в форме беззнаковой однобайтной переменной. Естественно, для подключения данных в блок-диаграмме к индикаторам-сегментам необходима совокупность восьми соединительных проводников (фактически, переменных булевского типа). Наиболее удобно "собирать" данные 8 проводников в кластер с помощью блока ***Bundle:***



Как Вы, наверное, успели заметить, константы булевского типа также полезно компоновать в соответствии с внешним видом индикатора.

"Распаковка" же кластера будет осуществлена с помощью блока ***Unbundle*** уже при "выходах" из блок-диаграммы на лицевую панель прибора.

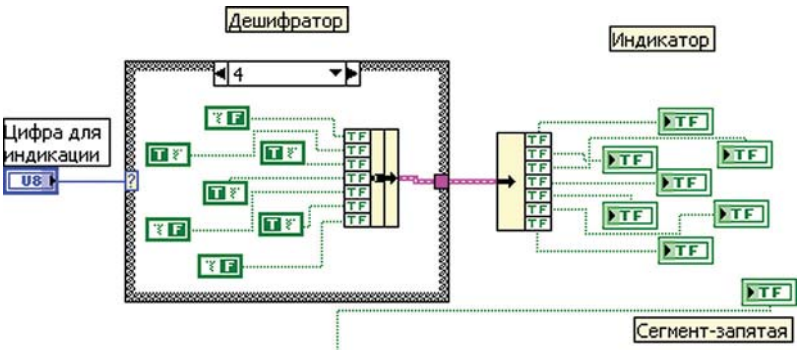
Использование множественных переменных типа "массив" также не возбраняется. Просто в этом случае "распаковка" будет несколько менее удобна. Можно, естественно, и отказаться от кластеров с массивами, но в таком случае блок-диаграмма Вашего виртуального прибора грозит стать непроглядной "паутиной" из соединительных проводников. Кроме того, соединение их по отдельности займет в семь раз больше времени.



Непосредственно дешифратор можно создать, используя структуру ***Вариант (Case structure)***. Ее управляющий вход необходимо связать с целочисленной переменной, значение которой надлежит вывести на индикатор. При этом варианты структуры, представленные в верхней ее части, изменятся с логических на арифметические. Правда, сначала их количество будет ограничено всего двумя (0,1). Поэтому для увеличения ихчисла, т.е. формирования диапазона 0 - 7, необходимо выбрать пункт ***Add case after*** в меню, выпадающем после нажатия на верхнюю часть ***Case structure***. Таким образом следует добавить варианты необходимое количество раз. Выбор именно ***Add case after*** (в отличие от ***Add case before***) необходим для увеличения номеров вариантов относительно 0,1.

Далее внутри каждого варианта следует сформировать кластер с соответствующими булевскими значениями для индикации. Это делается путем помещения булевских констант и блока ***Bundle***, изображенных на последнем рисунке, внутрь каждого из вариантов. А затем значения этих констант изменяются (либо остаются неизменными) в соответствии с отображаемой цифрой.

В конечном итоге, после соединения всех узлов блок-диаграммы проводниками, индикатор с



дешифратором будут выглядеть приблизительно следующим образом:

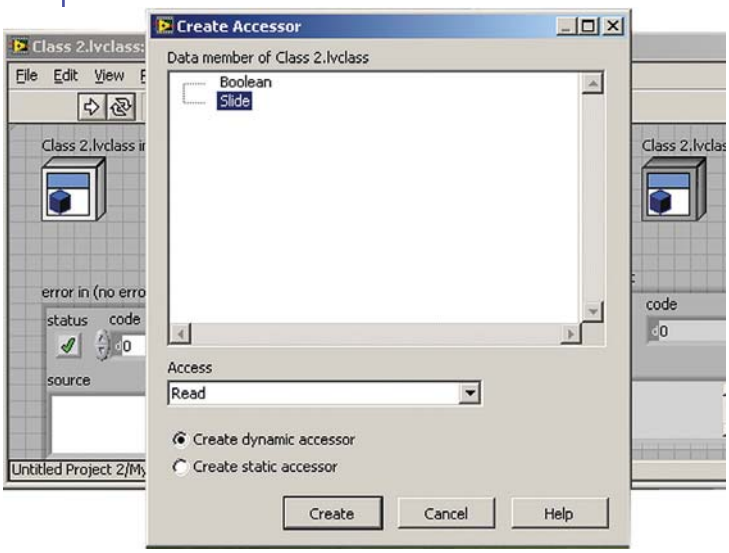
Весьма полезным будет создание ***subVI***, включающего созданный дешифратор. Его входом может быть значение целочисленной переменной - цифры для индикации, а выходом - кластер булевских переменных-сегментов. О том как создать ***subVI*** речь шла еще в Уроке по LabVIEW №2.

Кроме десятичных цифр на семисегментных индикаторах можно представлять и некоторые символы, формируя из них сообщения об ошибках, переполнениях и т.п. Широта выбора здесь, конечно, ограничена количеством сегментов, но все же некоторые решения вполне можно осуществить.

Ну, а вместо заключения упомянем о новых возможностях LabVIEW 8.5, о которых Вы могли и не знать. В новой версии появился ряд функций и улучшений, уже давно ожидаемых многими пользователями. Среди наиболее ожидаемых - новые утилиты, встроенные в ***Project Explorer***, которые дают возможность управлять большим количеством VI и предотвращают появления таких проблем, как перекрестные ссылки.

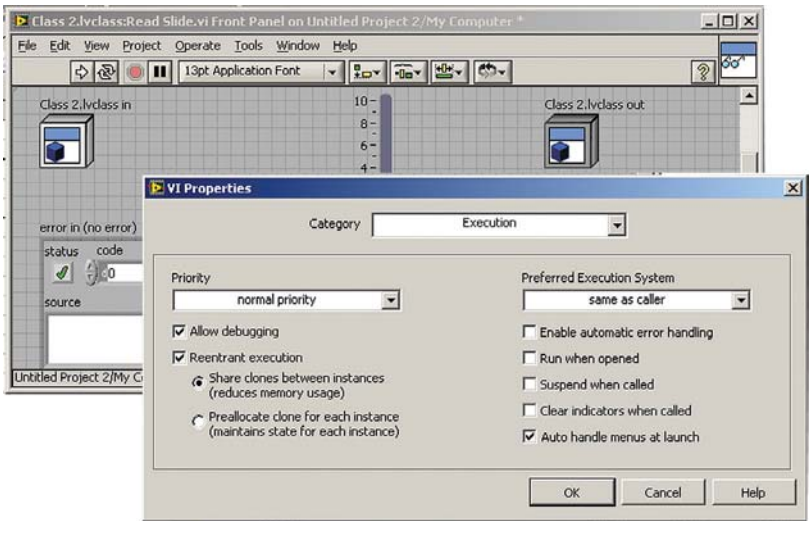
Разработчики включили также в новый дистрибутив несколько запрошенных пользователями функций, расширяющих возможности в области объектно-ориентированного программирования и взаимодействия с библиотеками DLL.

Автоматическое создание методов get/set для данных в классах. Данные, находящиеся внутри класса,



представляют собой приватную информацию. Это значит, что не существует способа изменить состояние объекта, кроме как взаимодействуя с ним через определенный интерфейс. Такой подход является широкораспространенной практикой, позволяющей более-менее точно контролировать поведение своих объектов посредством создания так называемых аксессоров. Аксессор представляет собой комбинацию методов ***get*** и ***set***, предназначение которых, как понятно из названия, получить требуемое свойство и записать свойство. Создание методов доступа сейчас автоматизировано, что позволяет значительно облегчить труд разработчиков, создающих классы с большим количеством свойств.

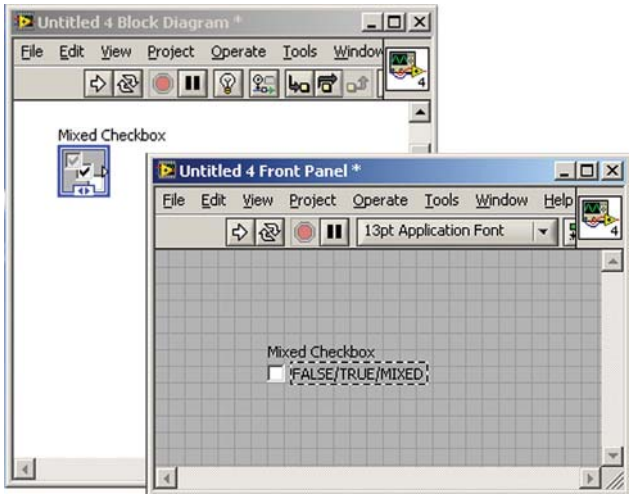
Для того, чтобы создать новый метод доступа, следует нажать правой кнопкой на имени класса в ***Project Explorer*** и выбрать ***Create New*** в ***VI for Member Data***



Access. Появившийся диалог будет содержать в себе все свойства класса. Нажав на одном из них, Вы получите возможность не только создать метод, но и указать права на чтение/запись. Для комплексных типов данных, какими являются массивы, Вы можете адресовать как целый массив, так и его отдельные элементы. В последнем случае нужно будет указывать индекс элемента в массиве.

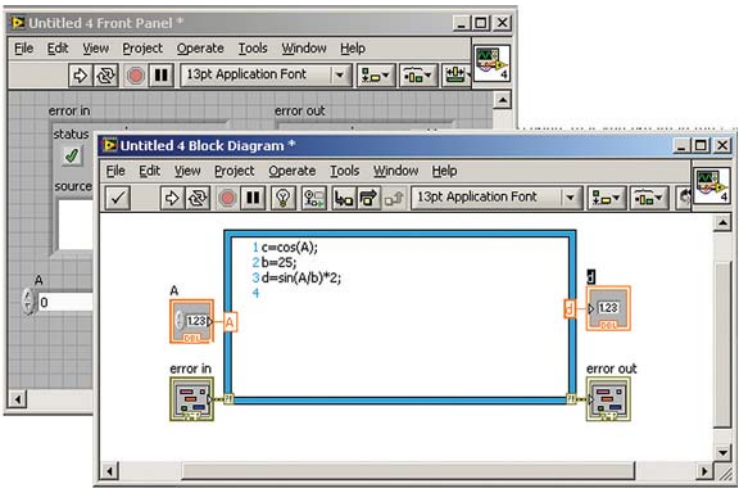
Рекурсия. Используя LabVIEW 8.5, Вы можете воспользоваться рекурсивными алгоритмами. Рекурсивные VI могут вызывать сами себя из собственной блок-диаграммы или из диаграммы любого другого VI. Это может оказаться полезным, если Вы планируете использовать выходные данные много раз в одном и том же процессе. Для того, чтобы сделать рекурсию доступной, выберите опцию ***Share clones between instances*** во вкладке ***Execution Properties*** диалогового окна ***VI Properties***. Правильное использование рекурсии может увеличить эффективность и помочь с оптимизацией кода, однако следует помнить, что ее использование также приводит к тому, что со временем алгоритм может значительно усложниться. Подходы, используемые в стандартных текстовых языках, справедливы и для LabVIEW. Рекурсивный алгоритм требует наличия так называемой базовой точки, при достижении которой будет произведен выход из рекурсивного вызова. Кроме того, рекурсивный VI не может быть интерфейсом верхнего уровня, а должен вызываться из другой программы.

Элемент Checkbox поддерживает три состояния. В редакторе лицевых панелей появился новый элемент управления, который называется ***Mixed Checkbox***. В отличие от традиционного ***Checkbox-а***, этот элемент имеет три состояния. Это бывает полезным для получения ввода, где возможны неопределенные или неизвестные состояния. Например, если Вы используете стандартный



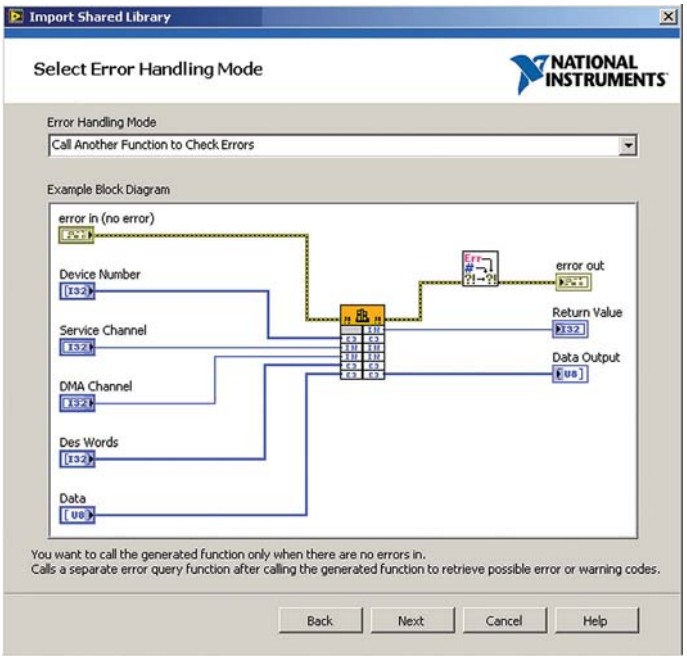
Checkbox для того, чтобы устанавливать опции для множества объектов, все конфликты между состояниями могут быть решены, если воспользоваться ***Mixed Checkbox*** и его третьим состоянием.

Усовершенствованный отладчик MathScript. В LabVIEW 8.5 появились функции вывода более детальной информации об ошибках, возникающих в ***MathScript Node***. Например, если Вы вызываете собственную функцию или же ***.m***-файл, содержащий ошибки, LabVIEW сможет их выявить еще во время редактирования. Согласитесь, значительно лучше самому разбираться с ошибкой



компиляции, чем выслушивать от пользователей жалобы об ошибках времени выполнения. Если Вы внесете изменения в ошибочный код, LabVIEW автоматически обновит список ошибок.

Упрощенная работа с DLL и указателями. Начиная с LabVIEW 8.2, National Instruments предлагает воспользоваться возможностями ***Import Shared Library Wizard***, который представляет собой мощный инструмент, позволяющий экономить время разработки программы и сократить количество ошибок. Данный помощник дает возможность легко подключать DLL библиотеки любой сложности, быстро и просто настраивать входные параметры, которые будут использоваться в графическом коде. Кроме того, помощник поддерживает работу с указателями. Указатели обычно используются в текстовых языках для адресации значений в памяти, что, в общем-то, не является необходимым для LabVIEW ввиду графической природы самого языка.



Материал подготовлен студентами старших курсов факультета авиакосмических систем Национального технического Университета Украины "Киевский политехнический институт".