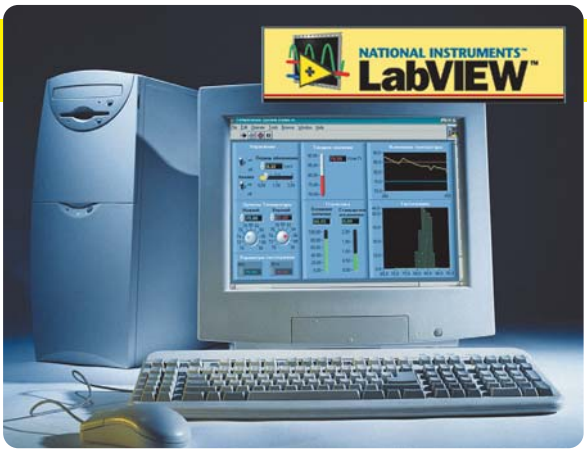
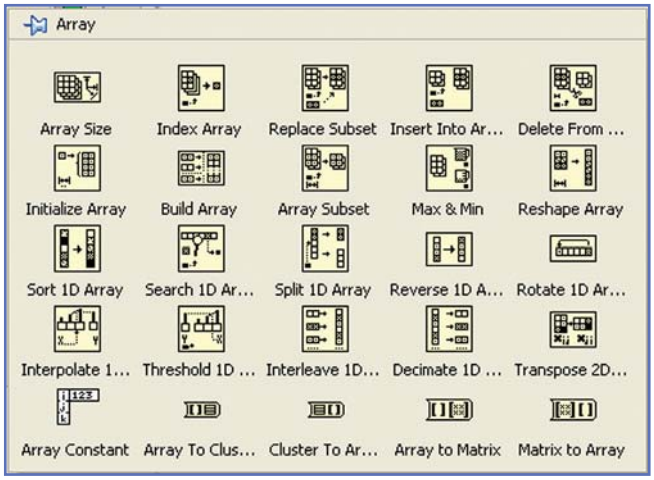
Учебный практикум по LabVIEW

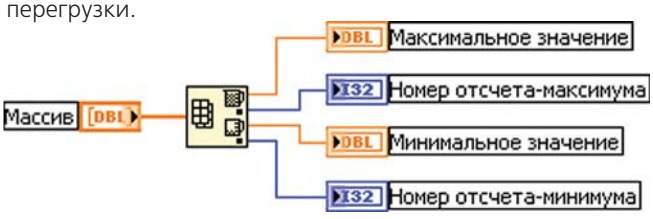


Пожалуй, трудно себе представить исходный текст "серьезной" программы, в которой не использовались бы функции работы с массивами и преобразования форматов данных. Без них – ну, никак не обойтись! Частично эти функции рассматривались в предыдущих выпусках учебного практикума. А теперь давайте рассмотрим их чуть более подробно, и с конкретными небесполезными примерами.

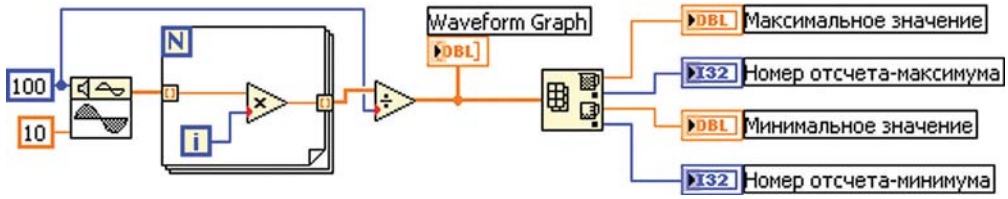
Практически все, что касается работы в LabVIEW с массивами, располагается в подпалитре ***Array***:



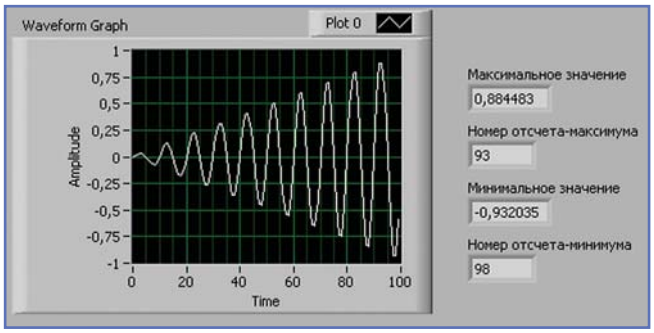
Работа функции ***Array max & min*** *(максимум и минимум массива)* очевидна из ее названия. Данная функция полезна, например, при определении, насколько полно используется динамический диапазон устройств аналогового ввода или не находится ли АЦП в режиме перегрузки.



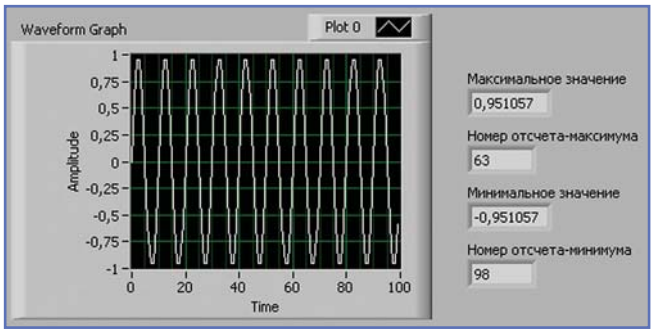
Для лучшего уяснения работы функции подадим на вход амплитудно-модулированный синусоидальный сигнал (закон модуляции в данном случае неважен, поэтому установим линейное возрастание амплитуды - это довольно просто сделать, как следует из блок-диаграммы). Для генерации синусоидального сигнала используем рассмотренную ранее функцию ***Sine Pattern***:



Оценив результаты работы функции, делаем вывод: ее минимум и максимум находятся в конечной части временного интервала. Об этом свидетельствуют номера отсчетов 98 и 93 соответственно, а всего отсчетов 100.



Следует отметить, что если в выборке присутствует несколько отсчетов, содержащих, например, минимум, то корректное определение номера отсчета является затруднительным. Для иллюстрации такой ситуации подадим на вход функции "чистую" синусоиду:

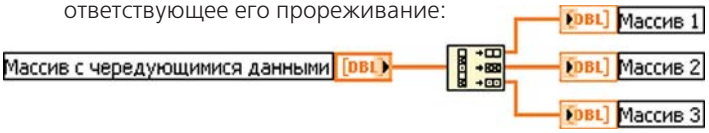


Еще один пример использования функции ***Array max & min:*** определение состояния перегрузки АЦП устройства сбора данных. Для этого достаточно сравнить минимум и максимум массива с соответствующими предельными значениями входного диапазона АЦП. Если же массив данных является целочисленным, т.е. Вы работаете с десятичным кодом, то пределы АЦП необходимо также выразить в виде десятичного числа:

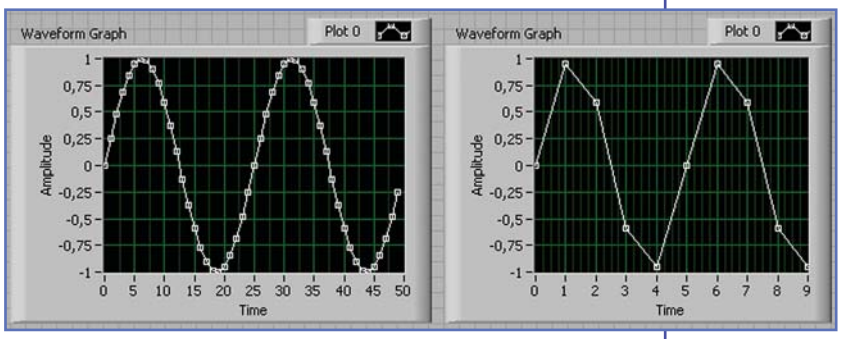
для 10-разрядного АЦП пределы будут равны -512 и 511.

В некоторых моделях систем сбора данных предполагается программная коррекция результатов аналого-цифрового преобразования. В этом случае предельные значения кода будут отличаться от номинальных $-(2^n)$ и 2^n-1 (n - разрядность АЦП) на значение соответствующих поправочных коэффициентов. Это отличие относительно незначительно: вместо упомянутых значений кода -512 и 511 вполне возможны например, -498 и 526. Но именно с этими данными и следует сравнивать полученные с выхода функции ***Array max & min*** значения.

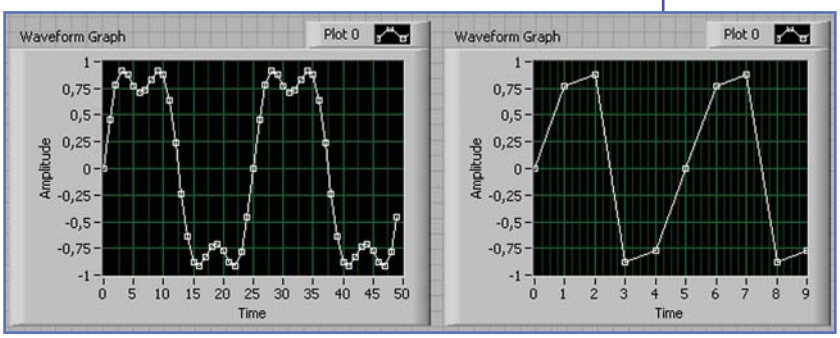
Функция ***Decimate 1D array*** *(децимировать одномерный массив)* осуществляет соответствующее его прореживание:



Данная функция полезна во многих случаях. Давайте рассмотрим два самых важных из них. Во-первых, она позволяет избавиться от избыточности информации о сигнале. Например, если по каким-то причинам Вам не удается установить требуемую частоту дискретизации АЦП (используется слишком быстродействующее оборудование для оцифровки и быстро и медленно изменяющихся сигналов). Проиллюстрируем это графически. Пятикратная децимация синусоиды, частота которой достаточно низкая для данных условий, не избавит Вас от информации, какого вида сигнал присутствовал на самом деле, перепады его достаточно четко отслеживаются:

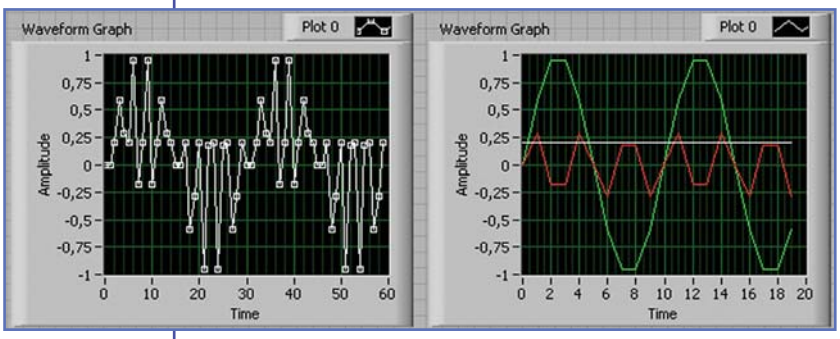


Однако использовать данную функцию необходимо осторожно, учитывая теорему Найквиста и возникающее при дискретизации "отсечение" части спектра сигнала. Таким образом, если в нашем сигнале вместе с "основной" синусоидой присутствует и "важная" высокочастотная составляющая (например, частота которой втрое больше), информация о ней не сохранится. И никакой алгоритм восстановления сигнала уже не даст Вам то, что потеряно раз и навсегда:



Во вторых, децимация массива очень полезна при работе с несколькими каналами, когда коммутатор модуля сбора данных последовательно подключает входы к собственно АЦП. В этом случае массив кодов, получаемый на выходе АЦП, это "смесь" отсчетов последовательно переключаемых каналов. Зная сколько каналов опрашивается, можно расщепить полученный массив, используя функцию ***Decimate 1D array***.

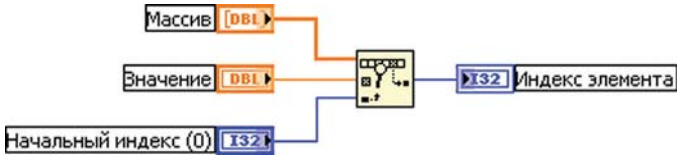
На представленных ниже экранах, в левой части - графическая интерпретация массива отсчетов, полученных с АЦП в многоканальном режиме. Нельзя сказать, что информация на этой диаграмме представлена в понятной форме. Зато после разделения массива на три (изображение справа), Вы уже можете с уверенностью сказать, сигнал какого типа присутствует на каждом из входов, а далее - обрабатывать полученные массивы, используя стандартные функции LabVIEW.



Замечание! Если размер массива с чередующимися данными не кратен количеству массивов после децимации, то несколько отсчетов в конце массива отбрасываются для обеспечения кратности. Таким образом, размеры возвращаемых массивов всегда одинаковы.

Используя функцию ***Search 1D array*** *(искать в одномерном массиве)*, Вы можете найти в массиве положение элемента, содержащего заданное значение, а точнее его индекс.

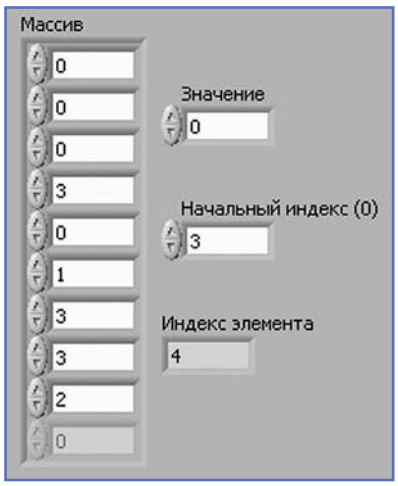
Эта функция весьма удобна в случае, когда, например, необходимо рассчитать время нарастания сигнала до какого-либо значения. Предположим, Вы получаете данные с выхода интегратора, который при достижении некоторого



напряжения автоматически разряжается. "Вытянув" номера элементов, соответствующих переходу напряжения через 0 и достижению заданного напряжения, легко можно рассчитать временной интервал заряда. В качестве простейшего примера приведена работа с массивом целых чисел.

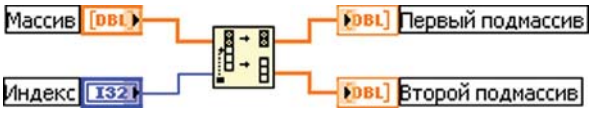
Следует заметить, что эта функция не обрабатывает ту часть исходного массива, которая находится до элемента с заданным начальным индексом, и не забывайте, что индексация в массивах

LabVIEW начинается с нуля. Т.е., для рассматриваемого примера, задав начальный индекс равным 3 (начинает поиск с четвертого по счету элемента) и желая найти в массиве нулевое значение, на выходе функции получим индекс 4. Функция в этом случае пропускает три нулевых элемента в самом начале массива. А еще нужно отметить, что в отличие от функции ***Array max & min*** , если после элемента с начальным индексом присутствуют два или более искомых элемента, то Вы всегда получите индекс первого из них. Если элемент с заданным значением не был найден, то возвращаемое значение индекса -1.



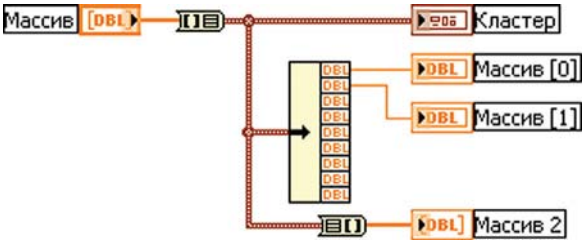
Функция ***Split 1D array*** *(разделить одномерный массив)* позволяет получить отдельно две части массива, разделенного в каком-либо заданном месте. На последнее указывает значение входа ***Индекс*** - номер элемента, с которого должен начаться второй подмассив. Одновременно ***Индекс*** - это и размер первого подмассива.

Очень часто эта функция используется при обработке информации, получаемой от устройства сбора данных. С ее помощью можно "отсечь" ту часть последовательности, в которой присутствует переходной процесс либо тренд, если эти явления для нас нежелательны.



Назначение функций ***Sort 1D array*** *(сортировать одномерный массив)* и ***Reverse 1D array*** *(обратить элементы одномерного массива)* понятно из их названий, и пока ни у кого особых трудностей с их использованием не возникало:

Функция ***Array to cluster*** *(массив в кластер)* преобразует массив, подаваемый на ее вход, в кластер, содержащий ту же самую информацию. Порядок и тип элементов кластера полностью соответствует порядку и типу элементов в массиве. Таким образом, получаемые данные представляют собой частный случай кластера. Казалось бы, главное преимущество кластеров - это как раз содержание разнотипных данных, и зачем тогда нужна функция ***Array to cluster***, ведь можно обходиться лишь массивами? Однако, следует отметить, что "распаковка" последовательности данных в этом случае намного удобнее. Ведь в этом случае достаточно использовать один раз ***Unbundle*** вместо многократного использования других функций для получения каждого отдельного элемента массива:

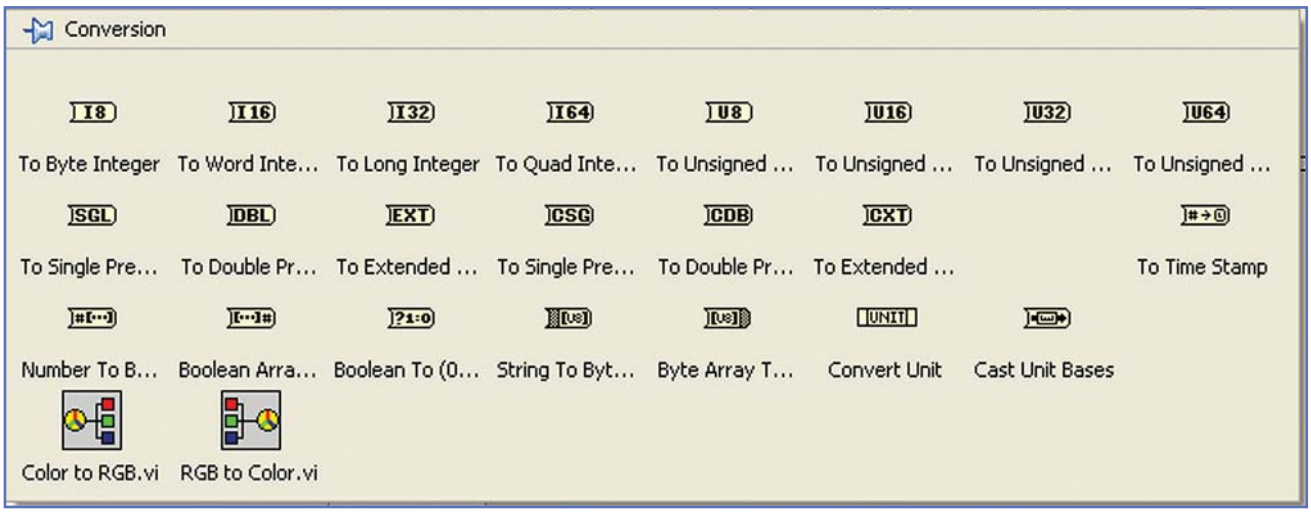


А если требуется полностью "распаковать" массив, состоящий из нескольких десятков элементов, то соответствующий фрагмент кода при использовании рассмотренной функции будет намного меньше.

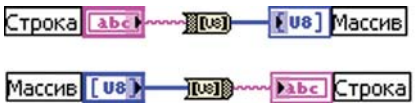
Для выполнения обратного преобразования, т.е. кластер в массив, следует применить функцию ***Cluster to array***. Но при этом следует помнить о том, что все элементы кластера должны быть одного типа, который впоследствии и определит тип возвращаемого массива.

А теперь рассмотрим особенности применения функций подпалитры ***Conversion*** в составе палитры ***Numeric***, которые предназначены для преобразования данных в другие, отличные от предыдущего, типы. Как видно из изображения подпалитры, подавляющее большинство функций имеют по одному входу и выходу.

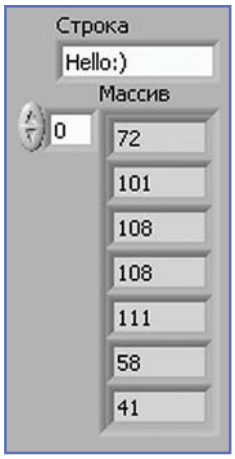
Функция ***String to byte array*** *(преобразование строки в массив байтов)* может быть востребована, если необходимо передать текстовое сообщение, например, между модулями В/В, находящимися в составе разных систем. Если модуль поддерживает работу с цифровыми данны-



ми, то, как правило, в его составе уже есть как минимум 8 линий для дискретных сигналов. А ***String to byte array*** как раз и делит строку на "порции" по 8 бит. Кроме того, эта функция может быть использована и в случае, если потребуется записать текстовую информацию (например, идентификатор устройства или его тип) в определенное место EEPROM устройства. Хотите Вы того или нет, а запись нужно выполнять именно побайтно.



Обратное преобразование, т.е. преобразование массива байт в строку, тоже достаточно часто бывает востребована. Реализуется это с помощью функции ***Byte array to string***. Вот в какое забавное сообщение будет преобразован массив, на первый взгляд, случайных чисел. А ведь это сообщение просто представлено в виде последовательности ASCII-кодов текстовых симоволов.



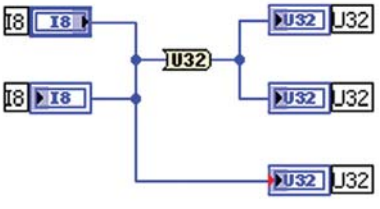
Функции, предназначенные для изменения формата чисел в LabVIEW, используются в случаях, когда необходима совместимость частей программного кода, либо для экономии ресурсов.

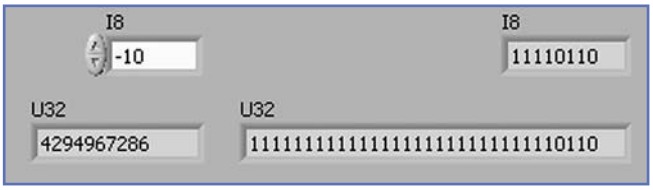
В предыдущих практикумах рассматривались особенности программирования в LabVIEW контроллеров NXT из популярных конструкторов LEGO, а также возможность создания собственных функциональных блоков для среды программирования Lego Mindstorms NXT. Так вот, представлять данные в этом случае необходимо в виде беззнакового однобайтного числа U8. Иначе возможно появление ошибок в работе контроллера NXT. С аналогичными особенностями представления данных приходится сталкиваться, работая в LabVIEW, и с другими интерфейсами. Учитывая требования побайтной передачи, Вы не сможете принять или отправить по шине, например, число подсчитанных счетчиком импульсов, которое может быть достаточно большим.

Что касается экономии памяти и вычислительных ресурсов, то достаточно вообразить, что в программе задействован массив объемом, предположим, 100 тысяч элементов. В процессе работы он постепенно заполняется данными, получаемыми с 10-разрядного АЦП, причем используется только половина рабочего диапазона. В случае, если вполне допустимо "отбросить" еще один бит в результате преобразования, то отсчет будет занимать уже на 2 байта, а один. Экономия существенная.

Итак, рассмотрим особенности преобразования форматов на примере некоторых функций из подпалитры ***Conversion***.

Как работает функция ***To unsigned long integer*** *(перевод в беззнаковое длинное целое)*? В приведенном примере для удобства начальный и конечный результаты выражены в двоичном и десятичном виде.

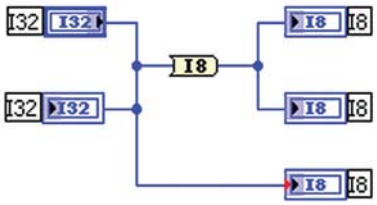


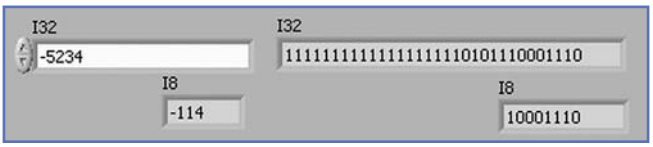


Вспоминайте особенности дополнительного кода, в котором представляются числа в памяти компьютера. Если на входе функции положительное число либо данные в беззнаковом представлении ***U8*** или ***U16***, то проблем не возникнет. Функция всего лишь "допишет" некоторое количество нулей в разрядах старше существующих таким образом, чтобы общее число разрядов было 32. А вот если на вход подать U64, старшие разряды будут просто отсечены. Если число, записанное в формате ***U64*** "не вмещается" в ***U64***, на выходе Вы получите несколько меньшее число. В противном случае результат на выходе будет идентичен входному.

А вот если же на вход поступит отрицательное число, то слева будут "дописаны" единицы, что в десятичном представлении будет выглядеть весьма загадочно. Именно этот случай и проиллюстрирован. Впрочем, в случае подачи отрицательного дробного числа (например, типа ***DBL***) функция не будет работать вообще.

Теперь об очень интересной функции ***To byte integer*** *(перевод в байтовое целое)*:





Из примера следует, что произошло отсечение старших разрядов таким образом, чтобы общее их количество стало равным 8-ми. После этого, чтобы получить десятичное представление, старший из оставшихся бит был определен как знаковый. А в первичных данных знаковым битом был ведь совершенно другой. Таким образом, в десятичном представлении связь между входом и результатом может быть полностью скрытой. Из отрицательного числа может образоваться положительное число и наоборот, а может и не произойти смены полярности. Если же подавать на вход, например, ***U32***, то принцип работы и результат будут аналогичными.

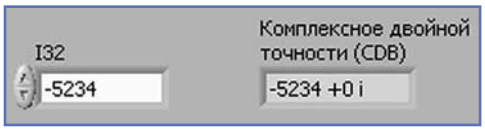
Следует заметить, что искажение десятичного представления вполне может появиться при соединении проводниками на блок диаграмме фрагментов, настроенных на работу с данными разного типа. В двух приведенных примерах, представляющих фрагменты блокдиаграмм с использованием ***To unsigned long integer*** и ***To byte integer***, соединены переменные и индикаторы разных типов, что автоматически отмечено красным цветом. Таким образом LabVIEW предупреждает Вас, что программа работать будет, но возможны неполадки и "побочные нежелательные эффекты".

Вообще говоря, результат на выходе функции пре-

образования формата не будет отличаться (в десятичном представлении) от входного только в том случае, если преобразуемое число попадает в оба диапазона - входной и выходной. Поэтому можно порекомендовать, во избежание возникновения перечисленных трудностей, следить за диапазоном значений, который фактически принимает переменная.

Выше описано лишь преобразование одного числа с многобайтным представлением в компактный вид (***I8***, ***U8***). А над тем, как разложить его в последовательность, т.е. просто разделить на однобайтные "куски", для последующей передачи, подумайте сами.

Не менее интересны функции ковертации в комплексные числа. Если "пощупать" функцию ***To double precision complex*** *(в комплексное двойной точности)*, то можно заметить, что данные, поступающие на ее вход, записываются лишь в действительную часть:



В таком случае удобнее формировать комплексное число заданного формата используя функции, подобные рассмотренной ранее ***Re/im to complex*** из подпалитры ***Complex***.

Функции для конвертации логических переменных ***Number to boolean array*** *(число в логический массив)*, ***Boolean to (0,1)*** *(логическое значение в число)*, ***Boolean array to number*** *(логический массив в число)* достаточно подробно в одном из предыдущих выпусков практикума. Повторяться не будем. Ну, и в заключение отметим, что на входы функций ***Conversion*** можно подавать не только лишь одиночные переменные, а и массивы с кластерами. При этом структура выхода функции будет соответствовать входной. Таким образом, подав на вход ***To byte integer*** кластер, состоящий из чисел различных типов (дробных, целых размером в разное количество байт, знаковых либо беззнаковых), на выходе получим кластер аналогичной структуры, но все числовые данные будут сведены к формату ***I8***.

Материал подготовлен студентами старших курсов Национального технического Университета Украины "Киевский политехнический институт" факультета авиакосмических систем, при технической поддержке ООО "ХОЛИТ Дэйта Системс" (Киев).
e-mail: info@labview.com.ua