Primeros pasos con LabVIEW

By:
Patxi Alkorta Egiguren

Primeros pasos con LabVIEW

By:
Patxi Alkorta Egiguren

Online:
< http://cnx.org/content/col10592/1.2/ >

C O N N E X I O N S

Rice University, Houston, Texas

Collection structure revised: October 29, 2008
PDF generated: February 5, 2011
For copyright and attribution information for the modules contained in this collection, see p. 52.

Table of Contents

Chapter 1

Instalación de LabVIEW[1]

1.1 Objetivo

La finalidad de esta actividad es que el usuario sea capaz de instalar por sí mismo el software **LabVIEW** de **National Instruments** en su ordenador habitual de trabajo. Se supone que el ordenador utilizado por el usuario es de tipo PC, y que el sistema operativo instalado en él es el **Windows** (XP o superior) de **Microsoft**. En cuanto al click del ratón, salvo que se diga lo contrario, se supone éste se realiza con el botón izquierdo del mismo.

1.2 Descripción

Paso 1. La instalación de **LabVIEW** comienza insertando el CD-ROM en el lector correspondiente del ordenador habitual. Con ello, se ejecutará automáticamente el programa **autorun** de instalación. Si no fuera así, habría que visualizar el contenido del CD-ROM y hacer doble click sobre el icono (Figure 1.1) del archivo **autorun.exe**.

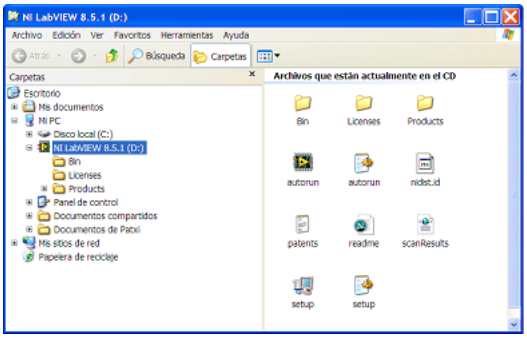

Figure 1.1: Contenido del CD-ROM de LabVIEW 8.5 Student Edition

Paso 2. A continuación, aparece la ventana inicial de instalación (Figure 1.2) , donde se hace click sobre **Install NI LabVIEW 8.5.1** para que comience la instalación de **LabVIEW**.

[1] This content is available online at <http://cnx.org/content/m18064/1.2/>.

Figure 1.2: Ventana inicial de instalación

Paso 3. Tras ello aparece la ventana de inicio de instalación (Figure 1.3), en la que hay que hacer click sobre **Next** para que continúe el proceso.

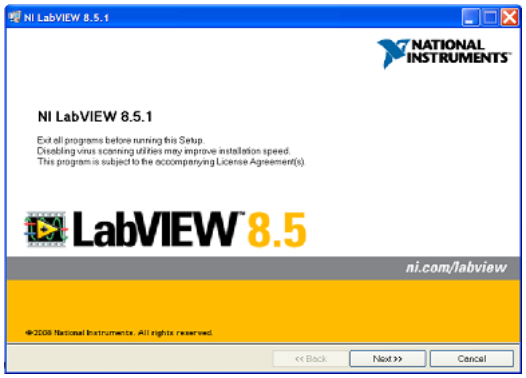

Figure 1.3

Paso 4. Acto seguido, aparece en pantalla la ventana que solicita los datos del usuario y el número de serie del software (Figure 1.4) (impreso sobre el CD-ROM). Lo lógico es que los datos de usuario coincidan con los del propio ordenador. Se elegirá la opción **Install NI LabVIEW 8.5.1 with this serial number** y se pulsará con el ratón sobre **Next**. La otra opción (no es la nuestra), permite hacer una instalación de evaluación, sin que para ello sea necesario introducir el número de serie.

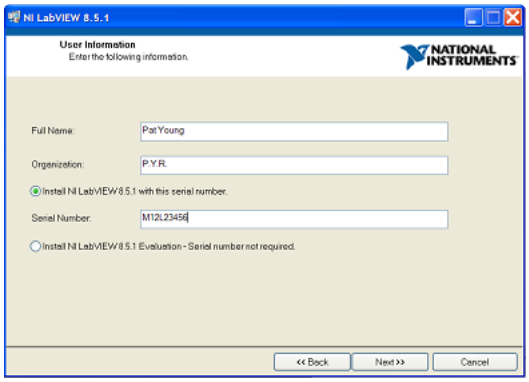

Figure 1.4: Pantalla de entrada de datos de usuario

Paso 5. La pantalla de Figure 1.5 nos muestra las ubicaciones por defecto sobre disco duro que se crearán para la instalación de **LabVIEW**. Salvo que se quieran cambiar (utilizando para ello **Browse**...), se pulsará sobre **Next**.

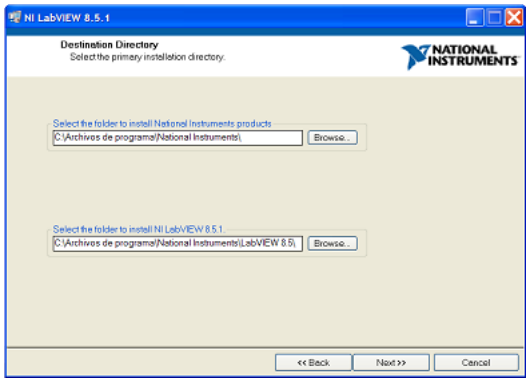

Figure 1.5

Paso 6. En la ventana que aparece en Figure 1.6, se muestra el software que compone la instalación básica oficial de **LabVIEW**, donde indica qué cantidad de disco duro necesita la misma. Pulsar sobre **Next** para continuar.

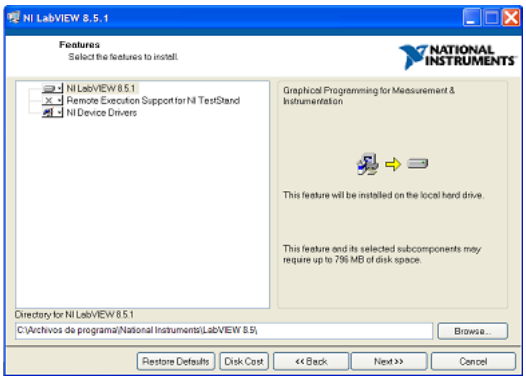

Figure 1.6

Paso 7. La ventana de Figure 1.7 produce el aviso que recomienda desbloquear el cortafuegos de red detectado, para no tener problemas en la conexión por red con **National Instruments** durante la activación de la licencia.

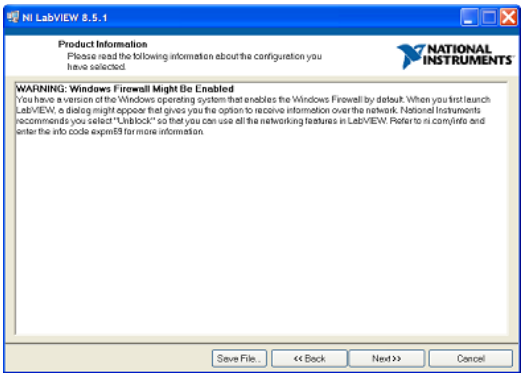

Figure 1.7

Paso 8. La ventana de Figure 1.8 nos muestra las condiciones del contrato de la licencia de **LabVIEW**. Es imprescindible estar de acuerdo con estas condiciones para poder continuar con la instalación, para lo cual se selecciona la opción **I accept the License agreement(s)** y se pulsa sobre **Next**.

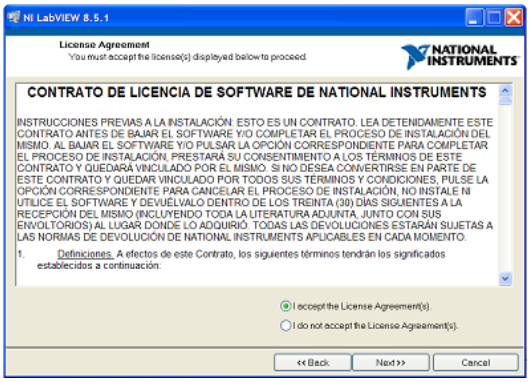

Figure 1.8

Paso 9. La ventana de Figure 1.9, nos indica que estemos al tanto del proceso de la instalación, ya que durante la misma se nos pedirá la inserción del CD que contiene unos drivers, aunque en nuestro caso no será necesario hacerlo. Pulsando **Next**, es cuando comienza realmente la istalación de **LabVIEW**.

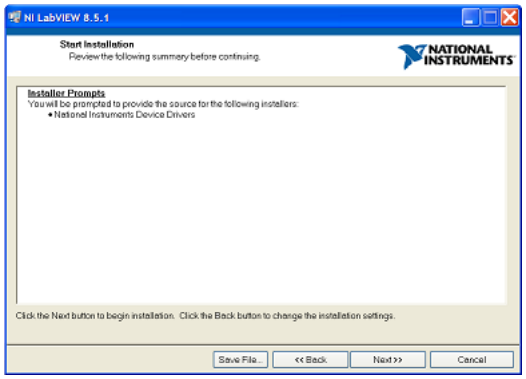

Figure 1.9

Paso 10. La instalación de **LabVIEW** muestra el aspecto que muestra la Figure 1.10 y puede tardar entre 30 y 60 minutos.

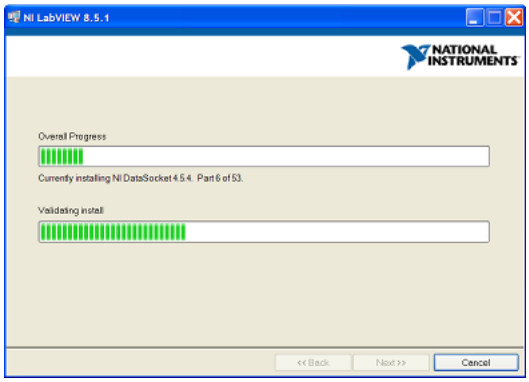

Figure 1.10

Paso 11. Cuando culmina la instalación, aparece la ventana de finalización de la instalación (Figure 1.11) donde la opción **Launch Activation Wizard** estará activada (en nuestro caso).

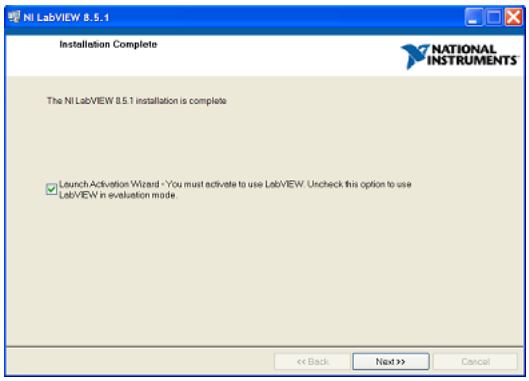

Figure 1.11

Paso 12. Esta opción permite activar la instalación con licencia por parte de **National Instruments**, para lo cual se ofrecen tres posibles formas para ponerse en contacto la empresa creadora de **LabVIEW**. En nuestro caso, la opción elegida es la de internet **Automatically activated through a secure internet connection**, tal y como se muestra en Figure 1.12, tras lo cual se pulsará **Next**.

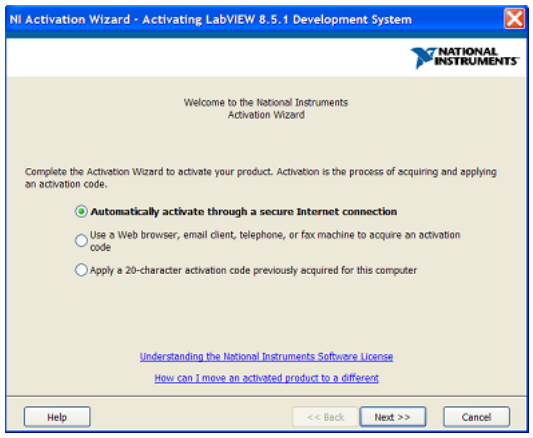

Figure 1.12

Paso 13. En la ventana de Figure 1.13, hay que introducir los datos de usuario, y pulsar **Next**.

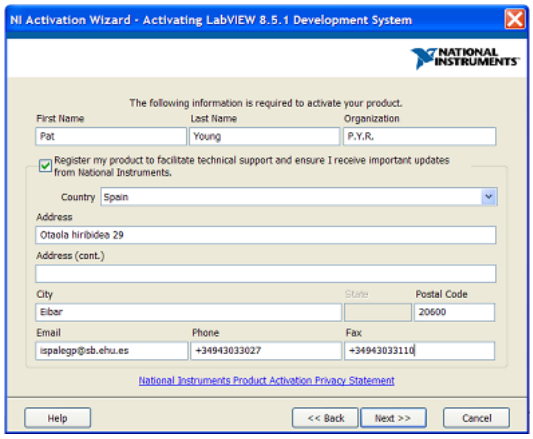

Figure 1.13

Paso 14. A continuación se confirma la solicitud de activación pulsando **Next** y aparece la ventana de Figure 1.14, donde nos indica que este paso puede llevar unos pocos minutos.

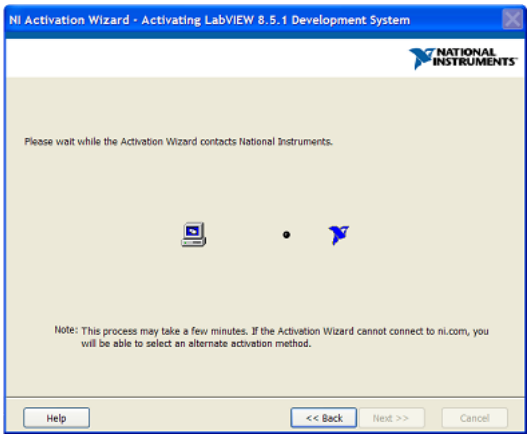

Figure 1.14

Paso 15. Al finalizar la activación, se podrá ver la ventana de Figure 1.15 donde nos indica que la activación se ha llevado a cabo con éxito, y una dirección de internet donde acudir por si tenemos algún problema. Pulsar **Finish**.

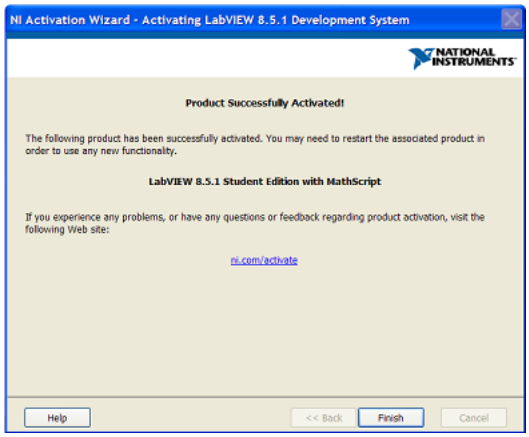

Figure 1.15

Paso 16. Después de la instalación, se nos pedirá reiniciar el ordenador, para lo cual hay que pulsar sobre **Restart**, Figure 1.16.

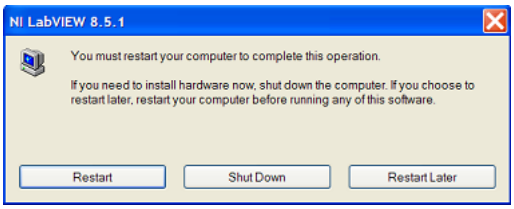

Figure 1.16

Tras el reinicio del ordenador, en **Inicio/Todos los Programas** de **Windows**, podremos comprobar cómo **LabVIEW** ha sido instalado correctamente.

Si por la razón que sea, se desea desinstalar **LabVIEW** de nuestro ordenador, hay que seguir los **Pasos 1 al 6** del proceso de instalación de este módulo, y en el último, hacer click sobre el icono de **NI LabVIEW 8.5.1** y elegir la opción **Remove this feature**, pulsando sobre ella, tal y como aparece en Figure 1.17.

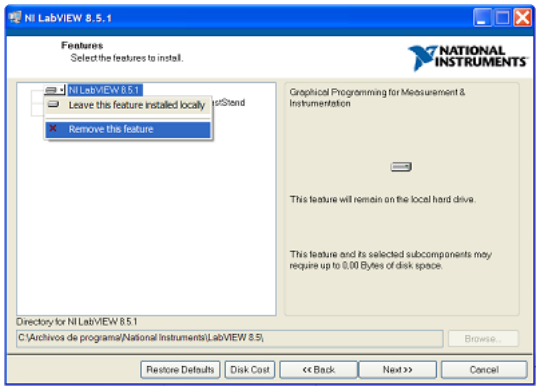

Figure 1.17

Chapter 2

Introducción a LabVIEW[1]

2.1 Introducción a la instrumentación

La instrumentación está presente en muchos ámbitos de nuestra vida. Un aparato de instrumentación consta básicamente de dos tipos de elementos: indicadores (monitores) y controladores (controles). Los indicadores muestran la información de una magnitud física. Por ejemplo, el indicador de velocidad de un automóvil monitoriza la velocidad a la que circula el coche en cada instante. El cuentarrevoluciones, muestra las revoluciones por minuto a las que gira el motor del coche. Los indicadores pueden mostrar la información de forma gráfica ó de forma numérica.

La forma clásica de los indicadores de velocidad y cuentarrevoluciones es gráfica: se trata de una aguja de tipo reloj que apunta sobre una cantidad escalada del valor de velocidad o revolución real, Figure 2.1. Pero actualmente también se utilizan indicadores numéricos para las mismas funciones: velocímetros y cuentarrevoluciones que muestran la información en números.

En cuanto a los controles, son elementos que, al contrario que los indicadores que sólo son capaces de mostrar la información de un proceso pero sin poder actuar sobre él, son capaces de incidir sobre el proceso y así poder controlar alguna de sus magnitudes. Por ejemplo, cuando viajamos en coche y accionamos el interruptor de luz de cruce, las luces de cruce se encienden.

Así mismo, cuando actuamos sobre el pulsador de un elevalunas concreto, esa luna comienza a cerrar o a abrir, según la orden haya sido dada en un sentido o en el otro. Cuando en verano hace mucho calor y viajamos en coche, establecemos la temperatura deseada en el interior del coche utilizando el fijador/regulador de temperatura del climatizador (aire acondicionado con termostato), y a continuación el dispositivo climatizador actúa en consecuencia proporcionando aire fresco hasta llegar a la temperatura deseada.

En los controles, se distinguen los de control todo-nada (interruptores, pulsadores, conmutadores), y los de control variable (reguladores, potenciómetros, ajustadores), los cuales permiten establecer el valor que se desea para una magnitud dada.

[1] This content is available online at <http://cnx.org/content/m18065/1.1/>.

Figure 2.1: Panel de un automóvil: cuentarrevoluciones y reloj (izda), velocímetro y kilometraje (dcha), ordenador de a bordo (centro abajo), temperatura del motor y estado del depósito de combustible (centro arriba).

Al igual que ocurre con el coche, hay otros ejemplos cotidianos que utilizan la instrumentación, como es el caso del televisor, el teléfono móbil, los frigoríficos más recientes, el GPS. En todos ellos, hay elementos indicadores y elementos controladores o actuadores.

La instrumentación clásica se ha utilizado sobre todo en la industria y en el control de procesos con varias magnitudes. Podemos imaginar o recordar cómo el control y monitorización de una máquina o proceso se hace normalmente a cierta distancia, de forma separada, desde donde realmente se encuentra el mismo. Es el caso de un alto horno, donde debido a las altas temperaturas que utiliza, su control y monitorización se realiza desde el cuarto o sala de mandos: allí habrá un gran panel de indicadores y controladores del alto horno, donde los técnicos destinados en esa tarea estarán vigilando y actuando sobre él de forma constante. Lo mismo ocurre con la sala de máquinas de un barco: hay infinidad de variables críticas que hay que visualizar para vigilarlas y en consecuencia realizar varios controles, si se da el caso. En este sentido, el panel asociado a la sala de máquinas se encontrará situado en otro habitáculo separado, desde donde se tiene toda la información relevante del estado de las máquinas.

En este sentido, las magnitudes que interesa medir, son convertidas en señales eléctricas de corriente continua proporcionales a las mismas mediante sensores, y llevadas al panel para conectarlas a los indicadores con entrada eléctrica. En cuanto a las magnitudes que interesa controlar, son modificadas mediante actuadores de entrada eléctrica y salida igual a la naturaleza de la magnitud. Cuando la instrumentación es clásica, estas señales eléctricas son de tipo analógico y los paneles son rígidos, es decir, no se pueden modificar o ampliar, y normalmente sólo sirven para monitorizar el proceso para el que han sido creados.

Pero en los últimos años, debido a la tendencia de bajada de precios y altas prestaciones que ofrecen los ordenadores actuales, se está utilizando cada vez más la instrumentación virtual. Esto significa, que los viejos y rígidos paneles están siendo sustituídos por pantallas de ordenadores que contienen paneles virtuales: con el mismo aspecto y funcionalidad que los antiguos paneles, pero con la ventaja de poder realizar cambios de forma inmediata y sin coste alguno.

Además, se pueden cambiar las formas, formatos y colores de los indicadores y controles, persolnalizándolos y dándoles un aspecto más moderno. La instrumentación virtual tiene la misma base que la clásica, sólo añade conversores analógico-digitales (ADC) para la monitorización y conversores digital-analógicos (DAC)

para el control. Esto se debe a que el ordenador, al ser un elemento basado en microprocesador, maneja solamente información digital, por lo que las señales eléctricas a monitorizar han de ser digitalizadas previamente mediante un ADC. Con las magnitudes a controlar sucede algo similar: su correspondiente señal eléctrica es tratada de forma digital por el instrumento virtual (ordenador), pero cuando ha de ser enviada al actuador, se tiene que convertir en una señal analógica mediante un DAC.

Las aplicaciones realizadas con **LabVIEW** son conocidas como Instrumentos Virtuales (Virtual Instruments, VI) y se suelen ejecutadar en un PC. Esto se debe a que estas aplicaciones tienen aspecto de instrumento de medida/control, a través de una pantalla de PC que se denomina Panel (debido a su gran parecido a los paneles de instrumentación clásicos de hardware). En cuanto al término Virtual, indica que se trata de la versión software o flexible del clásico instrumento de medida/control. Lógicamente, un VI también necesita un hardware básico para poder realizar medidas de señales reales y monitorizarlas, e incluso puede realizar control sobre éstas señales. Este hardware básico consta, en la mayoría de los casos, de una tarjeta de adquisición de datos (DAQ), aunque también es posible simular su funcionamiento si no se dispone de ella.

A continuación, vamos a ver cómo se genera una sencilla aplicación VI (aplicación de instrumentación virtual). Durante el curso no se utilizarán o realizarán aplicaciones con DAQs, debido a que se supone que la mayoría de los usuarios o alumnos no dispone de una DAQ. Pero sí se van a realizar ejercicios simulando que se tiene una DAQ, lo cual es tan valido como tener una DAQ real para la comprensión de su utilización.

2.2 Comenzando con LabVIEW

Para realizar cualquier aplicación VI hay que ir a **Inicio/Todos los Programas** en **Windows**, y ejecutar **National Instruments LabVIEW 8.5.1**. Tras una breve espera, aparece la pantalla que muestra Figure 2.2, donde se clickará con el ratón sobre **Blank VI**, en **Files/New**

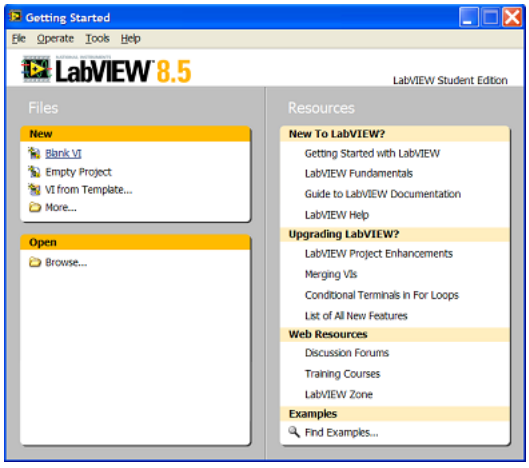

Figure 2.2

Seguidamente aparece la aplicación VI en blanco, con tres ventanas. La **Front Panel** (panel), es el panel del VI propiamente dicho: es donde el operario o usuario final de la aplicación podrá monitorizar algunas variables y desde donde podrá controlar otras. La ventana **Block Diagram** (diagrama) es la que contiene el código fuente para que el panel funcione correctamente, Figure 2.3.

Este código fuente es de tipo gráfico, y por lo tanto muy intuitivo: sólo son necerarios unos pocos y sencillos conceptos de programación. El tercer y último elemento es **Controls**, la paleta flotante de controles e indicadores. Esta paleta es válida para el panel, pero si hacemos activa la ventana de diagrama, entonces la paleta se convierte en **Functions**, y es valida sólo para el diagrama. Esta paleta contiene librerías de funciones clasificados por tipos, al igual que ocurre con otros lenguajes de programación.

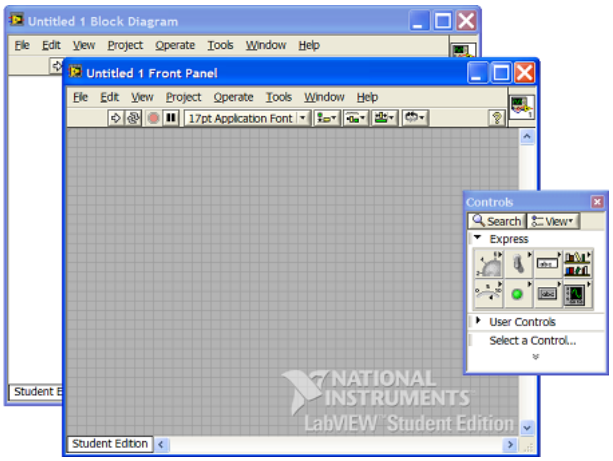

Figure 2.3: Aspecto inicial del archivo nuevo.

Lo más práctico y aconsejable suele ser separar el panel y el diagrama, para lo cual se pulsan las teclas **Ctrl+T**, Figure 2.4. Otra paleta que resulta imprescindible tenerla a la vista es la paleta **Tools**, para visualizarla basta con ejecutar **View/Tools Palette**, en panel o en diagrama, ver Figure 2.4 parte derecha y arriba.

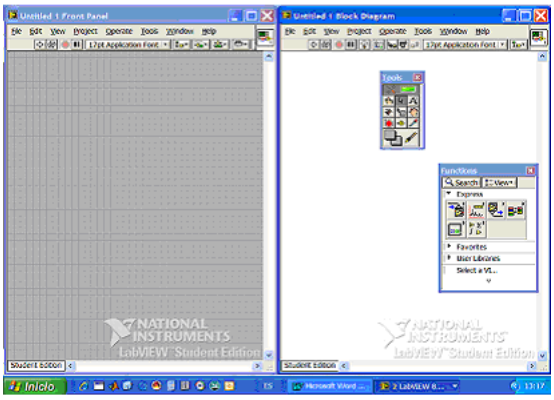

Figure 2.4: Aspecto del nuevo archivo tras ser separados el panel y el diagrama.

Realicemos una sencilla aplicación que suma los valores de dos sumandos dados y devuelve el resultado de la suma. En primer lugar, colocaremos los dos controles sumandos sobre el panel, para lo cual activaremos la ventana panel (clickando sobre ella), y desde la paleta de controles, seleccionaremos la primera opción de **Controls**, es decir **Express**, y allí **Numeric Controls**, y dentro de la misma, el objeto **Numeric Control**. Al hacer click sobre el objeto, lo arrastraremos al panel para que lo ubiquemos donde más lo deseemos, tras lo cual aparecerá también en el diagrama de la aplicación, Figure 2.5.

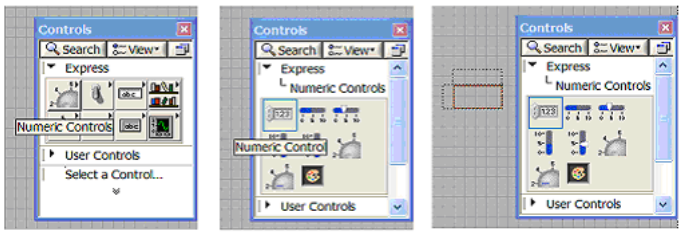

Figure 2.5

Al principio de la colocación, la etiqueta del objeto aparece marcada con su nombre por defecto, con lo que no está de más aprovechar que está marcada para darle el nombre que le queramos dar, p.e. **Sumando 1**, tecleando directamente en el teclado del PC. Se repetirá la operación con **Sumando 2**, Figure 2.6. De otra forma, para cambiar la etiqueta de un objeto, se marca sobre la etiqueta haciendo dos clicks seguidos con el ratón.

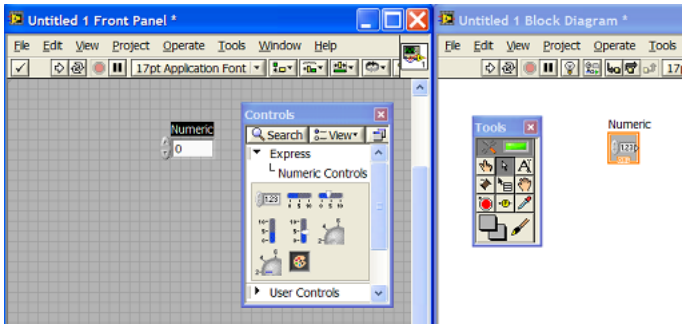

Figure 2.6

A continuación vamos a ubicar un indicador que albergue el resultado de la suma. En **Controls**, **Select a Control** y seleccionar **Modern**, tras lo cual seleccionar y ubicar en el panel un objeto de tipo **Numeric Indicator**, Figure 2.6.

Figure 2.7

Ahora procederemos a realizar el código fuente en el diagrama, el código que define la operación que se va a realizar con estos objetos. Para ello, si hace falta, cambiaremos de ubicación los objetos, tanto en el panel como en el diagrama. Seleccionaremos la ventana del diagrama e insertaremos la función suma desde la paleta **Functions** en **Express**, **Arithmetic & Comparison**, **Express Numeric**. A continuación se selecciona y se ubica en el diagrama en elemento **Add**, Figure 2.8.

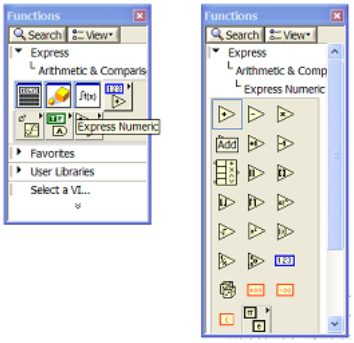

Figure 2.8

Seguidamente, se procederá a conectar las dos entradas y la salida utilizando la herramienta **Connect Wire de Tools**: se trata de picar y arrastrar para unir los terminales que interesa (una salida con una entrada). Esto se puede hacer directamente, clickando en la salida y arrastrándola hasta la entrada que interese, donde el cambio de trayectoria de la conexión se hace automáticamente, o también se puede hacer def horma manual, decidiendo dónde será cada cambio de trayectoria, para lo cual se pulsará cada vez el botón izquierdo del ratón, Figure 2.9 y Figure 2.10.

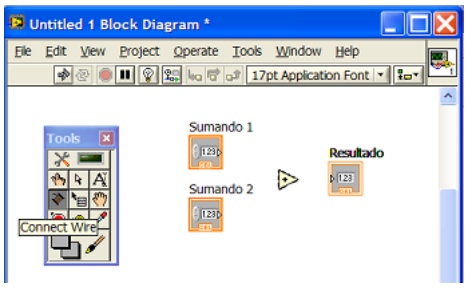

Figure 2.9

El cambio de trayectoria manual resulta interesante cuando se tienen muchas conexiones en un espacio reducido y es conviene llevar un orden para el futuro mantenimiento del código fuente: no interesa que las conexiones aparezcan cruzadas entre ellas.

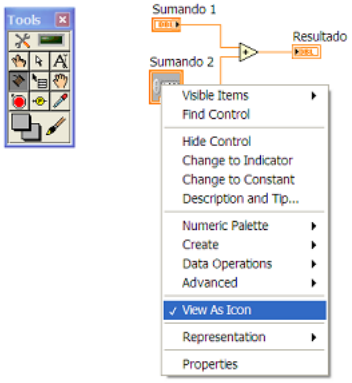

Figure 2.10

Para que el diagrama ocupe menos espacio que el debido, es buena práctica reducir el tamaño de los iconos. Para ello, hay que ubicarse con el cursor del ratón sobre el icono y al hacer click con el botón derecho aparece un menú flotante, donde se elige desactivar **View As Icon**, Figure 2.10. Para borrar una conexión, basta con marcarla con la herramienta **Position/Size/Select**, Figure 2.11, y aparecerá marcada con línea discontinua, tras lo cual se pulsa la tecla **Supr** para eliminarla.

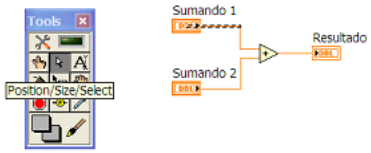

Figure 2.11

Si se comienza a establecer una conexión y se da cuenta de que no es la que se deseaba, ésta es eliminada tras pulsar la tecla **Esc**. La aplicación **LabVIEW** no permite conectar dos entradas ni dos salidas entre sí. Si se produce algún caso de conexión no permitida en el diagrama, las conexiones implicadas en el conflicto aparecen en línea discontinua con una x en color rojo indicando el lugar exacto del error o conflicto, Figure 2.12. Además, el icono del botón **Run**, que es una flecha gruesa, aparece rota, Figure 2.13.

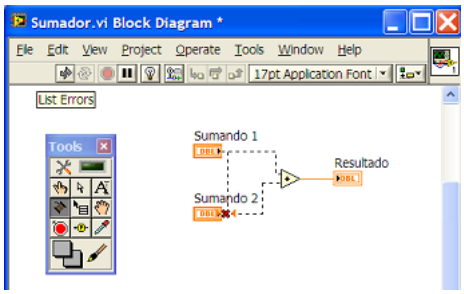

Figure 2.12

Figure 2.13: Pulsador Run partido, lo cual indica la existencia de al menos un error en el conexionado del diagrama.

Esto indica que la aplicación desarrollada contiene errores y no se puede ejecutar. Si se pulsa este botón (**Run**) en estas condiciones, aparecerá la ventana de listado de errores (Figure 2.14) que impiden la ejecución normal de la aplicación. De cada item, al ser marcado en la subventana central **Show Warnings**, se da una breve descripción de la causa del error en la subventan inferior. Al hacer doble click en un item de error en la subventana **Show Warnings** de **Error list**, **LabVIEW** señala en el diagrama dónde se encuentra exactamente. Esto favorece una rápida depuración de la aplicación realizada y es muy útil cuando el diagrama que estamos contruyendo es muy grande, donde no suele ser posible ver todos los errores de la aplicación en la misma pantalla.

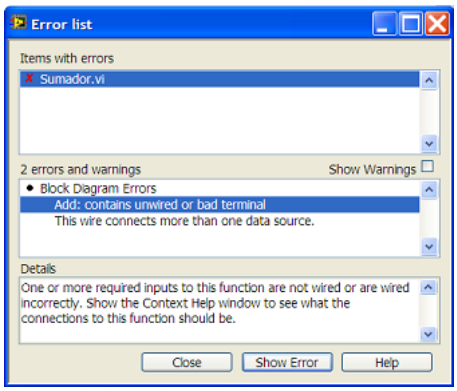

Figure 2.14: Ventana de listado de errores.

2.3 Más rápido y más estético

Otra forma más rápida de realizar un módulo es la siguiente. Se ubica el operador suma en el diagrama, se selecciona la herramienta **Connect wire**, y colocándolo encima de cada terminal del sumador, se pulsa el botón derecho del ratón, donde en el menú flotante que aparece, se selecciona **Create/Indicator**, en el caso de la salida, Figure 2.15, y **Create/Control**, en el caso de las dos entradas. En las aplicaciones donde los valores de las entradas sean siempre constantes, entonces se elegirá **Create/Constant** para éstas.

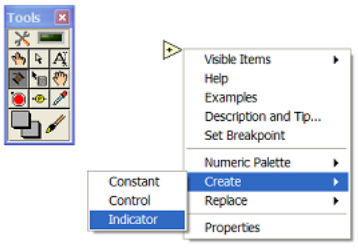

Figure 2.15

Finalmente, el aspecto del módulo será el que aparece a la izquierda de Figure 2.16, que tras deseleccioar la vista de los sumandos y resutado del modo icono, queda como se muestra en el centro de Figure 2.16. Cuando hay muchas señales de entrada y salida colocadas en columna, suele ser una buena práctica, con la intención de aprovechar el espacio disponible, desplazar la etiqueta que lleva su nombre, a la izquierda de cada elemento entrada, y a la derecha de cada elemento de salida, Figure 2.16.

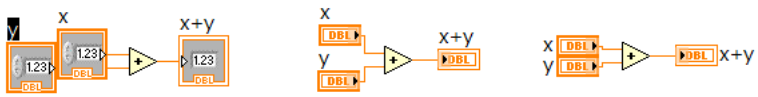

Figure 2.16

Al mismo tiempo, en el lado del panel, también es conveniente llevar un orden de colocación para mantener a estética del instrumento virtual. Se trata de de ir alineando los objetos del mismo tipo, en este caso los dos sumandos, para lo cual son previamente seleccionados con el ratón (definiendo un área que los incluya o los toque). A continuación se pulsa sobre el comando **Align Objects**, y se elige la opción **Left Edges** seguidamente, como los sumandos permanecen aún marcados y la distancia hasta el indicador de salida se estima que es muy grande, se pica sobre ellos y son arrastrados a un punto más cercano a la salida. Ahora, se seleccionan el primer sumando y la salida, ya que no están alineados horizontalmente, y se ejecuta de nuevo el comando **Align Objects**, donde ahora se elige la opción **Top Edges**, Figure 2.17.

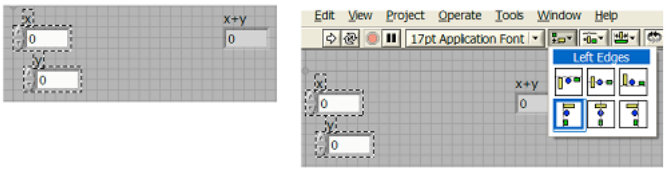

Figure 2.17

El aspecto final del panel es el que muestra la Figure 2.18, donde todos sus elementos aparecen alineados.

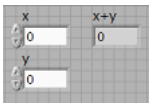

Figure 2.18

2.4 Ejecución de la aplicación

Una vez finalizada la aplicación y en ausencia de errores de conexionado, se puede proceder a su ejecución. Pero antes, es conveniente guardar la aplicación desde el menú **File/Save**. En este caso, el nombre que le daremos a la misma será **Sumador.vi** (la extensión .vi será proporcionada por **LabVIEW** de forma automática), Figure 2.19.

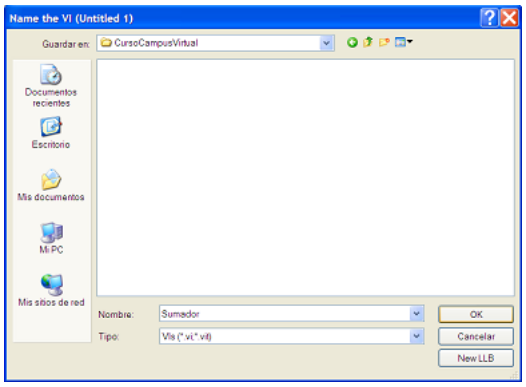

Figure 2.19

El comando que hace posible la ejecución de la aplicación es **Run**, donde se puede ejecutar pulsando el botón **Run**, Figure 2.20, o desde el menú **Operate/Run**.

Figure 2.20: Pulsador Run.

Pero antes, es conveniente dar dos valores distintos de 0 a los dos sumandos, utilizando la herramienta **Operate Value**, Figure 2.21, con la cual pulsando sobre los variadores de valor de los controles **Sumando1** y **Sumando2** se podrán cambiar sus valores. No obstante, esto valores son enteros, si se deseara que fueran números con decimales, habría que clickar con el cursor del ratón sobre la casilla del sumador en cuestión y escribir desde el teclado. Notar que para el punto decimal se utiliza la coma.

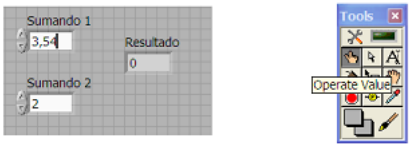

Figure 2.21

A continuación se pulsa **Run** y el resultado es el mostrado por Figure 2.22.

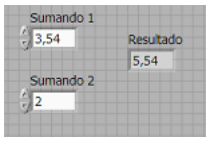

Figure 2.22

En realidad, la aplicación se ha ejecutado una sola vez y después se ha salido del modo de ejecución. En la práctica, una aplicación VI se está ejecutando continuamente, ya que si no no tiene sentido alguno. En el caso de la aplicación **Sumador.vi**, lo suyo sería que se estuviera ejecutado en todo momento para que nos proporcionara en tiempo real de ejecución el resultado de la suma de los dos sumandos que en cualquier momento podríamos cambiar, como si fuera una calculadora que está encendida o funcionando.

En **LabVIEW** existen dos formas para que la ejecución de una aplicación sea continua. La primera, la ofrece directamente **LabVIEW** con la opción **RunContinuously**. Para detener la ejecución, hay que pulsar sobre el **botón Abort Execution**, Figure 2.23.

Figure 2.23: Pulsador Run Continuously, izquierda, y Abort Execution, derecha.

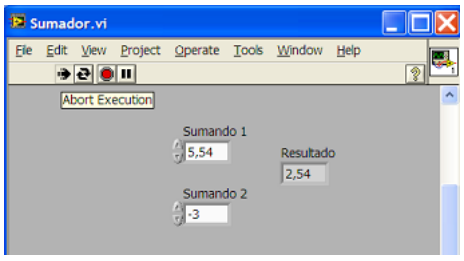

Figure 2.24

La segunda forma de ejecución continua de las aplicaciones VI de **LabVIEW**, es seguramente la más utilizada y extendida entre los programadores de los VI de esta marca. Se trata de utilizar el comando **Run** pero habiendo construido previamente una sentencia repetitiva o bucle de tipo **while** donde se inserta todo el código de nuestra aplicación. Este bucle se ejecutará en todo momento mientras no se pulse el pulsador, normalmente de tipo **STOP**, conectado a su condición de permanencia. Para ello, se pulsa sobre el diagrama con el botón derecho del ratón y se elige la función **While Loop**, Figure 2.25, y se lleva o arrastra al diagrama, para rodear al todo el código de nuestra aplicación, picando en un extremo y arrastrando hasta el otro, que se encuentra en su diagonal.

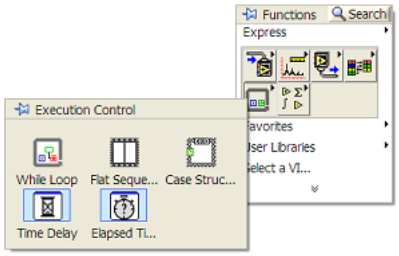

Figure 2.25

Como se puede observar, el pulsador **STOP** para detener el bucle while aparece tanto en el diagrama, como en el panel (desde donde será pulsado), Figure 2.26. En este segundo caso, la ejecución (continua) dará comienzo pulsando el comando **Run**, y terminará pulsando el pulsador **STOP**.

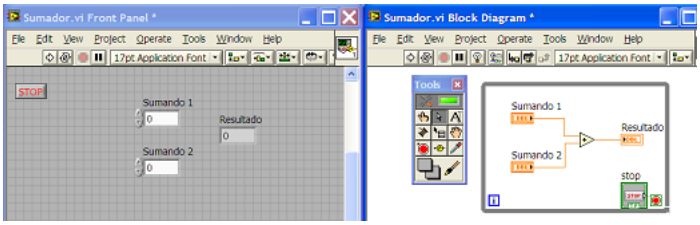

Figure 2.26

2.5 Ejercicios propuestos

Exercise 2.1

Observa a tu alrededor y cita al menos diez artilugios o aparatos que presenten un panel de instrumentación, real o virtual, identificando y citando los diversos elementos indicadores y controladores que dispongan.

Exercise 2.2

Desarrollar la aplicación **Calculadora.vi** que efectúe las cuatro operaciones aritméticas (suma, resta, producto y división) con dos operandos y proporcione el resultado en cuatro indicadores, uno por operación, Figure 2.27.


Calculadora1.vi
File Edit View Project Operate Tools Window Help
STOP
Operando1
-0,77
Operando2
2,55
Suma
1,78
Resta
-3,32
Producto
-1,963
División
-0,301

Figure 2.27

Chapter 3

Programación modular con LabVIEW[1]

3.1 Programación modular

Como en el resto de los lenguajes de programación, también en **LabVIEW**, el concepto de programación modular descansa sobre el elemento función. Esto quiere decir que la programación modular se basa en la programación con funciones, es decir, que la función es la base de la programación modular. Muchas de estas funciones son proporcionadas por el propio fabricante de software, y se presentan ordenadas por tipos en las librerías de funciones. Estas librería pueden ser vistas si se pulsa con el botón derecho del ratón sobre cualquier punto del diagrama de nuestra aplicación, a continuación se pulsa en la opción final del menú flotante (doble ^ o extensión), con lo que se muestra el listado de las librerías de funciones. La mayoría de las funciones que utilizaremos en este curso, serán las de la librería **Programming** (programación), Figure 3.1.

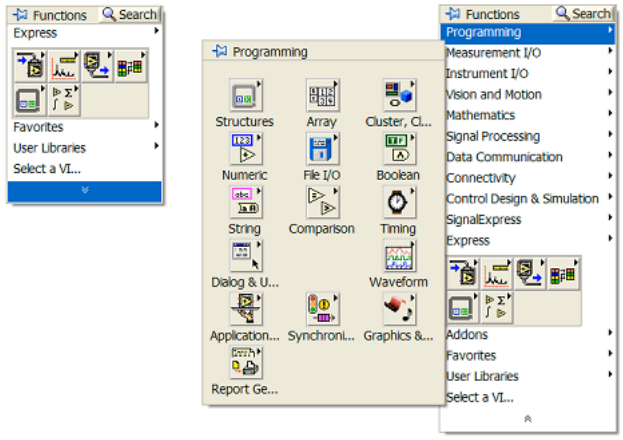

Figure 3.1

Como se puede observar, si se despliega esta librería, las funciones aparecen ordenadas por tipos. Por ejemplo, en el grupo **Structures** (estructuras), nos encontramos con las sentencias de repetición **while** y **for**, las sentencias de secuenciación, las de selección, etc, con las que trabajaremos más adelante.

[1] This content is available online at <http://cnx.org/content/m18068/1.1/>.

Por otro lado, el programador tiene la capacidad de generar por sí solo las funciones que necesita para el desarrollo de sus aplicaciones. Cuando una misma tarea es requerida en varios puntos de la aplicación, en lugar de construir esa porción del código (diagrama) de forma repetida, lo que se suele hacer es construirla una vez y llamarla en todos los puntos de la aplicación que sea necesaria. Además, una vez construidas o realizadas las funciones, el programador las puede utilizar en otros proyectos de aplicaciones futuras. Incluso las puede mejorar y/o cambiar para utilizarlas en otras aplicaciones.

3.2 Creando un SubVI

Una función en el lenguaje de programación **LabVIEW**, es conocida con el nombre de **SubVI** (sub instrumento virtual). Veamos a continuación, cómo se construye una **SubVI** y lo fácil que resulta utilizarla. Para ello, se utilizará el ejercicio propuesto **Calculadora.vi** del módulo anterior y se guardará como **Calculadora1.vi**. En primer lugar se crea el código con el que se quiere hacer el **SubVI**, a continuación se marca esa porción o bloque de código con el ratón y se ejecuta **CreateSubVI**, desde el menú **Edit**, con lo que el código marcado queda asociado al nuevo **SubVI** con el icono standard de **LabVIEW**, Figure 3.2. Para cambiar este icono, se hace doble click sobre él, donde se abre el archivo del **SubVI** y se ejecuta **Ctrl+T** para ver tanto el panel como el diagrama de esta función. Tanto en el panel como en el diagrama, las salidas aparecen nombradas por defecto con **Numeric** (izda), cuando lo que interesa es nombrarlas según su función (dcha), ya que de lo contrario cuando vayamos a utilizar esta función no sabremos qué proporciona en cada una de las salidas, Figure 3.3.

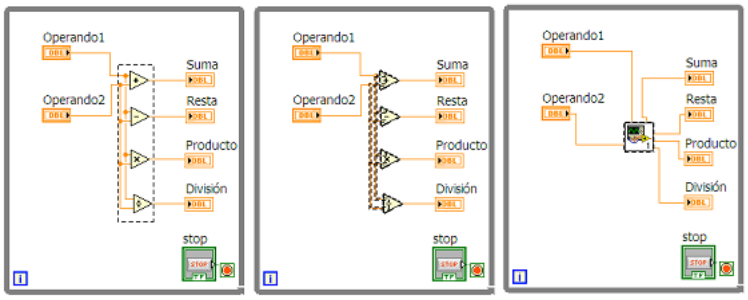

Figure 3.2

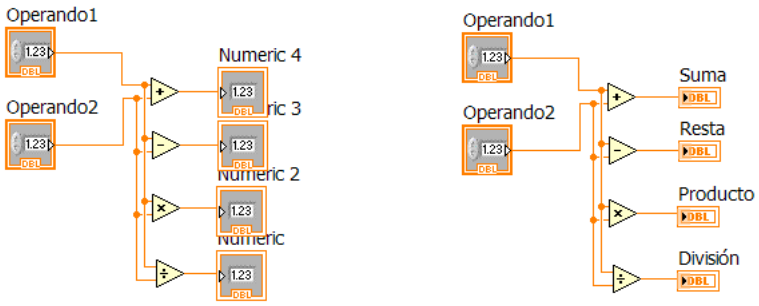

Figure 3.3

A continuación, se pulsa el botón derecho de ratón sobre el símbolo del icono (arriba a la derecha del diagrama) y se ejecuta **Edit Icon**, tras lo cual aparece la ventana de definición del icono del **SubVI**, Figure 3.4. Este editor de iconos es muy fácil de utilizar, ya que es muy intuitivo. Para eliminar el contenido del icono por defecto, se marca todo el contenido del mismo con la herramienta selección, habiéndo seleccionado previamente en la paleta de herramientas, y pulsando **Supr**. A partir de ahí, se puede escribir desde el teclado, para lo cual ha de seleccionarse la herramienta texto , o también se puede dibujar, utilizando la herramienta lápiz, Figure 3.5.

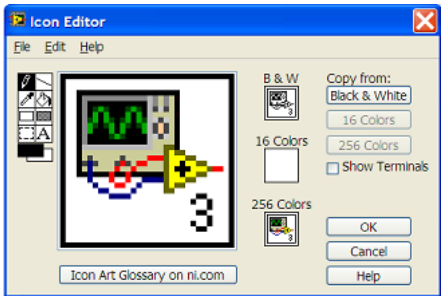

Figure 3.4: Editor de iconos de LabVIEW.

Figure 3.5: Herramientas de selección, lápiz y texto.

Para el caso que nos sigue, se utiliza la herramienta lápiz, con el que se dibujan los símbolos de las cuatro operaciones aritméticas, Figure 3.6. Tras ello se pulsa **OK**, donde se puede observar que el icono que aparece

ahora tanto en esta función como en la aplicación donde la hemos creado (**Calculadora1.vi**) es el definido por nosotros. A continuación, se guarda el archivo **SubVI** con el nombre **Calc4 (SubVI).vi**. Notar que el sufijo **SubVI** que aparece entre paréntesis, en el nombre de la función, es insertado por la herramienta **LabVIEW** de forma automática.

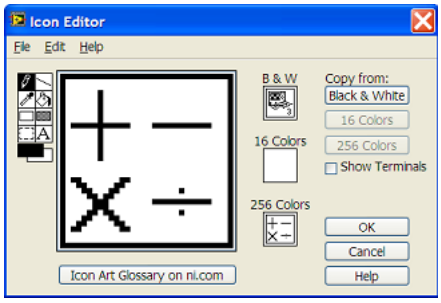

Figure 3.6

El aspecto final que mostraría la aplicación utilizando el **SubVI Calc4** sería el que aparece en la Figure 3.7 izquierda. Conviene comentar que esta función ha sido creada desde la aplicación **Calculadora1.vi**, pero a partir de ahora, cuando es insertada para ser utilizada en alguna otra aplicación, habrá que identificar cada una de las entradas y salidas, por lo que tendrá que ser expandida desactivando su opción **View As Icon**, Figure 3.7 derecha. Ésto también es válido para las funciones de librería de **LabVIEW**.

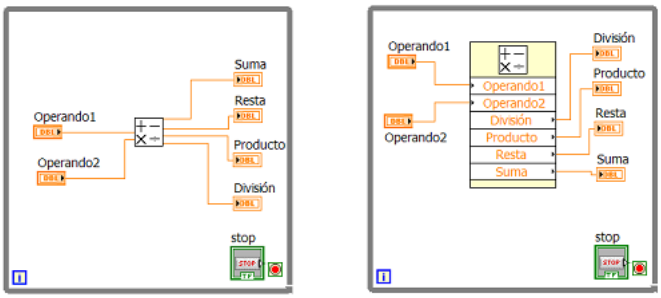

Figure 3.7

A partir de ahora, cada vez que se desee insertar cualquier **SubVI** previamente definido en cualquier aplicación que tengamos abierta en ese momento, basta con ejecutar **Select a VI**, del menú flotante de **Functions** del diagrama, Figure 3.8. En realidad, esta forma sirve para introducir tanto **SubVI**s como **VI**s, en la aplicación abierta. Es conveniente comentar, que en este caso concreto, en un principio podríamos pensar que se podía haber introducido el **VI Calculadora1.vi** directamente, el ejercicio propuesto en el módulo anterior. Lo que ocurre es que tal y como se planteó, éste se ejecuta dentro de una sentencia **while**, por lo que a nosostros nos sobra esta sentencia y por ello no nos sirve incluir este proyecto directamete. En el caso de que ese proyecto no incluyera la sentencia **while**, entonces sí hubiese servido de **SubVI**.

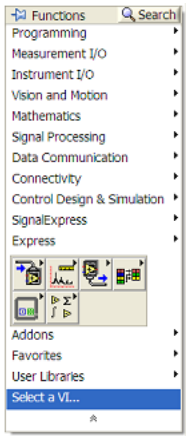

Figure 3.8

Hay que añadir, que la generación de iconos personalizados no está restringida solamente a **SubVI**s, también es posible hacerlo en cuanquier **VI**.

De esta manera, utilzando las **SubVI**s, es posible establecer varios niveles de llamadas entre **SubVI**s, al igual que ocurre con las funciones en los lenguajes de programación estructurada: una función es llamada desde una aplicación, y ésta a su vez llama a otra y así sucesivamente. Ejecutando **View/VI Hierarchy** es posible visualizar la jerarquía de llamadas entre **SubVI**s de una palicación dada. La Figure 3.9 muestra la jerarquía del ejemplo que nos ocupa. Notar que el nivel superior es el entorno de desarrollo **LabVIEW** de la aplicación **Calculadora1.vi**, y el inferior **Calc4.vi**.

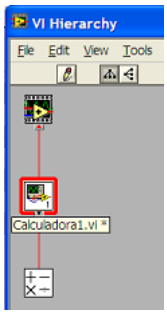

Figure 3.9: Jerarquía de la nueva aplicación Calculadora1.vi

3.3 La ayuda de LabVIEW

La herramienta **LabVIEW** dispone de una potente ayuda que facilita enormemente la tarea de programar. Cuando tenemos alguna duda, nos la puede aclarar acudiendo a ella. Cuando queremos ampliar nuestros conocimientos sobre las posibilidades que tiene la herramienta, también podemos acudir a ella, ya que en

realidad se trata de un libro en soporte informático. A continuación se van a describir las opciones más utilizadas de la ayuda de esta herramienta: el menú **Help**, accesible tanto desde el panel como del diagrama.

Help/Show Help Context, al ejecutarla, esta opción queda marcada y ello indica que la ventana de contexto (**Context Help**) está abierta. Esta ventana muestra la información resumida del objeto sobre el cual se sitúa el cursor del ratón, funciona tanto con objetos del panel como con los del diagrama. La Figure 3.10 muestra la información sobre el objeto divisor de dos operandos.

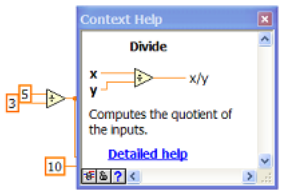

Figure 3.10

Si se quiere información más detallada, se pulsa sobre el link **Detailed help**, dentro de la ventana de contexto, y a continuación aparece otra ventana, la de ayuda de **LabVIEW** (**LabVIEW Help**), donde se muestra la información detallada, en este caso, de la función división, Figure 3.11.

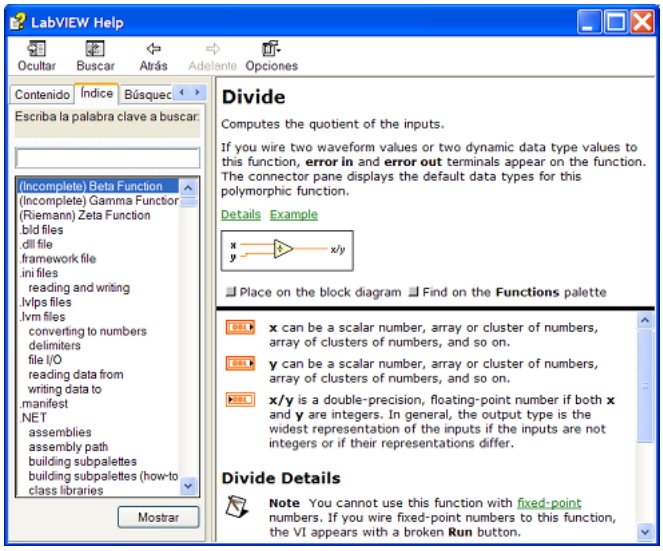

Figure 3.11

Help/Search the LabVIEW Help..., al ejecutar esta opción, aparece la ventana anterior **LabVIEW Help**. En su parte izquierda muestra tres lengüetas. Así, **Contenido**, como su nombre indica, muestra la información de las capacidades de **LabVIEW** organizada por temas y subtemas. **Índice**, sirve para realizar búsquedas por todo el contenido de la ayuda, y **Búsqueda**, para hacer una búsqueda más profunda.

Justo sobre la mitad de la ventana, en el centro, aparece el botón **Place on the block diagram**, así, si se pulsa sobre él, el objeto o función del que se muestra la información es colocado sobre el diagrama, sin tener que saber en qué submenú de **Functions** se encuentra exactamente. Ahora bien, si se desea saber dónde se encuentra exactamente (para ir conociendo la paleta de las funciones), se pulsará sobre el botón **Find on the Functions palette**, donde a continuación aparecerá la ventana **Functions**. Siguiendo con el ejemplo de la función **Divide**, si se desea saber dónde está exactamente esta función, al pulsar el mencionado botón, aparece la ventana de Figure 3.12. Así podemos desplazarnos hacia arriba a través de esta paleta, mediante el coamdo flecha hacia arriba, y conocer exactamente su ubicación en ella.

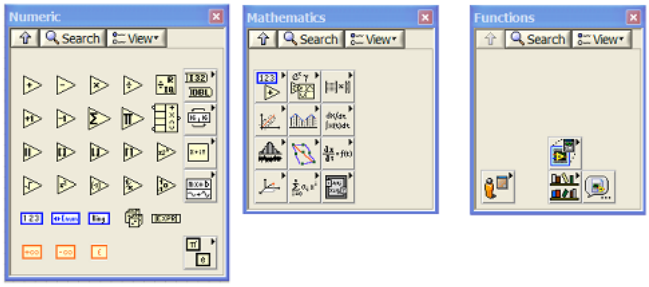

Figure 3.12

3.4 Ejercicios propuestos

Exercise 3.1

Explora en la paleta de funciones **Functions** los menús **Programming** y **Mathematics** e intenta comprender las funciones, una a una, que aparecen en ellas. Utiliza para ello el conocimiento que tienes de haber cursado las asignaturas de tu carrera. Seguro que hay funciones que las identificarás nada más verlas, y otras, aunque te lleven algún tiempo, también las acabarás identificando, aunque posiblemente no las identifiques todas. Intenta también, escribiendo en una hoja, dibujar el arbol organizativo de los menús que se te piden conocer.

Exercise 3.2

Realizar un **VI**, **Calculadora2.vi**, que contenga a un **SubVI** llamado **AriTriRaInv.vi** y que realice 3 tipos de operaciones definidas en 3 **SubVI**s de un nivel inferior: **SubVI Calc4.vi** que realiza las cuatro operaciones aritméticas con dos operandos (1 y 2) y que proporciona un resultado por operación, **SubVI Trig4.vi** que realiza las cuatro operaciones trigonométricas de seno, coseno, tangente y cotangente con el operando 1 con el ángulo en grados, y que también proporciona un resultado por operación, y, **SubVI RaiztInv.vi**, que realiza las operaciones de raíz cuadrada e inversa con el operando 2, y que proporciona un resultado por operación. Desplegar en pantalla la jerarquía de **SubVI**s y meditar sobre el resultado.

Exercise 3.3

Busca y asimila la información, utilizando la ayuda de **LabVIEW**, acerca de las operaciones con matrices (matrix) suma (add), resta (substract), producto (product), determinante (determinant), inversa (inverse) y transpuesta (transpose). Esta información será de gran ayuda para realizar el último ejercicio del tema de los arrays y clusters.

Chapter 4

Sentencias en LabVIEW[1]

4.1 Sentencias de selección

Se trata de un tipo de sentencia que permite ejecutar un código dado u otro, según el valor que tenga la variable de entrada de la sentencia de selección. Este tipo de sentencia se llama **Case Structure** y se encuentra en **Functions/Programming/Structures**. Esta sentencia puede ser utilizada para dos casos, o para más de dos o casos múltiples.

En el ejemplo que se muestra en Figure 4.1, se pretenden obtener solo los valores aleatorios entre 0 y 0'7 a partir de la función que genera números aleatorios entre 0 y 1 (**Functions/Programming/Numeric/Random Number**). Entonces, se recurre a la función de comparación **Greater or Equal?** de la paleta **Functions/Programming/Comparison**, ya que ésta proporciona una salida booleana **true**, si 0'7 es mayor o igual que el valor aleatorio generado, y **false**, si no es así. En este sentido, la salida en el caso true será el valor aleatorio generado, y en caso contrario, 0'7 , ya que en ese caso el valor aleatorio es superior a éste.

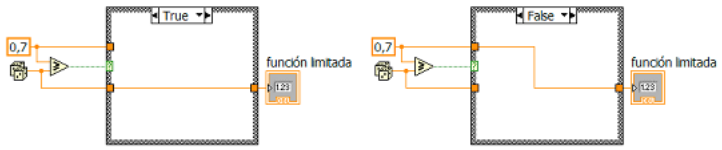

Figure 4.1

Para los casos de más de dos entradas, es decir, los de entradas múltiples, se inserta una sentencia case en el diagrama. Inicialmente éste está configurada para el primer caso, con lo que tendremos que modificarlo. En primer lugar, se cambia la entrada de tipo booleano a tipo constante entero: se crea una constante de tipo entero en el diagrama desde **Functions/Programming/Numeric/Numeric Constant**. Si se desea que este elemento pase a ser un control, solo hay que sustituirlo por un elemento de ese tipo. Seguidamente, clickando sobre el **Selector Label** con el botón derecho se elige la opción **Duplicate Case**, donde al repetir la operación se van añadiendo los caso 2, 3, 4 ..., Figure 4.2. Ahora solo queda incluir en cada caso el módulo de código correspondiente.

[1] This content is available online at <http://cnx.org/content/m18070/1.1/>.

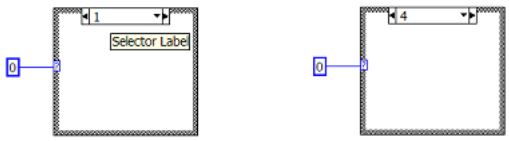

Figure 4.2

Para borrar uno de los casos, se repite la operación, pero ejecutando en ese caso **Delete Case**. La Figure 4.3 muestra el panel resultado final en funcionamiento.

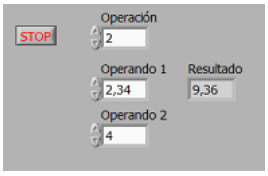

Figure 4.3

4.2 Sentencias de secuenciación

Se trata de un tipo de sentencia que garantiza la ejecución secuencial (uno detrás de otro) de varios módulos de código. Cuando en el diagrama de **LabVIEW** son definidos varios módulos de código, en realidad no se sabe en qué orden van a ser ejecutados, es decir, no se sabe cuál de ellos se ejecutará en primer lugar y cuál en el último lugar. Hay aplicaciones en las que este orden de ejecución no es importante, pero en otras es de vital importancia. Por ello, en las aplicaciones donde el orden de ejecución de los distintos módulos es crítico, resulta imprescindible utilizar las sentencias de secuenciación para garantizar esa secuencia de ejecución de los módulos.

La herramienta de desarrollo de aplicaciones **LabVIEW** ofrece dos sentencias de secuenciación. La primera es la **Flat Sequence Structure** y se encuentra en **Functions/Programming/Structures**. Cuando es insertada en el diagrama de nuestra aplicación, aparece con una única ventana o **Frame**: es la unidad o elemento donde irá un módulo dado del código de nuestra aplicación. Para añadir más ventanas, se pulsa sobre la ventana con el botón derecho del ratón y se ejecuta **Add Frame Before** o **Add Frame After**, para añadir la ventana antes o después de la actual, respectivamente. Veamoslo con un ejemplo. Pensemos en dos operaciones aritméticas, suma y producto, que han de realizarse con dos operandos independientes en cada caso, donde primero se tiene que realizar la suma y luego el producto, Figure 4.4. Notar que las conexiones de los operandos han de realizarse desde los controles o exterior, hacia el interior de las ventanas.

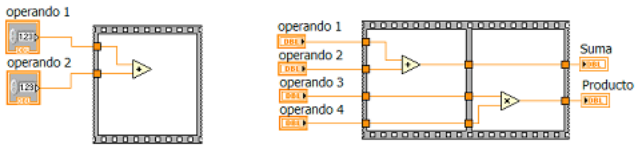

Figure 4.4

Cuando son necesarias muchas ventanas, porque así lo requiere la aplicación, aparece el problema del espacio en el diagrama, de tener que desplazar varias pantallas desde el inicio de la secuencia hasta su final. Entonces, lo lógico es sustituir la sentencia **Flat Sequence Structure** por la sentencia **Stacked Sequence Structure**, la cual funciona exactamente igual que la anterior, solo que apila las ventanas una encima de otra. Aunque a simple vista no sea posible ver la secuencia, resulta muy práctico su uso cuando el espacio disponible es pequeño. Repitamos el ejercicio anterior con esta nueva sentencia.

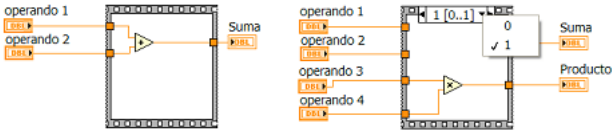

Figure 4.5

En este caso, al añadir la segunda ventana después de la primera, automáticamente aparece un índice para seleccionar cada una de las ventanas y acceder a su código, Figure 4.5.

4.3 Sentencias de repetición

Se trata de sentencias que permiten la ejecución repetida de un módulo de código dado, mientras se cumpla la condición de ejecución de la sentencia (condición de permanencia en la sentencia). Los hay de dos tipo: **for** y **while**.

4.3.1 For

La sentencia iterativa for, al igual que el resto de las sentencias, se encuentra en **Functions/Programming/Structures**. Se trata de una sentencia que repite N veces la ejecución del módulo de código que se encuentra en su interior. Este parámetro, es fijado bien mediante una constante o bien mediante un control, y está asocido al índice i de la sentencia. El parámetro i parte de 0, y tras ejecutarse el código de la sentencia y comprobarse que aún es menor que N, se incrementa en una unidad y se repite la operación. Así hasta que i es igual que N, entonces, deja de ejecutarse la sentencia y el código que ella contiene. Veámoslo con un ejemplo sencillo donde N=3 y el código consiste en multiplicar por 1 el valor del índice **i** y visualizarlo, Figure 4.6.

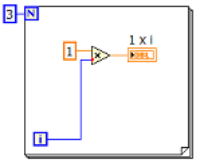

Figure 4.6

Al poner en ejecución esta aplicación, se inicializan los dos parámetros de la sentencia, N=3 y i=0, y a continuación comienza a ejecutarse la parte repetitiva o iterativa:

1. como se cumple que i<N, 0<3, entonces se ejecuta y se visualiza 1xi=1x0=0. A continuación se incrementa i, es decir, i=i+1=0+1=1
2. como se cumple que i<N, 1<3, entonces se ejecuta y se visualiza 1xi=1x1=1. A continuación se incrementa i, es decir, i=i+1=1+1=2
3. como se cumple que i<N, 2<3, entonces se ejecuta y se visualiza 1xi=1x2=2. A continuación se incrementa i, es decir, i=i+1=2+1=3
4. como no se cumple que i<N, 3<3, entonces no se ejecuta el código, ni tampoco se incrementa i, con lo que deja de ejecutarse la sentencia for.

Así, el resultado final de la ejecución de este ejemplo queda como se puede ver en la Figure 4.7. Entonces, un ciclo for se ejecuta N veces y su indice i se va incrementando desde 0 a N-1.

Figure 4.7

Es posible anidar varias sentencias **for**, es decir, introducir sentencias **for** con sus módulos de código en otras sentencias **for**. Quizás, el caso más usual de sentencias anidadas sea el de una sentencia **for** dentro de otra **for**, o si no, dentro de una sentencia **while**. P.e., si se desea que la aplicación anterior se repita 4 veces, entonces lo que se había hecho para el ejemplo aterior es introducido dentro de una nueva sentencia cuyo parámetro N lo fijamos a 4, tal y como se vé en la Figure 4.8.

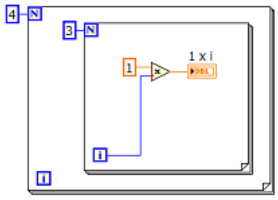

Figure 4.8

4.3.2 While

La sentencia de repetición **while** ya se ha descrito brevemente en la primera parte de este curso. Se trata de una sentencia que permite ejecutar de forma repetida el código que se encuentra definido en su interior mientras se cumpla la condición de permanencia o de ejecución de esta sentencia. La condición de permanencia es un elemento de tipo bit, es decir, de tipo booleano, y para que la sentencia **while** ejecute el código que se encuentra en su interior, el valor del elemento booleano de permanencia ha de ser de **false** (falso). En el momento en que este elemento adquiera el valor **true** (verdadero), la condición de permanencia desaparece y el ciclo **while** deja de ejecutarse. Se dice que se sale del bucle **while**, con lo que deja de ejecutarse el código que se encuantra en su interior.

Normalmente, la condición de permanencia suele ser el resultado de una operación lógica, o bien, un pulsador de tipo **STOP**. Otro elemento que contiene la sentencia **while** es el índice **i**. Este elemento indica el numero de veces que se ha ejecutado el código que se encuentra en el interior de la sentencia.

Un elemento que se suele utilizar mucho a la hora de comprobar el funcionamiento en tiempo de ejecución es el temporizador **Wait (ms)**. Este objeto se encuentra en **Functions/Programming/Timing** y lo que hace es insertar un retardo temporal definido en milisegundos (ms) fijado en su entrada (lado izquierdo). El ejemplo que se muestra en Figure 4.9, muestra la versión del ejercicio anterior, Figure 4.8, pero con una sentencia externa de tipo **while**, donde se ha insertado un retardo un retardo de 1 segundo en la ejecución del ciclo **for** interno.

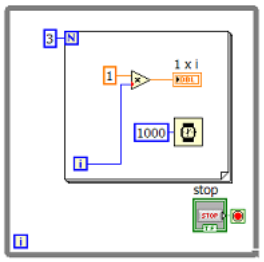

Figure 4.9

4.4 Nodos de realimentación y Registros de desplazamiento

Los nodos de realimentación y registros de desplazamiento son objetos que están relacionados con las sentencias de repetición. Se trata de elementos que se utilizan en módulos de código donde algún o algunos valores pasados son utilizados en el cálculo actual. P.e., si queremos obtener el valor actual de un parámetro donde para obtenerlo hace falta sumar un 2 al valor anterior del mismo, es decir **y=y+2** (**léase y_actual= y_anterior+2**), entonces es necesario utilizar un nodo de realimentación o si no un registro de desplazamiento, tal y como se ve en la Figure 4.10 derecha.

Para introducir un nodo de realimentación en el diagrama Figure 4.10 izquierda, se inserta un objeto **Feedback Node** desde **Functions/Programming/Structure**, cuya salida se conecta con la otra entrada del sumador, y la salida de éste con la entrada del nuevo objeto. A acontinuación, clickando con el botón derecho del ratón sobre el nuevo objeto, se ejecuta **Move Initializer One Loop Out**, con lo que el elemento que permite insertar el valor inicial del valor anterior, se desplaza a la sentencia repetitiva, donde se le asigna un valor inicial, en este caso 3.

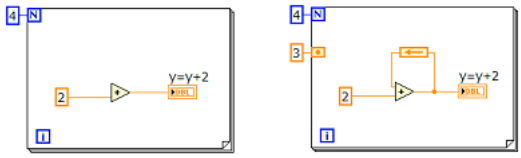

Figure 4.10

Para el caso del registro de desplazamiento, también se parte del diagrama de Figure 4.10 izquierda del caso anterior, solo que ahora nos ubicamos con el ratón en la parte derecha de la sentencia **for**, justo en el límite, y clickando con el botón derecho ejecutamos **Add Shift Register**. Seguidamente aparecen dos flechas, Figure 4.11, donde la de la izquierda permite asignar el valor inicial del valor anterior, igual que en caso anterior, y es conectada a la otra entrada del sumador. En cuanto a la flecha de la derecha, la salida del sumador se conecta a ella.

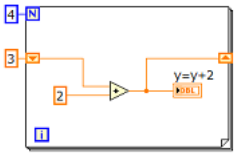

Figure 4.11

Cuando la ecuación a implementar tiene varios valores anteriores, p.e., el valor anterior y su anterior, **y(k-1)** e **y(k-2)**, respectivamente, siendo el valor actual **y(k)**, entonces hay que añadir más elementos a la izquierda de la sentencia **for**, clicando sobre la flecha que se encuentra ahí y ejecutando **Add Element**. Cada nuevo elemento insertado es anterior al anterior y se puede inicializar de forma individual. P.e., el siguiente diagrama, Figure 4.12, representa a la ecuación en diferencias **y(k)=y(k-1)+2+3*y(k-2)**.

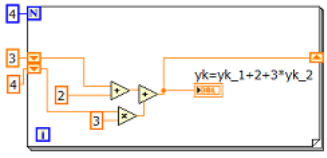

Figure 4.12

4.5 Ejercicios propuestos

Exercise 4.1

Desarrollar un instrumento virtual que proporcione el valor de una de las cuatro operaciones

aritméticas con dos operandos, según el valor del control **Operación** (si es 0, suma, si es 1, resta, si es 2, producto, y si es 3, división). La aplicación dispondrá de un pulsador STOP para que deje de ejecutarse.

Exercise 4.2

Realiza un instrumento virtual que genere las primeras 10 coordenadas y, u ordenadas, de una recta del plano Cartesiano, cuya ecuación es **y=m x + d**, siendo **m** la pendiente positiva, **d** el origen, positivo o negativo, y **x** la ordenada abcisa positiva. Utiliza la sentencia for.

Exercise 4.3

Construye un VI que genere X números aleatorios entre 0 y 7 con la función **Random Number**. X está condicionado al primer número generado fuera del rango pedido.

Exercise 4.4

Construye un VI que genere X números aleatorios entre 4 y 8 con la función **Random Number**. X está condicionado al primer número generado fuera del rango pedido.

Exercise 4.5

Implementar la ecuación en diferencias **y(k)=[4 − 3*y(k-1)+ 53*y(k-2)-34*y(t-3)]/7**, siendo los valores iniciales los siguientes: **y(k-1)in= -2; y(k-2)in= 6.79; y(k-3)in= 12,09**;

Chapter 5

Arrays y Clusters en LabVIEW[1]

5.1 Tipos de datos

LabVIEW soporta principalmente 4 tipos de datos. Los números enteros o **integer**, los números reales de coma flotante (con decimales) o **float**/**double**, los elementos booleanos o de tipo bit, **boolean**, y las cadenas de caracteres o **string**. En el diagrama de cualquier aplicación, los elementos enteros aparecen con el color azul marino, los reales o doubles con color naranja, los booleanos en verde, y las cadenas de caracteres en rosa. Existen más tipos de datos en **LabVIEW**, como los reales de coma fija, etc, pero no los analizaremos dado que quedan fuera del interés de este curso básico.

Dentro de los enteros, podemos hacer una clasificación por su tamaño en bits, habiendo enteros de 8 bits, de 16, de 32 y de 64 bits. Además, es posible definirlos tanto con signo (**signed**, con prefijo `I`), como sin signo (**unsigned**, con prefijo `U`). Así, un entero con signo de 16 bits se define como `I16`. Por defecto, un entero es creado con el tamaño de 32 bits y es de tipo **signed**, `I32`, aunque si se desea cambiar, se pulsa el botón derecho del ratón con el cursor sobre el objeto (una constante, un indicador) y se ejecuta **Properties**, donde a continuación se pulsa sobre **Representation** y se selecciona otro formato, p.e. `U8` y se pulsa **OK**, tal como se refleja en Figure 5.1.

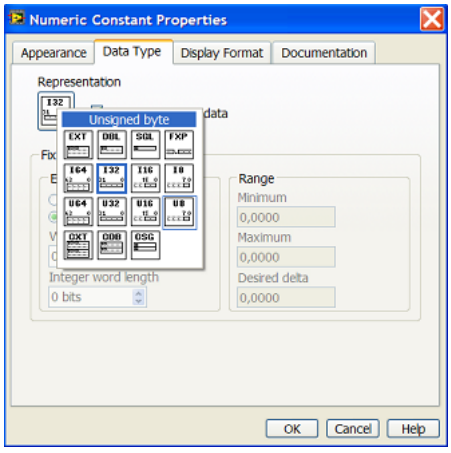

Figure 5.1

[1] This content is available online at <http://cnx.org/content/m18072/1.1/>.

La mayoría de las funciones matemáticas consideran que tanto sus entradas como la salida son de tipo double. El tipo double es el número de coma flotante de doble precisión y se representa con DBL.

Todas las entradas y salidas de las funciones de la paleta **Functions/Programming/Boolean** son de tipo booleano, así como las salidas de las funciones de la paleta **Functions/Programming/Comparison**.

Los datos de tipo cadena se utilizan para enviar y recibir mensajes de texto entre los distintos módulos y funciones de una aplicación. Se utilizan para el envío y recepción de información. Muchas funciones de librería tienen una entrada y también una salida de este tipo. Normalmente la entrada está conectada a la salida de este tipo de la función anterior, y a su vez, la salida está conectada a la entrada de la siguiente función. En este sentido, al final de la secuencia de funciones, se coloca un indicador para ver si se ha producido o no algún error, y para saber de qué tipo es. En el módulo dedicado a la gestión de ficheros se podrá ver mejor su uso.

Cuando interesa convertir un tipo de dato en otro, entonces se utiliza un conversor de tipo. Se trata de un objeto que convierte un tipo de dato en otro. En la paleta **Functions/Programming/Numeric/Conversion**, existen objetos para convertir números a números reales de simple precisión, doble precisión e incluso de coma fija, o también números enteros de distintos tamaños con o sin signo, etc, Figure 5.2derecha. Existe otra paleta, **Functions/Programming/String/String-Number Conversion**, donde hay varias funciones para convertir cadenas de texto o strings en números de distinto formato, y otras que hacen la función inversa, Figure 5.2 izquierda. Finalmente, en la paleta de **Functions/Programming/Array**, y también en la de **Functions/Programming/Cluster**, podemos encontrar conversores de tipo entre array y cluster.

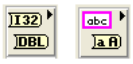

Figure 5.2

5.2 Arrays

Los arrays son conjuntos de datos o elementos del mismo tipo, accesibles mediante los índices del propio array. En **LabVIEW** existen arrays unidimensionales, y también los bidimensionales. El array es en realidad una tabla de dos dimensiones, donde los elementos guardados en sus casillas son accesibles mediante los índices de la tabla, conocidos como filas y columnas.

Hay dos formas de inicializar los arrays. La primera es utilizando las funciones específicas de inicialización de arrays, y la otra, mediante las sentencias repetitivas (**for, while**).

5.2.1 Arrays unidimensionales

Los arrays unidimensionales son aquellos que tienen una única fila y C columnas, es decir son arrays de tamaño 1xC. Para inicializar un array unidimensional utilizando las funciones de arrays, se ejecuta en el diagrama **Functions/Programming/Array/Build Array** , Figure 5.3, y se coloca esa función sobre el diagrama, Figure 5.4 izquierda.

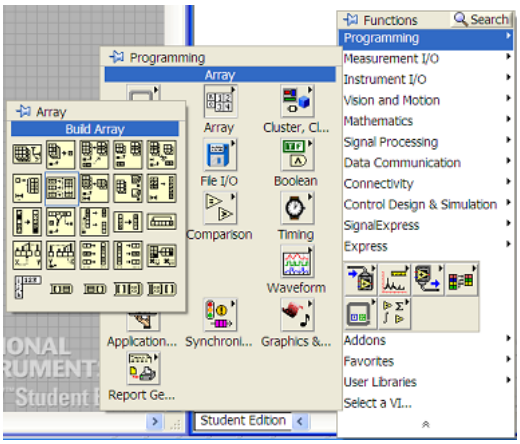

Figure 5.3

A continuación se marca ese objeto con el ratón y se extiende hacia abajo dándole el tamaño de C columnas, p.e. 3 columnas, Figure 5.4 centro. Seguidamente se generan las tres entradas (desde las 3 entradas del objeto **Build Array**), es decir, los controles que proporcionarán los valores de los 3 elementos del array, y también el indicador que muestra el contenido del array (desde la salida del objeto **Build Array**), Figure 5.4 derecha.

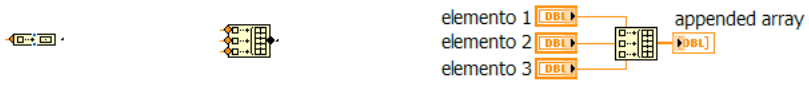

Figure 5.4

A esto, le añadimos una sentencia **while** para que la aplicación se ejecute de forma continua. Al ejecutar la aplicación, se asignan los valores de los elementos del array en los controles correspondientes, Figure 5.5. El array puede ser indexado desde su índice (elemento de control del array), es decir, cada elemento del array se puede ver desde el su índice. Como se puede observar en Figure 5.5 izquierda, el índice comienza en 0, y no en 1.

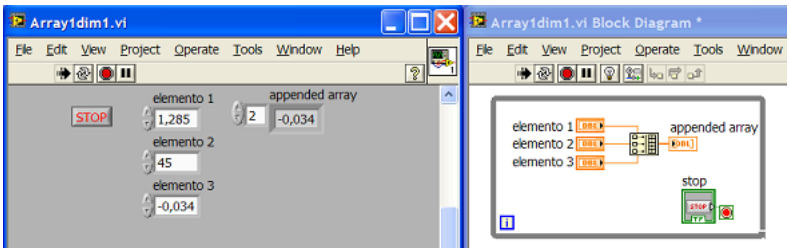

Figure 5.5

Además, si se pretende acceder a un elemento inexistente del array, se muestra un 0, Figure 5.6.

Figure 5.6

Cuando los array son pequeños, o cuando no hay problemas de espacio en el panel, se puede prescindir del uso del índice. Para ello, se expande el array con la herramienta **Position/Size/Select** de la paleta de herrramientas, habiendo seleccionado previamente el elemento visible del array (el primero), Figure 5.7.

Figure 5.7

Otra forma de inicializar los arrays es utilizando una sentencia **for**. Si se emplea una sentencia de tipo **for** que se ejecuta N veces, se crea, a la salida de la misma, un array de 1 fila y N columnas. El valor que se asigna a cada elemento del array es aquél que se obtiene de esa iteración de la sentencia **for**. Este valor puede ser incluso una función del índice de esta sentencia, además de otras cosas.

En el siguiente ejemplo, Figure 5.8, se muestra cómo cada elemento del array proviene de una función que genera números aleatorios (**Functions/Programming/Numeric/Random Number**) entre 0 y 1. El número de elementos creados viene fijado por el valor del número de iteraciones de la sentencia **for**, que en este caso se fija en 3 mediante una constante.

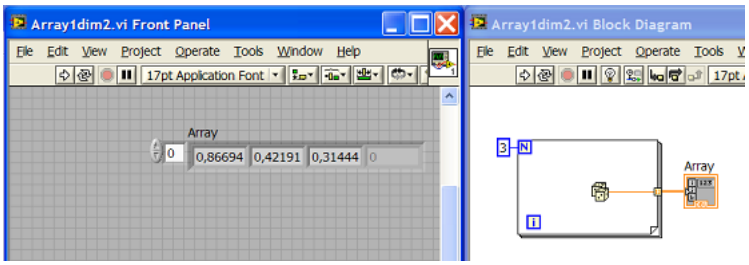

Figure 5.8

Como se puede observar en Figure 5.8 izquierda, el cuarto elemento no existe (es 0), ya que el número de elementos del array está fijado por el número de iteraciones de la sentencia **for**.

En teoría, también se podría utilizar la sentencia de repetición **while** para crear un array unidimensional. Pero esto no suele ser lo habitual, ya que normalmente esta sentencia se ejecuta muchisimas veces el código que se encuentra en su interior, es decir, no suele estar limitada por un número de iteraciones fijo, sino que depende de que se dé una condición dada. Esto significa que hasta que se dé esa condición de parada, ha habido un montón de iteraciones, lo cual se traduce en que se ha generado un array de muchisimos elementos, con el consiguiente consumo de memoria que esto conlleva y problemas que puede generar en el sistema donde se ejecuta la aplicación. Por todo ello, conviene que la generación de arrays con sentencias repetitivas se haga solamente con los de tipo **for**, y nunca o casi nunca con los de tipo **while**.

5.2.2 Arrays bidimensionales

Se trata de arrays de más de una fila, con lo que se consiguen arrays de F filas y C columnas, es decir arrays FxC. Como se puede deducir, un array de dos dimensiones es ideal para guardar o representar tablas de datos, ya que la tabla tiene la misma organización bidimensional de filas y columnas que el array bidimensional.

En cuanto a la inicialización de estos arrays, tenemos las dos formas explicadas que para los arrays unidimensionales. Por una parte, se pueden inicializar utilizando las funciones **Build Array**, y por otra, mediante las sentencias de repetición **for**. Como hemos podido ver con los arrays de una dimensión, una función **Build Array** crea un array de una única fila. Pues bien, para crear un array de dos dimensiones, lo que se hace es unir varios arrays de una dimensión en un array mediante otra función **Build Array**. La única condición que han de cumplir estos arrays unidimensionales, es que todos ellos tengan el mismo número de elementos, es decir, que el mismo número de columnas, Figure 5.9.

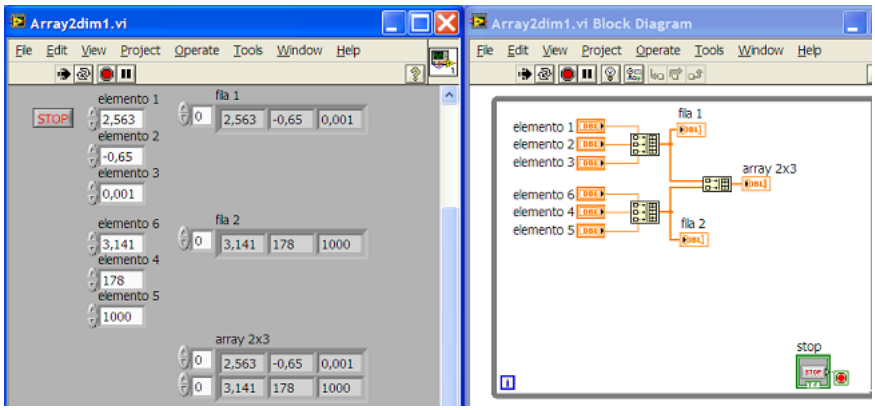

Figure 5.9

A continuación, tenemos un ejemplo donde podemos ver cómo se genera un array de números aleatorios de dos dimensiones utilizando dos sentencias de repetición **for**, una dentro de la otra, Figure 5.10. La sentencia **for** interna, tal y como hemos visto en el caso unidimensional, genera cada una de las filas del array, por lo tanto su número de iteraciones define el número de columnas o elementos de cada fila. Por ello, esto queda fijado mediante un control llamado **Columnas**. Así mismo, la sentencia **for** externa, permite repetir varias veces lo que ocurre en su interior: generar una fila de array. Entonces, fijando su número de iteraciones, se fija el número de filas que tendrá el array resultante, mediante otro control llamado **Filas**.

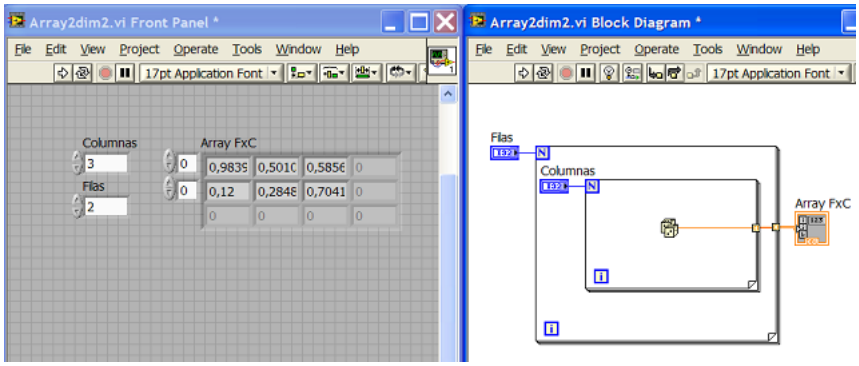

Figure 5.10

Llegados a este punto, conviene notar la diferencia que hay, observando las líneas de los datos, entre un número escalar (salida de la función de número aleatorio), un array unidimensional (salida del **for** interno) y un array bidimensional (salida del **for** externo). Se observa que la línea más delgada corresponde al número escalar. Una línea algo más gruesa representa un array de una dimensión o de una fila, y una línea más gruesa aún, representa un array bidimensional.

5.3 Clusters

Los **clusters** son tipos de datos compuestos por varios elementos de distintos tipos. Son los equivalentes a las estructuras de datos del lenguaje de programación C y similares, o a las clases, en los lenguajes de **Programación Orientados a Objetos**, como el **C++, Java**, etc. En **LabVIEW**, cuando interesa unir varios tipos de datos bajo el mismo nombre, entonces se crea un **cluster** con ese nombre. Esto, suele ser muy práctico para que en el diagrama, aparezcan muchas menos líneas de conexiones y éstas estén agrupadas según un criterio dado. En el ejemplo que viene a continuación, se muestra cómo tres tipos de datos distintos (número real, número entero y cadena de caracteres) que se encuentran en los controles de la izquierda son llevados a la parte derecha del diagrama como una única unidad, Figure 5.12 derecha. Para aguparlos, se utiliza el objeto **Bundle**, Figure 5.11 izda., que se encuentra en **Functions/Programming/Cluster, Class & Variants**. En cambio, en el lado derecho, tanto del diagrama como del panel, se encuentran los indicadores que permiten visualizar los valores enviados desde los controles antes mencionados. Para ello, se utiliza el objeto **Unbundle**, Figure 5.11 dcha., cuya misión es obtener los elementos individuales del grupo que llega a su entrada.

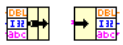

Figure 5.11

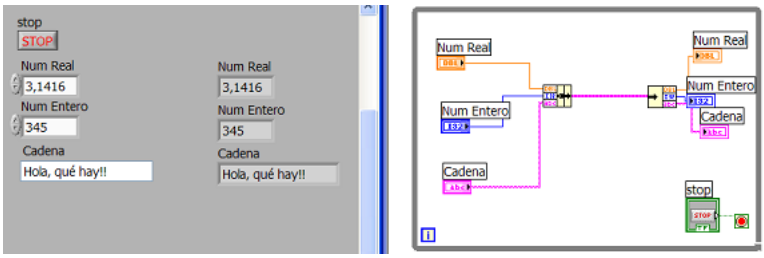

Figure 5.12

Otra interesante aplicación de los **clusters** es la de crear aplicaciones con menús de botones y máquinas de estados. Para crear un menú de botones se inserta un objeto **cluster** desde el panel, desde **Controls/Classic/Cluster/Classic Array, Matriz & Cluster**. A continuación se insertan dos botones de tipo **OK** a los que se les elimina la etiqueta externa, en el **cluster**. Se edita cada uno de ellos con el botón derecho del ratón y ejecutando **Properties** y en la ficha **Appearance**, en **Off text**, se elimina **OK** y se escribe "+" en un caso y "-" en el otro.

En el diagrama, se introduce una sentencia **while** para controlar la ejecución de la aplicación, donde el **cluster** anterior es introducido en ella, Figure 5.13. A la salida del **cluster** se conecta un conversor de tipo **Cluster to Array**, que se encuentra en **Functions/Programming/Array**. A su vez, la salida de este conversor es llevado a la entrada superior del **Search 1D Array**, donde su entrada inferior está conectada a una constante booleana de valor **true**. Asi, este objeto proporciona el índice del array cuyo elemento ha proporcionado un **true**, es decir, en este caso, cuyo botón ha sido pulsado. De este modo, si se pulsa el

primer botón, la función **Search 1D Array** devuelve un 0, si se pulsa el segundo, devuelve un 1, y así si hubiera más botones. En cambio, si no se pulsa ningún botón, la función devuelve –1.

En este sentido, si a la salida de esta función conectamos una sentencia de selección múltiple, entonces se puede colocar en cada sentencia el código asociado al botón o función que se quiera realizar. Así, tendriamos una sentencia o estado de reposo para cuando no se pulsa ningún pulsador (-1), otro para la suma de los dos operandos (0), y finalmente para le resta de los operandos (1). Además del código asociado a cada estado o función, se colocará una constante de tipo cadena o string, que irá conectado a la salida de la sentencia para que en el panel se pueda ver el estado en el que se encuentra la aplicación en todo momento. Dendro de la sentencia **while**, se añadirá un temporizador de 1 segundo para que se puedan ver los resultados en tiempo de ejecución de la aplicación.

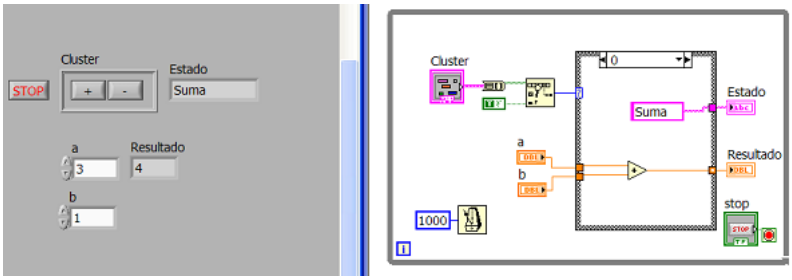

Figure 5.13

La aplicación anterior vuelve automáticamente al estado de reposo. Otra forma de hacer la misma aplicación es introduciendo el código del cluster y la **Search 1D Array** en la sentencia asociada al reposo, pero añadiendo también un registro de desplazamiento a la sentencia **while**, Figure 5.14. Así, tal y como se puede ver, el estado inicial es el de reposo (-1), y al pulsar alguno de los dos pulsadores, la salida de la **Search 1D Array** proporciona el código asociado y vía registro de desplazamiento vuelve por la entrada de la sentencia múltiple para ejecutar el correspondiente caso. Pero cada uno de estos casos lleva el valor del código del estado que se quiere ejecutar tras su ejecución, que en nuestro caso es –1, el de reposo (ver los dos casos de Figure 5.15).

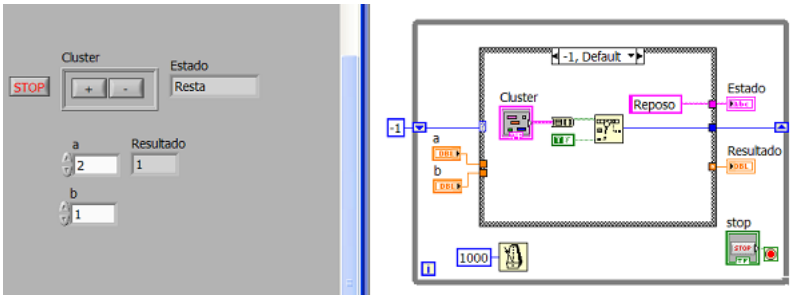

Figure 5.14

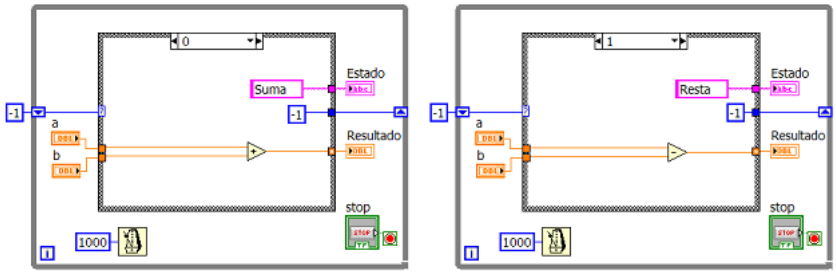

Figure 5.15

Esto muestra claramente que estos códigos no tienen por qué ser el de reposo, si no otros que el programador desee. De esta forma se pueden definir ciclos de máquinas de estados, aunque **LabVIEW** ya posee objetos específicos para realizar máquinas de estados.

5.4 Ejercicios propuestos

Exercise 5.1
Construye un instrumento virtual que genere un array unidimensional con los valores de dos ciclos completos de la función trigonométrica seno.

Exercise 5.2
Construye un instrumento virtual que genere un array bidimensional con los valores de dos ciclos completos de las funciones trigonométricas seno (1ª fila), coseno (2ª fila), tangente (3ª fila) y cotangente (4ª fila).

Exercise 5.3
Construye un instrumento virtual que genere un array bidimensional con los valores de 4 fechas en el formato: día mes año (p.e 23 de septiembre de 2008: 23 09 2008).

Exercise 5.4
Construye un VI con un menú de botones que realiza exactamente las mismas operaciones que la aplicación **Calculadora2.vi**, es decir, con los dos operandos, las operaciones artméticas +, -, * y división. Con el operando 1 tomando como el ángulo en grados, las operaciones trigonométricas sin, cos, tan y cotan, y con el operando 2, la raíz cuadrada y la inversa. Todos los resultados se muestran en el mismo indicador.

Exercise 5.5
Realiza un VI que implementa la máquina de estados de Figure 5.16. Los estados son los círculos, y las letras, las acciones para pasar de un estado a otro (pulsadores).

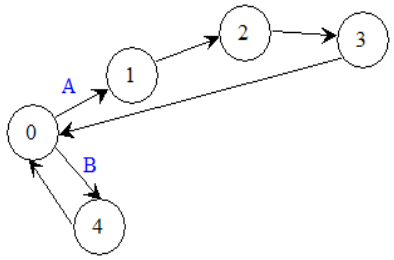

Figure 5.16

Exercise 5.6
Realiza un VI con un menú de botones que realiza varias operaciones con dos arrays cuadrados de 3x3, A y B. Elige el modo de iniciaización más cómodo para los arrays. Las operaciones son: para los dos arrays: +, -, * , para el array A: determinante, inversa, transpuesta.

Index of Keywords and Terms

Keywords are listed by the section with that keyword (page numbers are in parentheses). Keywords do not necessarily appear in the text of the page. They are merely associated with that section. *Ex.* apples, § 1.1 (1) **Terms** are referenced by the page they appear on. *Ex.* apples, 1

Attributions

Collection: *Primeros pasos con LabVIEW*
Edited by: Patxi Alkorta Egiguren
URL: http://cnx.org/content/col10592/1.2/
License: http://creativecommons.org/licenses/by/2.0/

Module: "Instalación de LabVIEW"
By: Patxi Alkorta Egiguren
URL: http://cnx.org/content/m18064/1.2/
Pages: 1-9
Copyright: Patxi Alkorta Egiguren
License: http://creativecommons.org/licenses/by/2.0/

Module: "Introducción a LabVIEW"
By: Patxi Alkorta Egiguren
URL: http://cnx.org/content/m18065/1.1/
Pages: 11-24
Copyright: Patxi Alkorta Egiguren
License: http://creativecommons.org/licenses/by/2.0/

Module: "Programación modular con LabVIEW"
By: Patxi Alkorta Egiguren
URL: http://cnx.org/content/m18068/1.1/
Pages: 25-31
Copyright: Patxi Alkorta Egiguren
License: http://creativecommons.org/licenses/by/2.0/

Module: "Sentencias en LabVIEW"
By: Patxi Alkorta Egiguren
URL: http://cnx.org/content/m18070/1.1/
Pages: 33-39
Copyright: Patxi Alkorta Egiguren
License: http://creativecommons.org/licenses/by/2.0/

Module: "Arrays y Clusters en LabVIEW"
By: Patxi Alkorta Egiguren
URL: http://cnx.org/content/m18072/1.1/
Pages: 41-50
Copyright: Patxi Alkorta Egiguren
License: http://creativecommons.org/licenses/by/2.0/

Primeros pasos con LabVIEW
Este curso básico sirve para hacer una introducción y adquirir los fundamentos de la programación gráfica con el software de desarrollo de propósito general LabVIEW. En el primer módulo o apartado, se explica el proceso de instalación de LabVIEW en un PC. El segundo, muestra la base de la programación gráfica, para así en el tercero explicar el aspecto de la programación modular. El cuarto módulo describe las sentencias de programación más útiles de esta herramienta, y el quinto, hace un estudio de los arrays y clusters de LabVIEW. Todos los módulos son explicados con ejemplos sencillos y conceptuales, y al final de cada uno, se plantean nuevos ejercicios de dificultad creciente de forma gradual para que el/la lector/a pueda poner en práctica los conocimientos adquiridos en cada módulo.