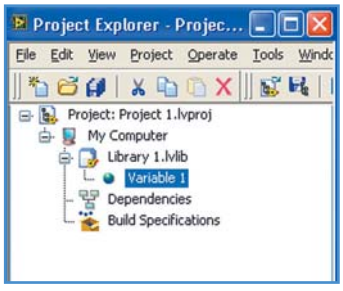
Уроки по LabVIEW

Довольно часто любому разработчику приходится иметь дело с передачей данных или параметров в самой программе или же другому приложению. В любом языке программирования, и конечно же в LabVIEW, есть различные способы реализации таких задач. Но давайте не будем забывать, что LabVIEW является средой графического программирования, призванного облегчить жизнь разработчикам. С выходом новой версии пакета, а именно LabVIEW 8, разработчики NI порадовали нас еще одним "облегчением".

LabVIEW поддерживает доступ к широкому перечню технологий, обеспечивающих создание распределенных приложений, то есть межпрограммного обмена данными на одной или нескольких машинах. Переменная Shared Variable, впервые представленная в LabVIEW 8, призвана облегчить жизнь разработчикам при создании межпрограммного обмена данными. Используя Shared Variable, Вы можете "распределять" данные между отдельными циклами, входящими в одну программу, между независимыми программами или же между программами, запущенными на отдельных PC. В отличие от имеющихся методов обмена данными, таких как LabVIEW queue, UDP/TCP и Real-Time FIFO, в данной ситуации нет необходимости вставлять в программу дополнительный код или дополнительные куски диаграммы - просто сконфигурируйте Shared Variable, используя диалоговое окно ***Shared Variable Properties*** (Свойства Распределенной Переменной).

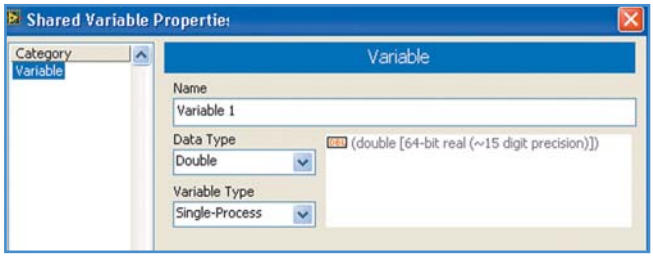
Вам доступны три типа ***Shared Variable*** - для отдельного процесса, для обмена через сеть и Shared Variable с временным триггером. Предлагается на этом уроке создать Shared Variable для отдельного процесса и для обмена через сеть. А вот работа Shared Variable с временным триггером потребует установки на Ваш компьютер дополнительного модуля LabVIEW 8 Real-Time, основные характеристики которого будут рассмотрены чуть ниже.

После такого небольшого введения давайте все-таки посмотрим, какие возможности предоставляет новая переменная. Создайте новый проект (Project), далее нажмите правой кнопкой мыши на компьютерном устройстве, таком как ***My Computer*** либо устройстве ***Real-Time*** в окне проекта, и выберите пункт меню ***New≫Variable***. Если Вы сделали все верно, откроется окно ***Shared Variable Properties***. В настройках Вы можете задать имя переменной, изменить тип данных и тип самой переменной.

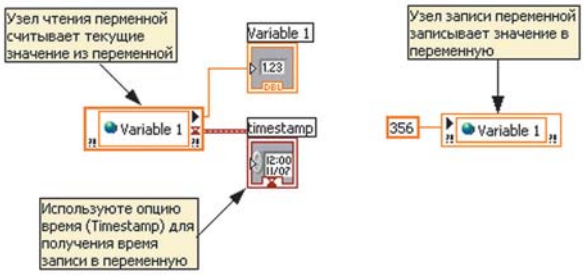


Если с именем переменной все понятно, то с типом данных есть маленькие нюансы. Вы можете выбрать из большого количества стандартных типов данных необходимый тип для новой ***Shared Variable***. Дополнительно к перечню стандартных типов существует возможность установить свой собственный, выбрав пункт меню ***Custom*** из выпадающего меню ***Data Type***. Впрочем, использование своего собственного типа данных исключает возможность работы с модулем ***Real-Time FIFO***. А кроме того, если у Вас установлен модуль ***LabVIEW DSC*** (Datalogging and Supervisory Control), т.е. SCADA-модуль, не будет полностью функционировать аварийная сигнализация Alarm.

Тип переменной определяет ее функциональные возможности. ***Single-Process Shared Variable*** (распределенная переменная для отдельного процесса) обеспечивает обмен данными в пределах компьютера. ***Network-Published Shared Variable*** (распределенная переменная для сети) позволяет обмениваться информацией по сети. Выберите ***Single-Process Shared Variable*** и установив все необходимые опции нажимайте клавишу OK. Теперь Ваша переменная добавлена в проект в библиотечной секции ***Library***. Приведенное окно ***Shared Variable Properties*** применяется к одиночному процессу. А при использовании модулей Real-Time или DSC поддерживаются дополнительные функции.



Для начала работы с переменной перетяните ее из проекта на Block Diagram. В зависимости от последующего применения установите переменную на считывание или на запись. Для этого нажмите правой клавишей мышки на переменной и выберите пункт Change to Read (изменить на чтение) или Change to Write (изменить на запись). По умолчанию Ваша переменная устанавливается на чтение.



Обратите внимание на то, что если Вы используете буфер для переменной, то ***Timestamp*** возвращает время записи этой переменной в буфер, а не время последнего обновления буфера.

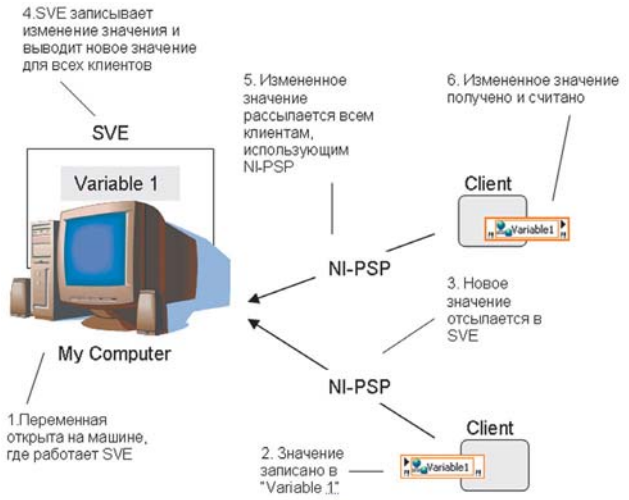
Если Вы захотите изменить свойства переменной, просто щелкните по ней правой клавишей мыши в окне проекта. Любые внесенные изменения будут применены ко всем узлам распределенной переменной, находящихся в памяти. При сохранении библиотеки все изменения будут сохранены на диске.

Использование ***Shared Variable Single-Process*** похоже на использование глобальной переменной Global variable. Главное преимущество Shared Variable для отдельного процесса (Single process) по сравнению с Глобальной Переменной - это возможность преобразовать Shared Variable для отдельного процесса в Shared Variable для передачи через сеть.

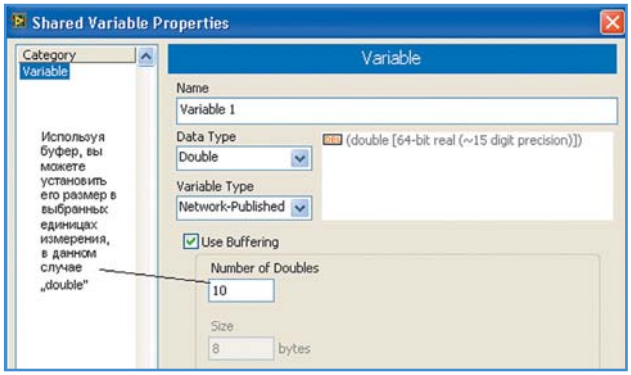
Но это еще не все. Вся мощь новой переменной раскрывается в ***Network-Published Shared Variable***. Используя Shared Variable для передачи данных по сети, Вы можете записывать и считывать данные через Ethernet. При этом сетевая реализация полностью управляется через сеть и открываются дополнительные опции, которые отсутствовали для одиночного процесса.

В сетевой версии Shared Variable используется протокол ***NI-PSP (NI Publish-Subscribe Protocol)*** для передачи данных. NI-PSP протокол является надстройкой UDP, который предназначен для отслеживания состояния сети. В отличие от UDP, NI-PSP гарантирует доставку данных, поскольку имеет дополнительную программную надстройку. NI-PSP более эффективно использует сетевое соединение, чем TCP/IP.

Передача данных по сети обеспечивается "движком" ***SVE (Shared-Variable Engine)***. SVE запускается вместе с LabVIEW и принцип его работы прост: LabVIEW посылает данные в SVE, а он в свою очередь отправляет данные адресату.

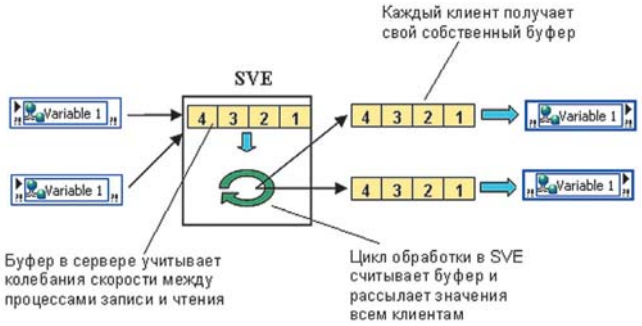


В данном случае используется клиент-серверная технология, причем SVE является сервером, а все остальные - клиентами, т. е. в каждый узел переменной встроен SVE клиент. Network-Published Shared Variable предоставляет возможность для работы с буфером. Давайте посмотрим, в каких случаях необходимо включать буфер. Применяя буферирование нет необходимости учитывать разность скорости записи и считывания Shared Variable. Дело в том, что чтение иногда осуществляется медленнее, чем запись, и часть данных может быть потеряна. Если такая ситуация не критична, т. е. Ваше приложение может терять часть данных, то буфер можно и не использовать. Но если в программе необходимо считывать все поступившие данные, то Вы обязательно должны включить буферирование. Для работы в сети можно использовать буфер Shared Variable, размер которого устанавливается в диалоговом окне ***Shared Variable Properties***:



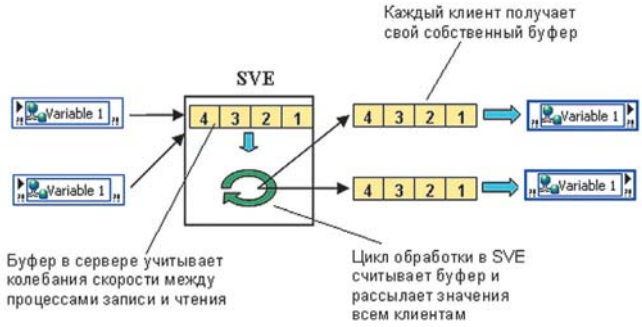
LabVIEW использует указанный размер для двух внутренних буферов: один находится в "движке" SVE, а другой - в клиенте, именно из этого буфера Вы считываете информацию.

В отличие от остальных режимов, таких как Real-Time, FIFO-enabled и Single-process variable, где используется всего один буфер для всех "отправителей" и "получателей", в Network-published для каждого "получателя" создается свой собственный буфер.



При написании приложения не забывайте, что буферирование помогает только в том случае, когда скорость чтения/записи временно флуктуирует. Если же Ваше приложение будет запущено постоянно, то "получатели", которые читают на скоростях, как правило, более низких, чем скорость записи, в конечном счете потеряют данные вне зависимости от заданного вами размера буфера.

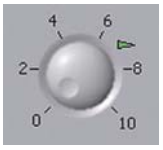
Итак, LabVIEW создает сетевые буферы, инициированные для чтения или записи, в зависимости от их местонахождения. Буферы, находящиеся на стороне сервера, создаются тогда, когда "отправитель" делает первую за-



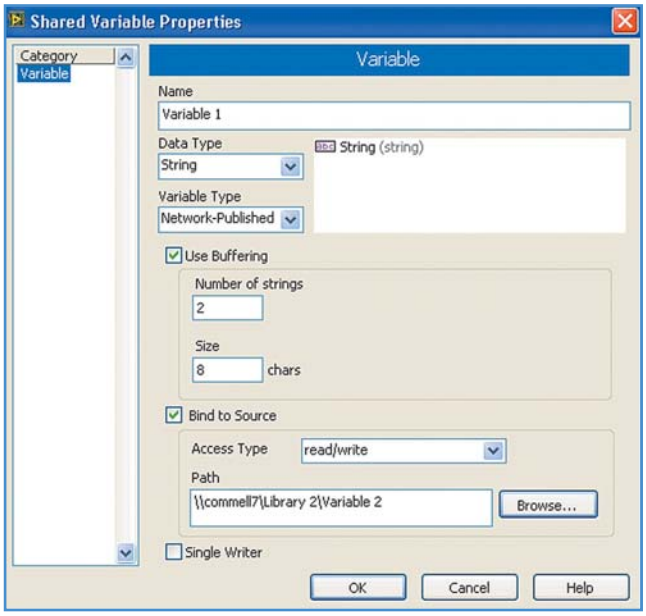
пись в Shared Variable. Буферы, находящиеся на стороне клиента, создаются тогда, когда клиент читает значение из Shared Variable в первый раз. Если "отправитель" записывает данные в Shared Variable до того, как "получатель" в первый раз прочитает Shared Variable, инициированные данные "получателю" доступны не будут. Почему? Правильный ответ - потому, что он еще не создал свой буфер.

В режиме ***Network-published*** переполнение буфера не отображается в сетевом буфере. Как было определено выше, все Shared Variables являются частью проектной библиотеки. SVE регистрирует проектные библиотеки и хранящиеся в них Shared Variables. После выключения VI или перезагрузки машины Shared Variable будет также доступна для сети. Удалить переменную из сети можно в окне ***Tools≫Shared Variable≫Variable Manager***.

Дополнительная особенность, доступная лишь для режима Network-published - создание управляющего элемента привязки. Для создания элемента управления для Вашей переменной перетяните Shared Variable из окна ***Project Explorer*** на переднюю панель программы. При запуске программы и активном SVE о нормальной работе переменной Вас будет оповещать зеленый индикатор на передней панели. Вы можете поменять привязку любого элемента управления или индикатора в закладке ***Data Binding*** в диалоговом окне ***Properties***.

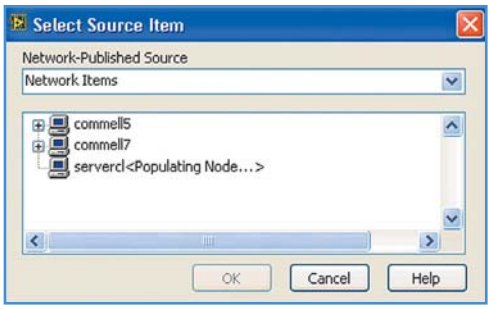


Для подтверждения важности распределенной переменой приведем простой пример программы, осуществляющей передачу сообщений по Ethernet между двумя клиентами. Откройте конфигурационное окно переменной и задайте все опции как показано ниже.

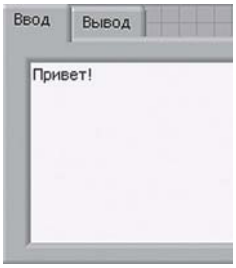


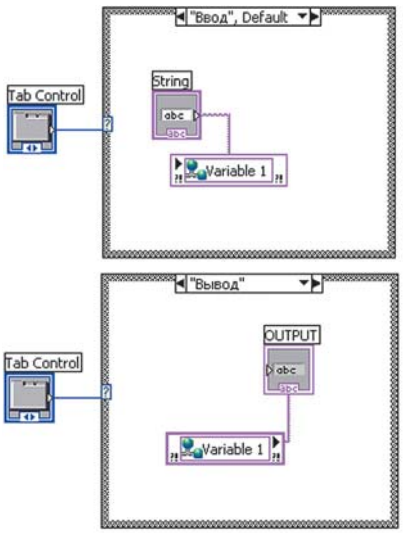
Как видите, при переходе к сетевому типу переменной пользователю предоставляется возможность включения буфера. Опцией ***Bind to Source*** (Привязка к источнику) указывается расположение распределенной переменной на втором компьютере, а также режим работы переменной на Вашей машине.

Существуют три режима: ***Only read*** (Только чтение), ***Only write*** (Только запись), ***read/write*** (чтение/запись). Определившись с работой Shared Variable, выберем переменную в сети. Нажмите кнопочку ***Browse***, в появившемся окне выберите компьютер, на котором находится переменная, библиотеку и саму переменную. Результат должен выглядеть так: ***\\commell7\Library2\Variable2***



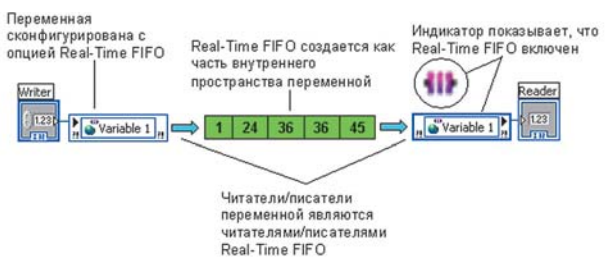
На втором компьютере проделайте те же операции, кроме ***Bind to Source***, ее включать не нужно. А теперь создайте как показано ниже Front Panel и Block Diagram, и в Вашем расположении - простой сетевой пейджер. Можете общаться:



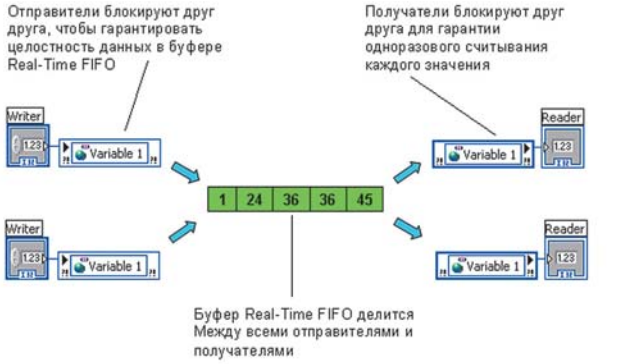


И, наконец, рассмотрим "теоретически" ***Real-Time FIFO*** буфер. Практически пощупать что это такое и "с чем его едят" можно в случае установки модуля ***LabVIEW Real-Time***, а его у Вас нет. Это коммерческий продукт. Но, пусть и заочно, все же есть смысл получить представление о возможностях тандема Shared Variable и Real-Time FIFO.

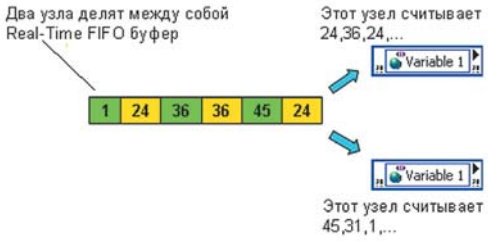
National Instruments рекомендует использовать Real-Time FIFO для передачи данных между циклом с приоритетом ***time-critical*** и циклом с самым низким приоритетом. Для избежаниия низкоуровненого программирования следует включить Real-Time FIFO для переменной. Буфер Real-Time FIFO создается при первом чтении или записи. В результате при первом считывании/записи будет происходить задержка, когда каждая Shared Variable подготавливается для дальнейшего использования. Если приложение требует очень точной временной "разметки", то нужно каждую Shared Variable прочитать один раз до выполнения основного цикла программы.



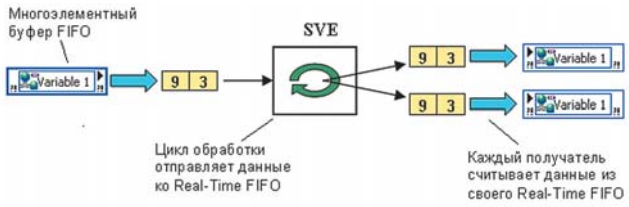
LabVIEW создает отдельный буфер Real-Time FIFO для каждого процесса, даже если используется несколько узлов чтения или записи. Для обеспечения целостности данных, при наличии нескольких "отправителей" на одну переменную, каждый "отправитель" блокирует друг друга до тех пор, пока "получатели" не прочтут данные.



При организации Real-Time FIFO следует выбрать один из двух типов буферов FIFO - буфер из одного элемента или буфер из нескольких элементов. Первое отличие между буферами - одноэлементный FIFO не выдает предупреждения о переполнении. Второе отличие - само значение, которое возвращает LabVIEW, когда несколько "получателей" читают пустой буфер. "Читатели" одноэлементного FIFO будут считывать одно и тоже значение до тех пор, пока "отправитель" не обновит данные в буфере. "Читатели" многоэлементного FIFO получают последнее значение, которое было помещено в буфер, или значение по умолчанию.



При попытке читать мультиэлементный буфер, каждый из "читателей" доходит до последней ячейки и при последующей попытке чтения возвращает самое последнее значение. Если необходимо, чтобы каждый "читатель" получал все записанные в буфер значения, следует использовать отдельные Shared Variable для каждого "читателя".



Network-Published Shared Variable так же предоставляет возможность для работы с буфером Real-Time FIFO. При одновременном использовании сетевого буферирования и Real-Time FIFO будут созданы два буфера. При этом следует заметить, что у каждого пользователя будет свой собственный буфер Real-Time FIFO, и в таком случае "отправители" и "получатели" блокировать друг друга не будут. Размеры этих двух буферов устанавливаются отдельно, однако National Instruments рекомендует устанавливать для них одинаковый размер. В отличие от режима Network-published, Real-Time FIFO каждый раз при переполнении буфера возвращает Вам ошибку.

Вроде как с Shared Variable пока все. Однако урок еще не закончен. Эта Shared Variable, конечно, несколько нарушила планы уроков, но не познакомить с ней было нельзя. Это достойное нововведение.

Поехали далее. Пора приступать к нерассмотренным ранее функциональным элементам LabVIEW. Их очень много. Поэтому для начала давайте попробуем сделать этакую классификацию всех функций LabVIEW. Разобьем все на 4 "кучки":

- базовые функции;
- функции генерации, ввода и обработки;
- функции интерфейса VI и приложений;
- функции плат и стандартных интерфейсов ввода/вывода.

С некоторыми представителями первой "кучки" Вы познакомились еще на вводном занятии и использовали многие базовые функции на всех последующих уроках. А с учетом самостоятельной работы, пожалуй, это самая освоенная Вами группа, которая включает:

- числовые функции;
- логические функции;
- строковые функции;
- функции сравнения;
- функции работы с массивами и кластерами;
- функции времени и диалогов;
- функции и VI ввода/вывода файлов.

Вторая "кучка" еще более многочисленная и, без сомнений, самая интересная. Представим ее пока тремя подгруппами:

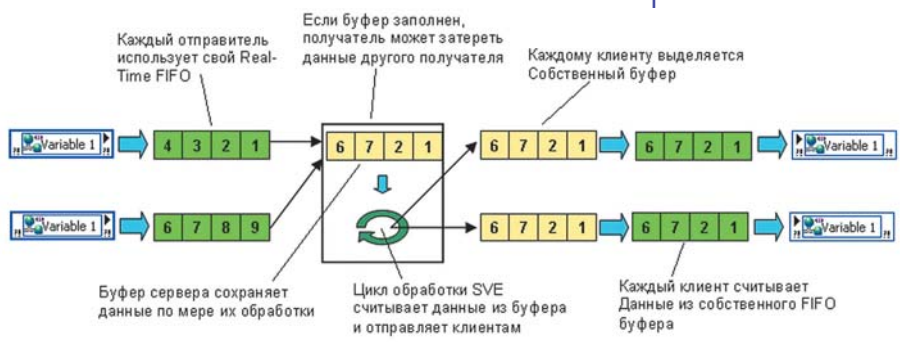
- функции генерации и обработки сигналов;
- математические функции;
- дополнительные функции.

Взаимодействие VI между собой и с другими приложениями обеспечивается широким набором функций из двух палитр ***Application Control*** и ***Communication***. Это и есть третья "кучка":

- функции управления приложением;
- функции коммуникации LabVIEW (технология и функции ActiveX и .NET, технология передачи данных и функции DataSocket, функции электронной почты, функции протоколов TCP/IP и UDP).

На работу с "железом" (платы и внешние модули аналогового и дискретного В/В и стандартных интерфейсов) ориентированы функции четвертой "кучки":

- функции формирования и обработки осциллограмм;
- функции сбора данных DAQmx;
- функции интерфейса IEE488, он же GPIB, он же КОП (канал общего пользования);
- функции последовательной коммуникации.

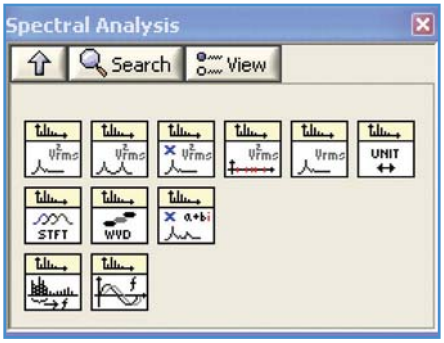


Такое упорядочение функций - это еще только первый уровень классификации. Вас можно понять - хочется уже кнопочки понажимать. Но до самих функций еще "как до неба", далеко. Давайте углубимся во вторую группу. Подгруппа функций генерации и обработки сигналов включает еще подгруппы более нижнего уровня, а именно:

- функции генерации сигналов и шумов;
- функции обработки сигналов во временной области;
- функции обработки сигналов в частотной области;
- функции фильтров;
- функции обработки весовыми окнами;
- создание осциллограмм;
- согласование осциллограмм;
- обработка осциллограмм;
- преобразования осциллограмм;

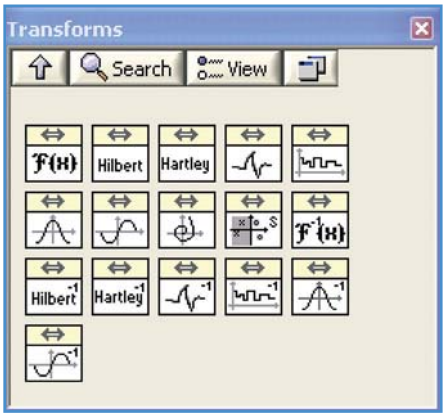
Углубляемся дальше, а что еще остается? Смотрим, что же там внутри, например, в подгруппе функций для частотного анализа сигналов. А там:

- спектр мощности (***Auto power spectrum и Power spectrum***);



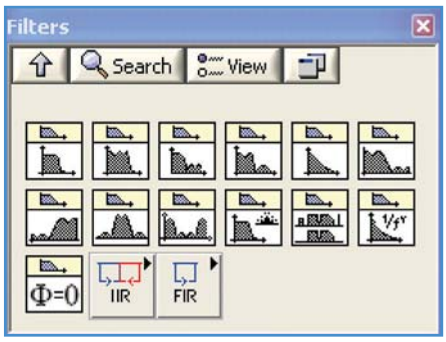
- взаимный спектр мощности (***Cross power spectrum***);
- спектр сигнала с неравномерными отсчетами (***Unevenly sampled signal spectrum***)
- амплитудный и фазовый спектр (***Amplitude and phase spectrum***);
- преобразования единиц спектра (***Spectrum unit conversion***);
- спектрограмма STFT (***STFT spectrogram***);
- спектрограмма WVD (***WVD spectrogram***);
- оценка мощности и частоты (***Power & frequency estimate***);
- оценка частоты колебания (***Buneman frequency estimate***) и много другое.

Т.е. это наборы функций для выполнения прямого и обратного преобразования Фурье и Гильберта, Хартли и Уолша-Адамара, Лапласа и вейвлет-преобразования... В этой же подгруппе на основе классического преобразования Фурье представлены также VI для оценки импульсной и частотной передаточных характеристик, функции когерентности, гармонических искажений.



У кого-то реакция на внушительный инструментарий будет типа "Ого!", а кто-то тихо скажет "Ой, куда я попал". Тогда давайте усилим ощущения обоих и посмотрим, что внутри подгруппы функций фильтрации:

- фильтр Баттерворта (***Butterworth filter***);
- фильтр Чебышева (***Chebyshev filter***);
- инверсный фильтр Чебышева (***Inverse Chebyshev filter***);
- эллиптический фильтр (***Elliptic filter***);
- фильтр Бесселя (***Bessel filter***);
- фильтр нижних частот с равномерными пульсациями (***Equi-Ripple lowpass***);

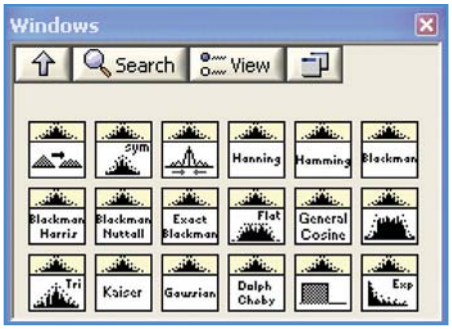


- фильтр верхних частот с равномерными пульсациями (***Equi-Ripple highpass***);
- полосовой фильтр с равномерными пульсациями (***Equi-Ripple bandpass***);
- режекторный фильтр с равномерными пульсациями (***Equi-Ripple bandstop***);
- оконный КИХ - фильтр (***FIR windowed filter***);
- медианный фильтр (***Median filter***);
- +1/f фильтр (***Inverse f filter***);
- новинка 8-й версии - ***Zero phase filter***.

Перечень этих фильтров находится в палитре ***Filters***, но там же Вы обнаружите еще две подпалитры - Advanced IIR Filtering и Advanced FIR Filtering, которые содержат дополнительные функции БИХ и КИХ - фильтрации соответственно. Это прежде всего VI, входящие в состав фильтров Баттерворта, Чебышева, Бесселя, эллиптического фильтра и др., для расчета коэффициентов этих фильтров и их конфигурирования:

- коэффициенты фильтра Баттерворта;
- коэффициенты фильтра Чебышева;
- коэффициенты инверсного фильтра Чебышева;
- коэффициенты эллиптического фильтра;
- коэффициенты сглаживающего фильтра;
- коэффициенты +1/f фильтра;
- каскадный БИХ-фильтр (***IIR cascade filter***)
- каскадный БИХ-фильтр с начальными условиями (***IIR cascade filter with I.C.***)
- БИХ-фильтр (***IIR filter***)
- БИХ-фильтр с начальными условиями (***IIR filter with I.C.***)
- коэффициенты оконного КИХ-фильтра;
- VI, реализующий алгоритм Паркса-Мак-Клеллана для расчета коэффициентов многополюсного КИХ-фильтра (***Park-McClellan***);
- узкополосный КИХ-фильтр (***FIR narrowband filter***);
- коэффициенты узкополосного КИХ-фильтра;
- свертка (Convolution) и др.

А есть еще подгруппа функций обработки весовыми окнами Хэмминга, Блэкмана, Блэкмана-Хэрриса, Хана, Кейзера-Бесселя, Гаусса, Чебышева, а также плосковершинным, треугольным, форсирующим, экспоненциальным, косинусным, косинусным 10% и др. окнами.



Так с чего же начнем? С частотного анализа или фильтров? Хотелось бы... - говорят, "хотеть не вредно". Ну, а если - "не стоит". Давайте попробуем... "попробуйте". А может быть... А вот это уже - на следующем уроке.

Материал урока подготовлен сотрудниками "ХОЛИТ Дэйта Системс", г.Киев