Técnicas y modelos de diseño comunes

Lo primero para desarrollar un proyecto en LabVIEW es explorar las arquitecturas que existen en LabVIEW. Los arquitecturas son esenciales para crear un buen diseño software. Las arquitecturas más comunes se suelen agrupar en modelos de diseño.

A medida que un modelo de diseño gana aceptación, resulta más fácil reconocer cuándo se ha utilizado uno. Este reconocimiento le ayuda a usted y a otros desarrolladores a leer y modificar VIs que se basan en modelos de diseño.

Existen numerosos modelos de diseño para VIs de LabVIEW. La mayoría de las aplicaciones usan al menos uno. En este curso explorará el modelo de diseño de máquina de estados. Aprenda más sobre modelos de diseño en *LabVIEW Básico II*.

Temas

A. Programación secuencial

B. Programación de estado

C. Máquinas de estados

D. Paralelismo

A. Programación secuencial

En la lección 1, *Resolución de problemas*, diseñó un diagrama de flujo para una estación meteorológica de temperatura. El VI Temperature Weather Station completa los siguientes pasos:

1. Leer la temperatura
2. Comprobar los límites y advertencias de visualización de la temperatura
3. Representar y registrar la temperatura

Tras llegar al final de la secuencia de pasos, se comprueba que se haya hecho clic en el botón de parada, de lo contrario la secuencia se repite.

Muchos de los VIs que escribe en LabVIEW cumplen tareas secuenciales. El modo de programar estas tareas secuenciales puede ser muy distinto. Observe el diagrama de bloques de la figura 10-1. En este diagrama de bloques, se adquiere una señal de tensión, un cuadro de diálogo indica al usuario que active la alimentación, la señal de tensión se adquiere de nuevo y al usuario se le indica que debe apagar la alimentación. No obstante, en este ejemplo no hay nada en el diagrama de bloques que fuerce el orden de ejecución de estos eventos. Cualquiera de estos eventos podría suceder el primero.

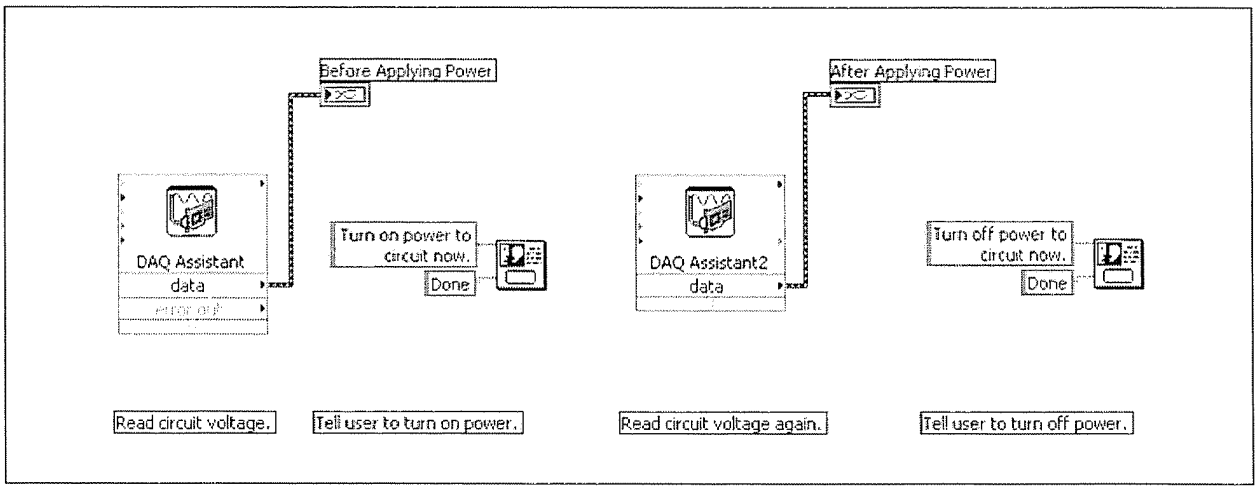


Figura 10-1. Tareas sin secuenciar

En LabVIEW puede completar tareas secuenciales colocando cada una en un subVI y ordenando los subVIs en el orden en que desee que se ejecuten usando los cables del cluster de error. Sin embargo, en este ejemplo sólo dos de las tareas tienen un cluster de error. Con los clusters de error puede forzar el orden de ejecución de los dos DAQ Assistants, pero no las funciones One Button Dialog, como en la figura 10-2.

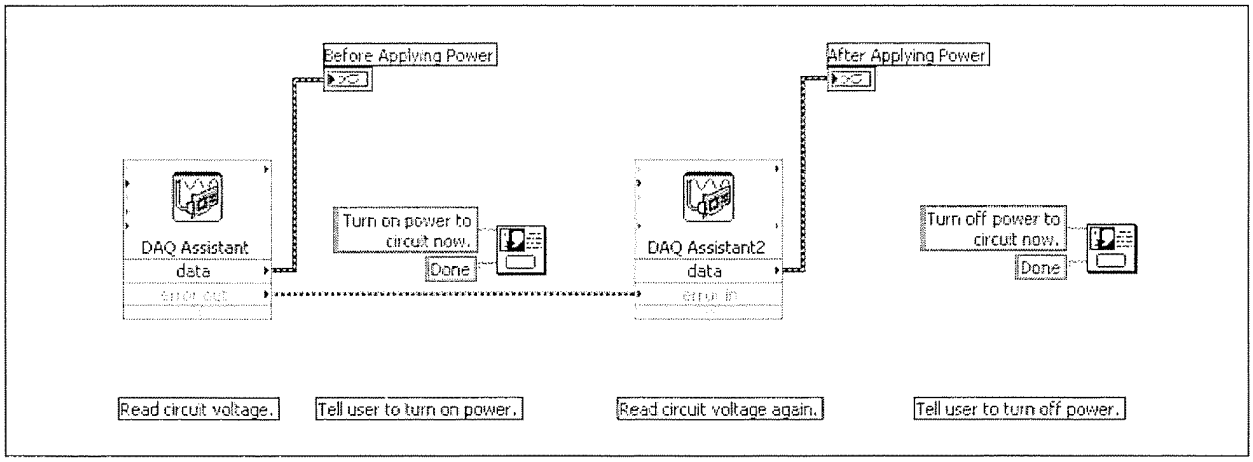


Figura 10-2. Tareas secuenciadas parcialmente

Puede usar una estructura Sequence para forzar el orden de operaciones de los objetos del diagrama de bloques. Una estructura Sequence contiene uno o más subdiagramas, o marcos, que se ejecutan en orden secuencial; un marco no puede empezar la ejecución hasta que se haya completado la ejecución de todo el código del marco anterior. La figura 10-3 muestra un ejemplo de este VI que usa una estructura Sequence para forzar el orden de ejecución.

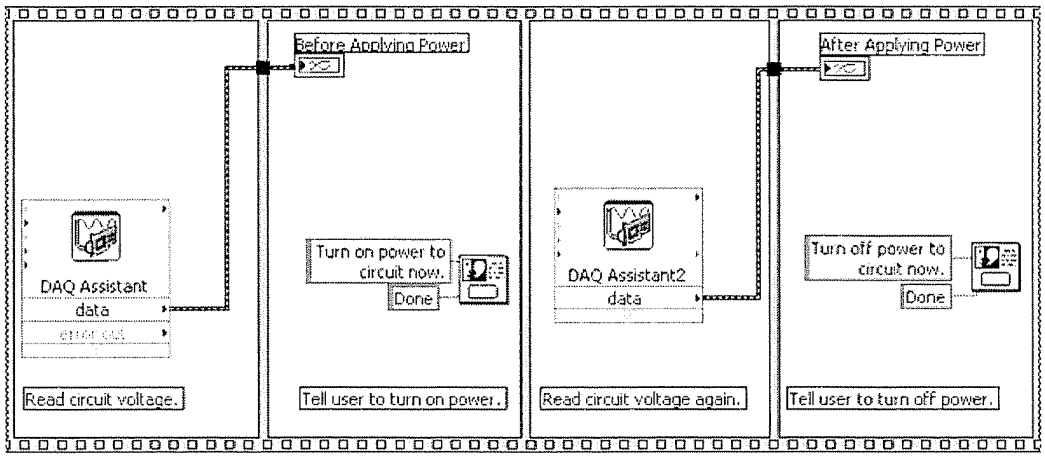


Figura 10-3. Tareas secuenciadas con una estructura Sequence

Para aprovechar el paralelismo inherente de LabVIEW, evite abusar del uso de estructuras Sequence. La estructuras Sequence garantizan el orden de ejecución, pero prohíben operaciones paralelas. Otra desventaja de usar estructuras Sequence es que no puede detenerse la ejecución en medio de la secuencia. Un buen modo de usar las estructuras Sequence para este ejemplo aparece en la figura 10-4.

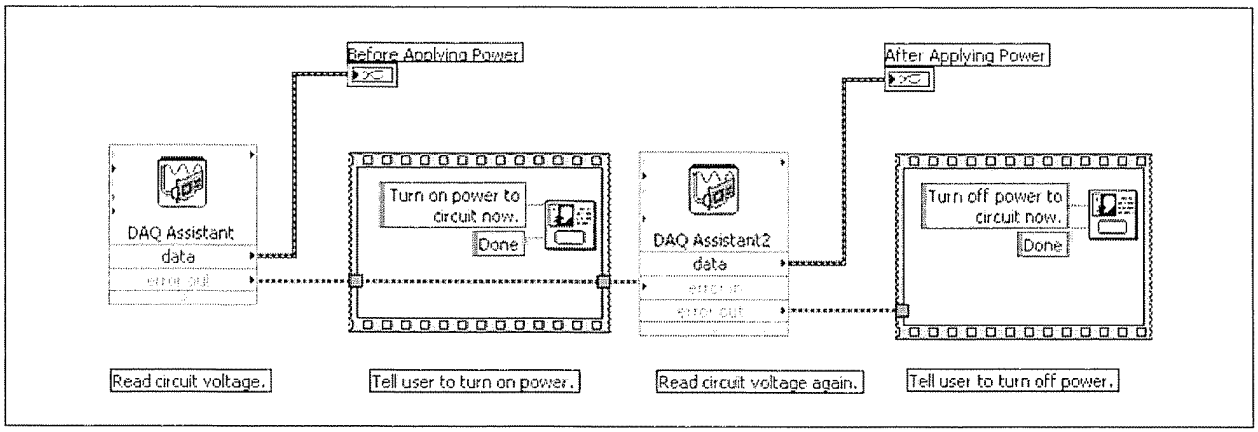


Figura 10-4. Tareas secuenciadas con estructuras Sequence y un cluster de error

Sin embargo, el mejor modo de escribir este VI es encerrar las funciones One Button Dialog en una estructura Case y cablear el cluster de error al selector de caso.

Use con moderación las estructuras Sequence porque no realizan la comprobación de errores y la ejecución de la secuencia continúa incluso aunque se detecten errores. Base su desarrollo en el flujo de datos más que en las estructuras Sequence para controlar el orden de ejecución.

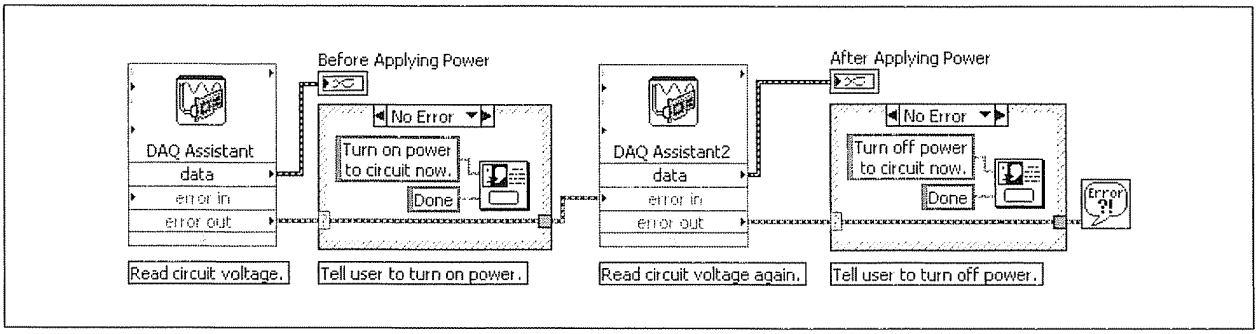


Figura 10-5. Tareas secuenciadas con un cluster de error y estructuras Case

B. Programación de estado

Aunque una estructura Sequence y subVIs cableados secuencialmente cumplen la tarea, los VIs suelen requerir programación más compleja:

- ¿Qué ocurre si debe cambiar el orden de la secuencia?
- ¿Qué ocurre si debe repetir un elemento de la secuencia más a menudo que los otros?
- ¿Qué ocurre si algunos elementos de la secuencia se ejecutan sólo ante ciertas condiciones?
- ¿Qué ocurre si debe detener el programa inmediatamente, en lugar de esperar hasta el final de la secuencia?

Aunque su programa no tenga ninguno de los requisitos anteriores, siempre existe la posibilidad de que el programa sea modificado en el futuro. Por este motivo, una arquitectura de programación de estado es una buena elección, aunque bastaría con una estructura de programación secuencial.

C. Máquinas de estados

El modelo de diseño de la máquina de estados es común y muy útil en LabVIEW. Puede usar el modelo de diseño de máquina de estados para implementar cualquier algoritmo que pueda describir explícitamente un diagrama de estado o un diagrama de flujo. Una máquina de estado normalmente implementa un algoritmo moderadamente complejo de toma de decisiones, como una rutina de diagnóstico o una monitorización de proceso.

Una *máquina de estados*, que se define mejor como una máquina de estados finitos, consta de una serie de estados y una función de transición que indica cuál es el siguiente estado. Las máquinas de estados tienen numerosas variaciones. Las dos máquinas de estados finitos más comunes son Mealy y Moore. Una máquina Mealy realiza una acción para cada transición. Una máquina Moore realiza una acción concreta para cada estado del diagrama de transición de estados. La plantilla del modelo de diseño de la máquina de estados en LabVIEW implementa un algoritmo que describe una máquina Moore.

Aplicación de máquinas de estados

Use máquinas de estados en aplicaciones donde se distingan los estados. Cada estado puede llevar a uno o más estados o finalizar el flujo del proceso. Una máquina de estados depende de la entrada del usuario o del resultado del estado actual para determinar qué estado irá después. Muchas aplicaciones requieren un estado de inicialización seguido de un estado predeterminado, donde pueden realizarse numerosas acciones. Las acciones realizadas pueden depender de entradas y estados anteriores y actuales. Un estado de cierre normalmente realiza acciones de limpieza.

Las máquinas de estados se suelen utilizar para crear interfaces de usuario. En una interfaz de usuario, varias acciones del usuario envían la interfaz de usuario a varios segmentos de procesamiento. Cada segmento de procesamiento actúa como un estado en la máquina de estados. Cada segmento puede llevar a otro segmento para el procesamiento o esperar a otra acción por parte del usuario. En una aplicación de interfaz de usuario, la máquina de estados monitoriza constantemente las acciones del usuario para conocer la siguiente acción a realizar.

La prueba de procesos es otra aplicación común del modelo de diseño de la máquina de estados. En una prueba de proceso, un estado representa cada segmento del proceso. En función del resultado de cada prueba del proceso, podría llamarse a otro estado. Esto puede suceder continuamente, lo que produce un análisis profundo del proceso que está probando.

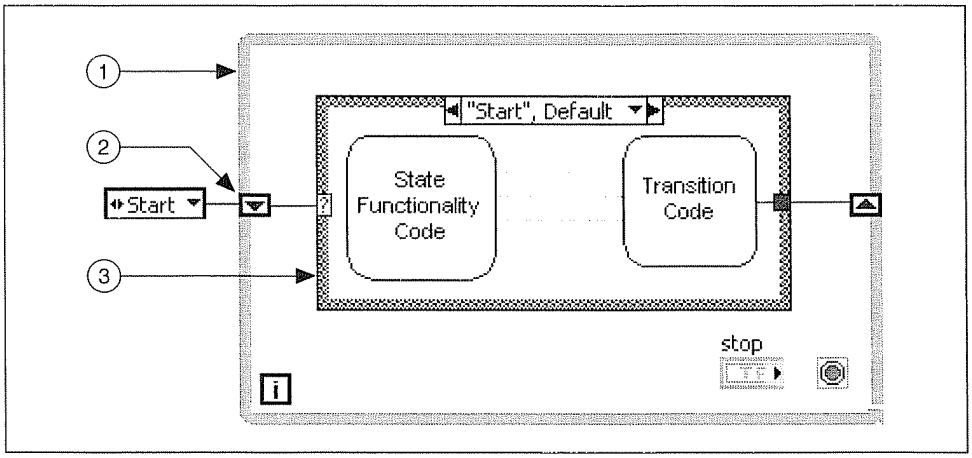
La ventaja de usar una máquina de estados es que puede crear VIs en LabVIEW fácilmente una vez creado un diagrama de transición de estados.

Infraestructura de la máquina de estados

Trasladar el diagrama de transición de estados a un diagrama de bloques de LabVIEW requiere los siguientes componentes de infraestructura:

- **Bucle While**: ejecuta continuamente los estados.
- **Estructura Case**: contiene un caso para cada estado y el código para ejecutar en cada estado.
- **Registro de desplazamiento**: contiene la información de transición de estados.
- **Código de funcionalidad de estado**: implementa la función del estado.
- **Código de transición**: determina el siguiente estado de la secuencia.

La figura 10-6 muestra la estructura básica de una máquina de estados implementada en LabVIEW para un sistema de adquisición de datos de temperatura.



1 Bucle While
2 Registro de desplazamiento
3 Estructura Case

Figura 10-6. Infraestructura básica de una máquina de estados de LabVIEW

El bucle While implementa el flujo del diagrama de transición de estados. Cada estado se representa con casos en la estructura Case. Un registro de desplazamiento en el bucle While realiza el seguimiento del estado actual y comunica el estado actual con la entrada de la estructura Case.

Control de máquinas de estados

El mejor método para controlar la inicialización y transición de máquinas de estados es el control de tipo enumerado. Los enums se utilizan ampliamente como selectores de caso en máquinas de estados. Sin embargo, si el usuario intenta añadir o eliminar un estado del control de tipo enumerado, se rompen los cables restantes conectados a las copias de este control de tipo enumerado. Es uno de los obstáculos más comunes al implementar máquinas de estados con controles de tipo enumerado. Una solución a este problema es crear un control de tipo enumerado con tipo definido. Así, todas las copias de control de tipo enumerado se actualizarán automáticamente si añade o elimina un estado.

Transición de máquinas de estados

Existen numerosos modos de controlar qué caso ejecuta una estructura Case en una máquina de estados. Elija el método que se adapte mejor a la función y complejidad de su máquina de estados. De los métodos para implementar transiciones en máquinas de estados, el más común y fácil de usar es el código de transición de estructura Case individual, que puede utilizarse como transición entre cualquier número de estados. Este método permite una arquitectura de máquina de estados más escalable, legible y mantenible. Los otros métodos pueden resultar útiles en situaciones concretas, y es importante que se familiarice con ellos.

Transición predeterminada

Para la transición predeterminada, no se requiere código para determinar el siguiente estado, porque sólo hay un estado posible que ocurrirá a continuación. La figura 10-7 muestra un modelo de diseño que usa una transición predeterminada implementada para un sistema de adquisición de datos de temperatura.

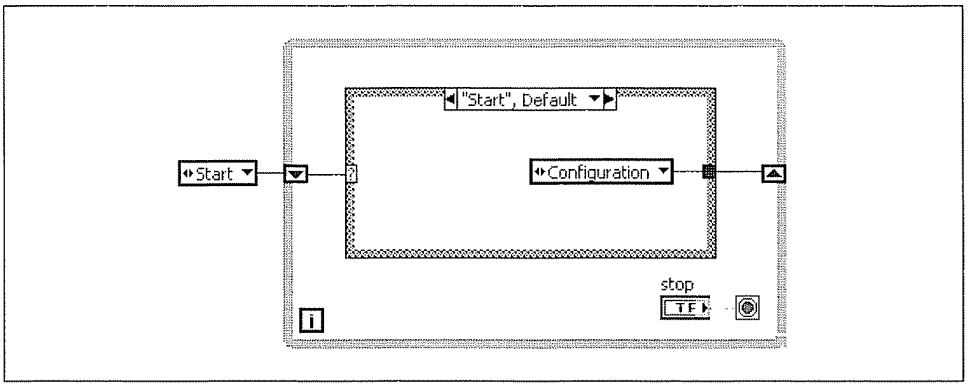


Figura 10-7. Transición predeterminada individual

Transición entre dos estados

El siguiente método implica tomar una decisión en una transición entre dos estados. Para ello se suelen utilizar varios modelos. La figura 10-8 muestra la función Select utilizada para la transición entre dos estados.

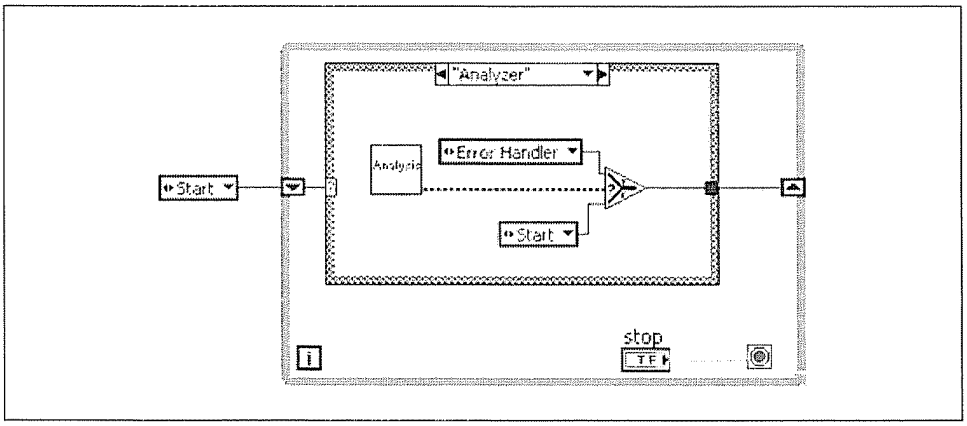


Figura 10-8. Código de transición de la función Select

Este método funciona bien si sabe que cada estado siempre transiciona entre dos estados. Sin embargo, este método limita la escalabilidad de la aplicación. Si necesitara modificar el estado para la transición entre más de dos estados, esta solución no funcionaría y requeriría una modificación considerable del código de transición.

Transición entre dos o más estados

Cree una arquitectura más escalable utilizando uno de los siguientes métodos para la transición entre estados.

- **Estructura Case**: use una estructura Case en lugar de la función Select para el código de transición.

  La figura 10-9 muestra la transición de la estructura Case implementada para un sistema de adquisición de datos de temperatura.

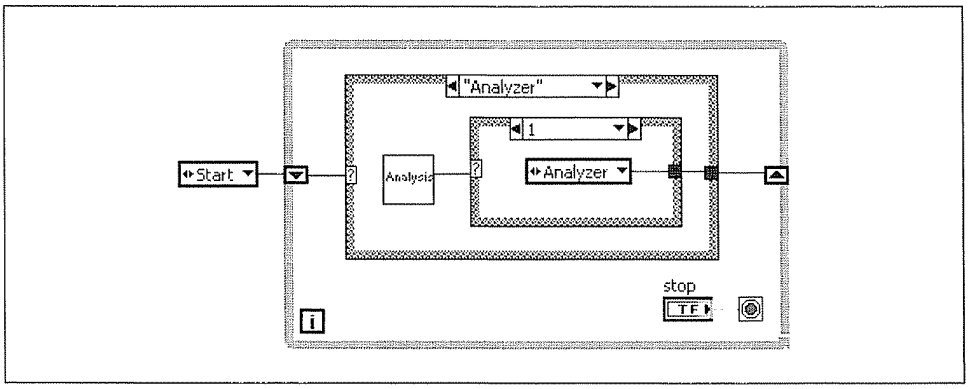


Figura 10-9. Código de transición de la estructura Case

Una ventaja de usar una estructura Case es que el código se autodocumenta. Como cada caso de la estructura Case corresponde a un elemento del enum, resulta fácil leer y comprender el código. Una estructura Case también es escalable. A medida que crece la aplicación, puede añadir más transiciones a un estado concreto añadiendo más casos a la estructura Case para ese estado. Una desventaja de usar una estructura Case es que no se ve todo el código a la vez. Debido a la naturaleza de la estructura Case, no se puede ver de una vez toda la funcionalidad del código de transición.

- **Array de transición**: si tiene que ver más código del que permite la estructura Case, puede crear un array de transición para todas las transiciones que tengan lugar.

  La figura 10-10 muestra el array de transición implementado para un sistema de adquisición de datos de temperatura.

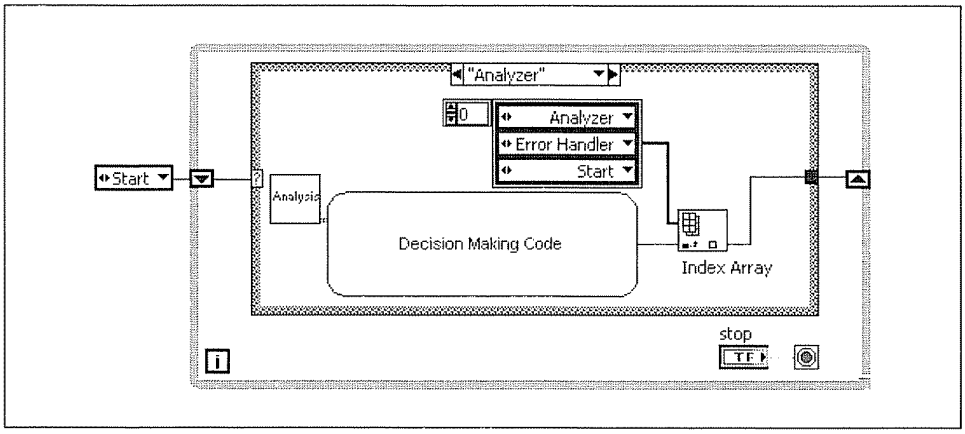


Figura 10-10. Código de transición del Array de transición

En este ejemplo, el código de toma de decisión facilita el índice que describe el siguiente estado. Por ejemplo, si el código debe avanzar al siguiente estado Error Handler, el código de toma de decisión genera el número 1 para la entrada index de la función Index Array. Este modelo de diseño hace el código de transición escalable y fácil de leer. Una desventaja de este modelo es que debe tener cuidado al desarrollar el código de transición porque el array tiene un índice cero.

Estudio de un caso: proyecto del curso

El proyecto del curso adquiere una temperatura cada medio segundo, analiza cada temperatura para determinar si es demasiado alta o baja y alerta al usuario si hay peligro de golpe de calor o congelación. El programa registra los datos si hay una advertencia. Si el usuario no ha hecho clic en el botón de parada, se repetirá todo el proceso. La figura 10-11 muestra el diagrama de transición de estados para el proyecto del curso.

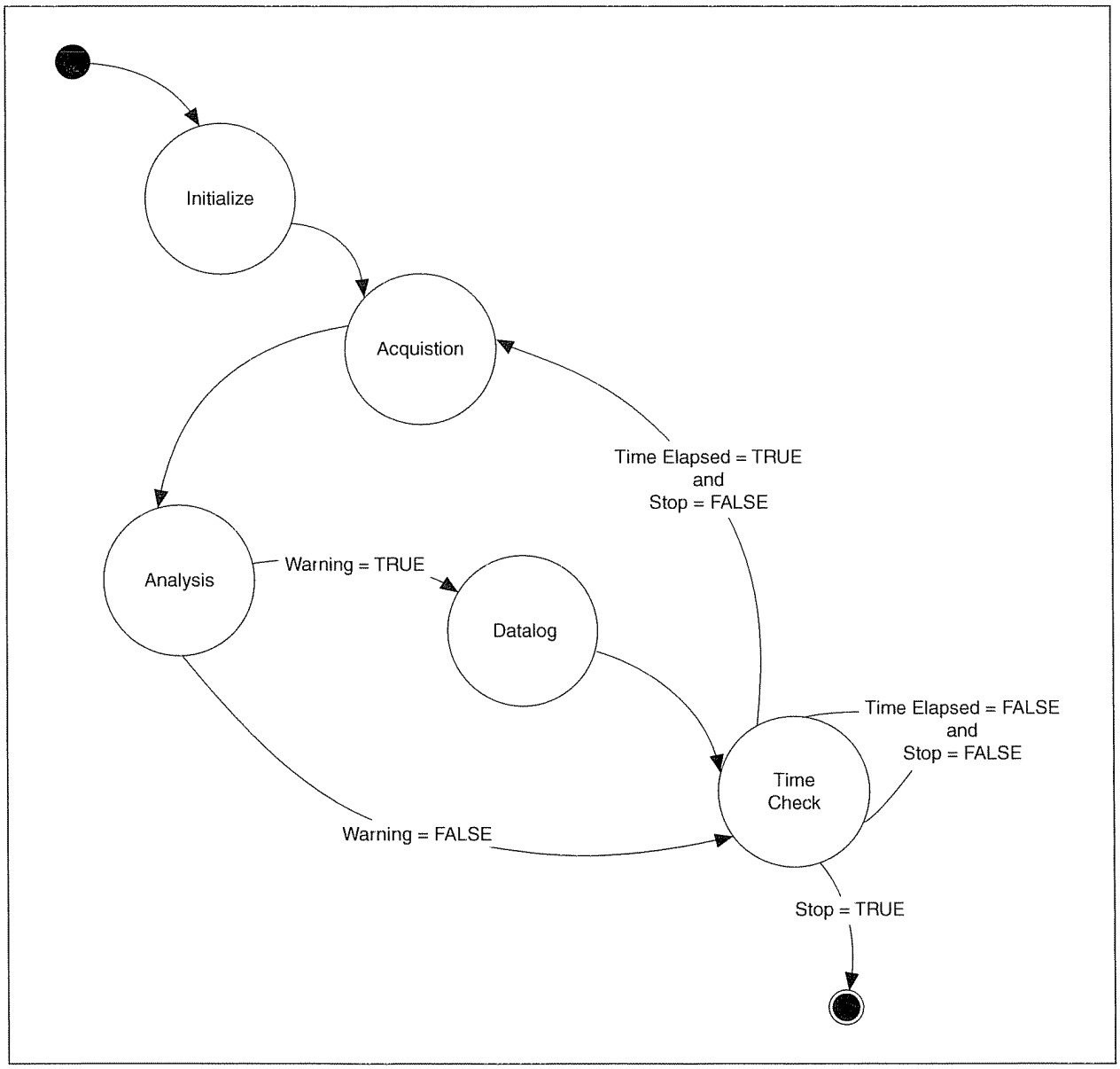


Figura 10-11. Diagrama de transición de estados para el proyecto del curso

Las figuras 10-12 a la 10-15 ilustran los estados de la máquina de estados que implementa el diagrama de transición de estados detallado en la figura 10-11. Si ha instalado los ejercicios y las soluciones, el proyecto se encuentra en el directorio `<Exercises>\LabVIEW Basics I\Course Project`. Podrá explorar más esta máquina de estados.

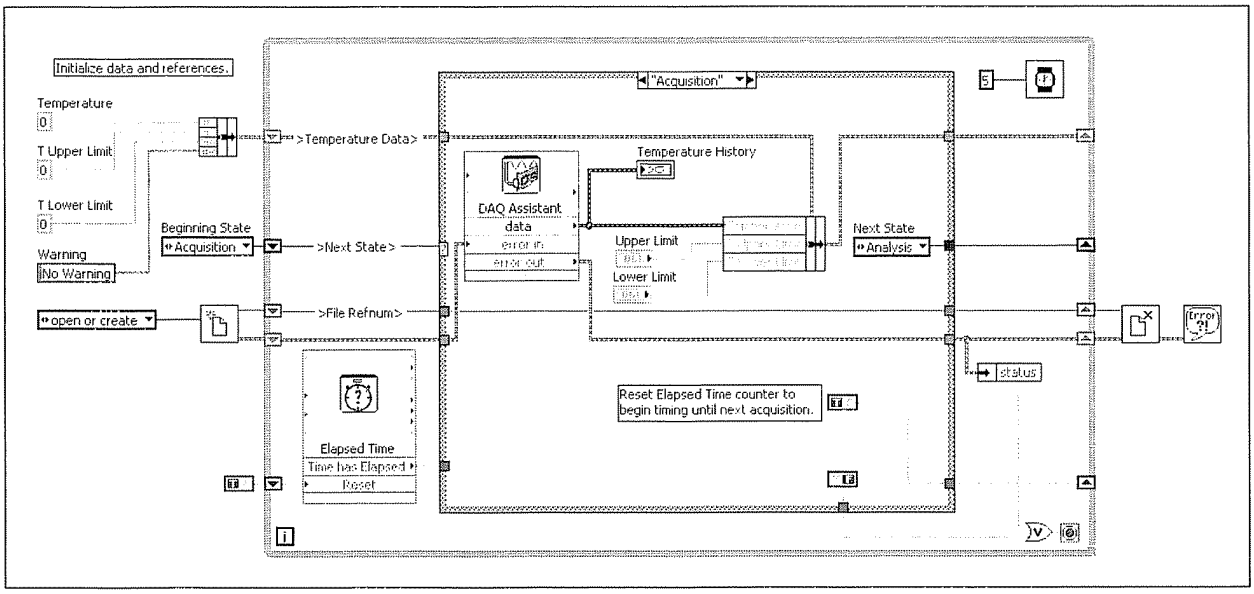


Figura 10-12. Estado de adquisición

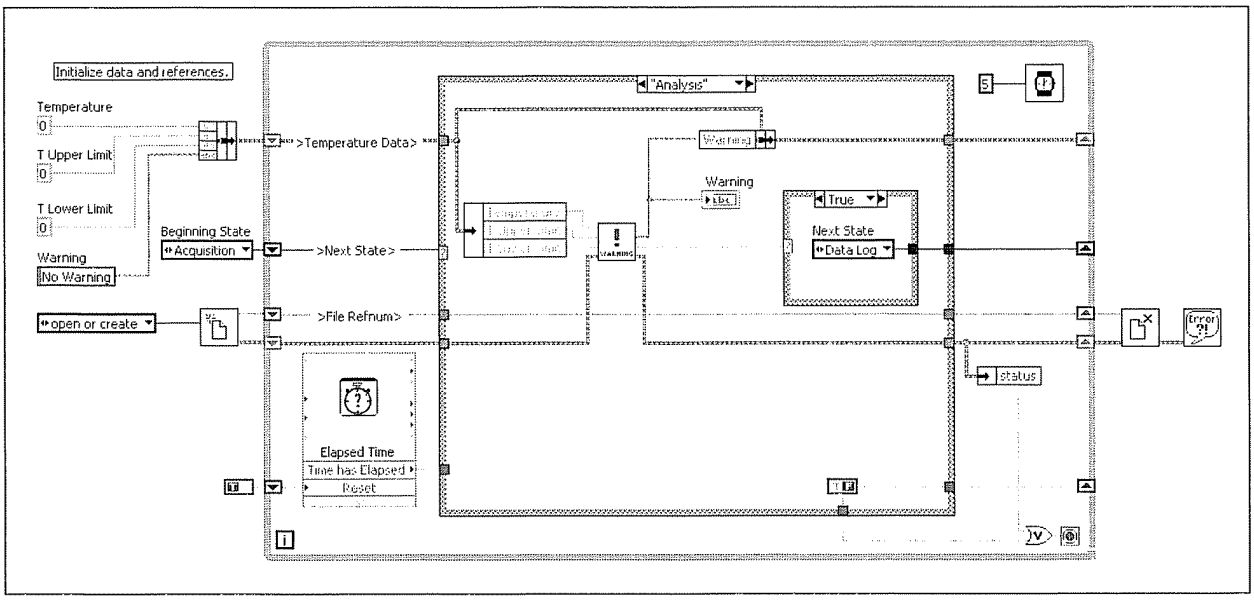


Figura 10-13. Estado de análisis

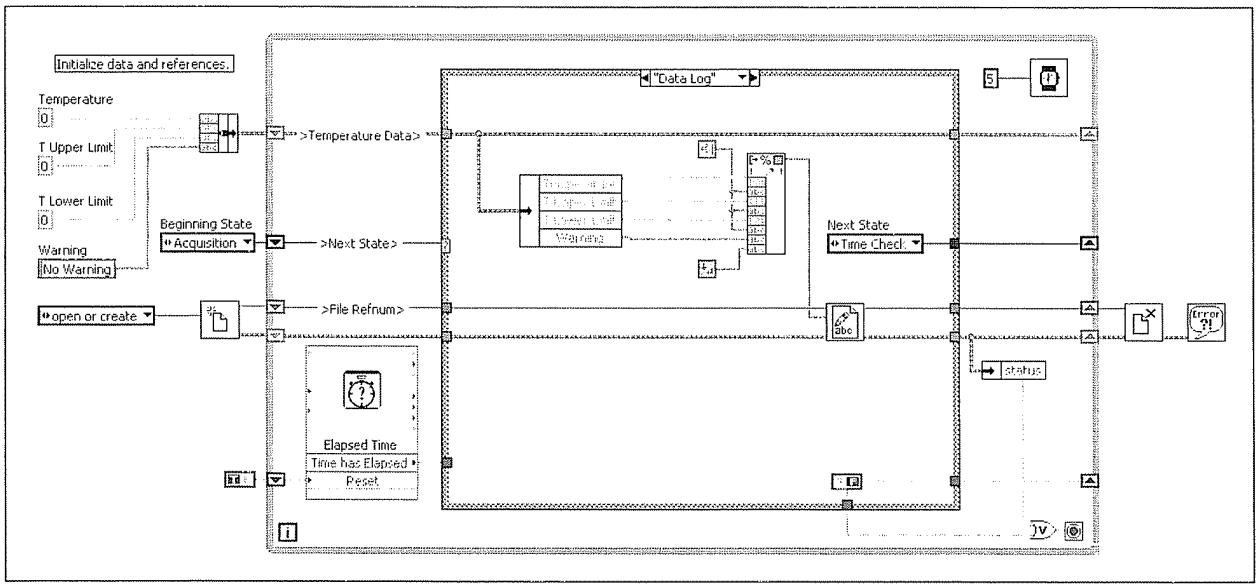


Figura 10-14. Estado de registro de datos

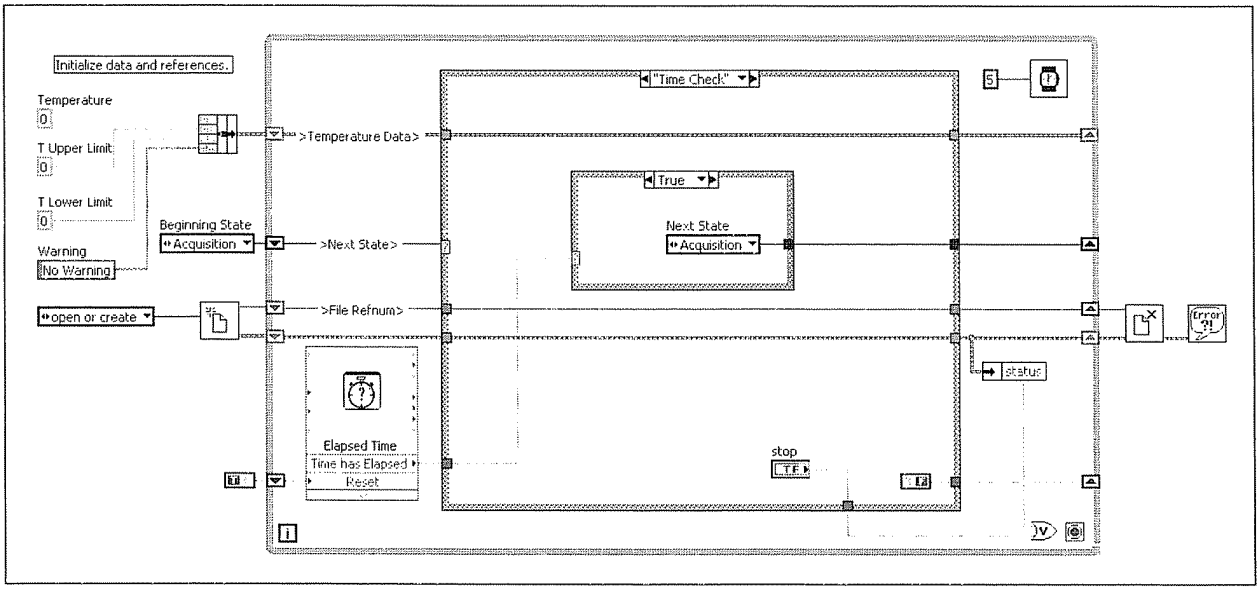


Figura 10-15. Estado de comprobación del tiempo

Ejercicio 10-1 VI State Machine

Objetivo

Crear un VI que implemente una máquina de estados usando una definición de tipo enum.

Escenario

Debe diseñar una plantilla para una máquina de estados de interfaz de usuario. La máquina de estados debe permitir al usuario activar el proceso 1 o el proceso 2 en cualquier orden. La máquina de estados también debe permitir la expansión, ya que pueden añadirse procesos en el futuro.

Diseño

Entradas y salidas

Tabla 10-1. Entradas y salidas

Tipo	Nombre	Propiedades
Cancel Button	`Process 1`	Texto booleano: `Process 1`
Cancel Button	`Process 2`	Texto booleano: `Process 2`
Stop Button	`Stop`	—

Transiciones de estado

Tabla 10-2. Entradas y salidas

Estado	Acción	Siguiente estado
Inactivo	Comprobar la selección en el interfaz de usuario	No se hace clic en ningún botón: `Estado Idle` Se hace clic en el proceso 1: `Estado Process 1` Se hace clic en el proceso 2: `Estado Process 2` Se hace clic en Stop: `Estado Stop`
Proceso 1	Ejecutar el código del proceso 1	Estado Idle
Proceso 2	Ejecutar el código del proceso 2	Estado Idle
Detener	Detener la máquina de estados	Estado Stop

Implementación

En los siguientes pasos, creará la ventana del panel frontal de la figura 10-16.

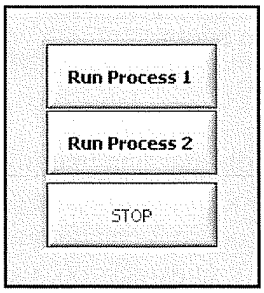


Figura 10-16. Ventana del panel frontal del VI State Machine

1. Cree un nuevo proyecto que contenga un VI en blanco.
   - ❑ Seleccione **Empty Project** en la ventana **Getting Started**.
   - ❑ Seleccione **File»Save Project**.
   - ❑ Llame al proyecto `State Machine.lvproj` en el directorio `<Exercises>\LabVIEW Basics I\State Machine`.
   - ❑ Seleccione **File»New VI**.
   - ❑ Guarde el nuevo VI como `State Machine.vi` en el directorio `<Exercises>\LabVIEW Basics I\State Machine`.
2. Cree un cluster de menú que contenga botones para ejecutar el proceso 1, el 2 y detener el VI.

   - ❑ Añada una estructura cluster a la ventana del panel frontal.
   - ❑ Llame al cluster `Menu`.

   - ❑ Añada un botón Cancel a la estructura de cluster.
   - ❑ Llame al botón Cancel `Process 1`.
   - ❑ Cambie el texto booleano `Run Process 1`.
   - ❑ Realice una copia del botón Process 1 y colóquela en la estructura de cluster.
   - ❑ Llame al botón copiado `Process 2`.
   - ❑ Cambie el texto booleano del botón copiado a `Run Process 2`.

- ❑ Haga clic con el botón derecho en el botón y seleccione **Visible Items»Label** para ocultar las etiquetas.

- ❑ Añada un botón de parada a la estructura de cluster.
- ❑ Haga clic con el botón derecho en el botón **Stop** y seleccione **Visible Items»Label** para ocultar la etiqueta.
- ❑ Modifique el texto booleano en los botones usando los **Text Settings** de la barra de herramientas.

  Ajustes de texto sugeridos: Application Font negrita de 24 puntos.
- ❑ Amplíe y ordene los botones del cluster con la herramienta de cambio de tamaño y los siguientes botones de la barra de herramientas: **Align Objects**, **Distribute Objects** y **Resize Objects**.
- ❑ Haga clic con el botón derecho en el borde del cluster y seleccione **Autosizing»Size to Fit**.
- ❑ Haga clic con el botón derecho en el borde del cluster y seleccione **Visible Items»Label** para ocultar la etiqueta.

3. Cree la definición de tipo enum para controlar la máquina de estados.
   - ❑ Añada un enum a la ventana del panel frontal.
   - ❑ Haga clic con el botón derecho en el enum y seleccione **Edit Items**. Modifique la lista de este modo:

Elementos	Indicador digital
`Idle`	0
`Process 1`	1
`Process 2`	2
`Stop`	3

   - ❑ Haga clic en **OK** para salir del cuadro de diálogo **Enum Properties**.
   - ❑ Llame al control del enum `State Enum`.
   - ❑ Haga clic con el botón derecho en el State Enum y seleccione **Advanced»Customize**.
   - ❑ Seleccione **Type Def.** en el menú desplegable Control Type.

- ❑ Haga clic con el botón derecho en el Enum y seleccione **Representation»U32**.
- ❑ Guarde el control como `State Enum.ctl` en el directorio `<Exercises>\LabVIEW Basics I\State Machine`.
- ❑ Cierre la ventana Control Editor.
- ❑ Haga clic en **Yes** cuando se le pregunte si desea sustituir el control.
- ❑ Cambie al diagrama de bloques.
- ❑ Haga clic con el botón derecho en el State Enum y seleccione **Change to Constant**. El control enumerado ya no aparece en la ventana del panel frontal.

En los siguientes pasos, creará el diagrama de bloques de la figura 10-17. Este diagrama de bloques contiene cuatro estados: Idle, Process 1, Process 2 y Stop.

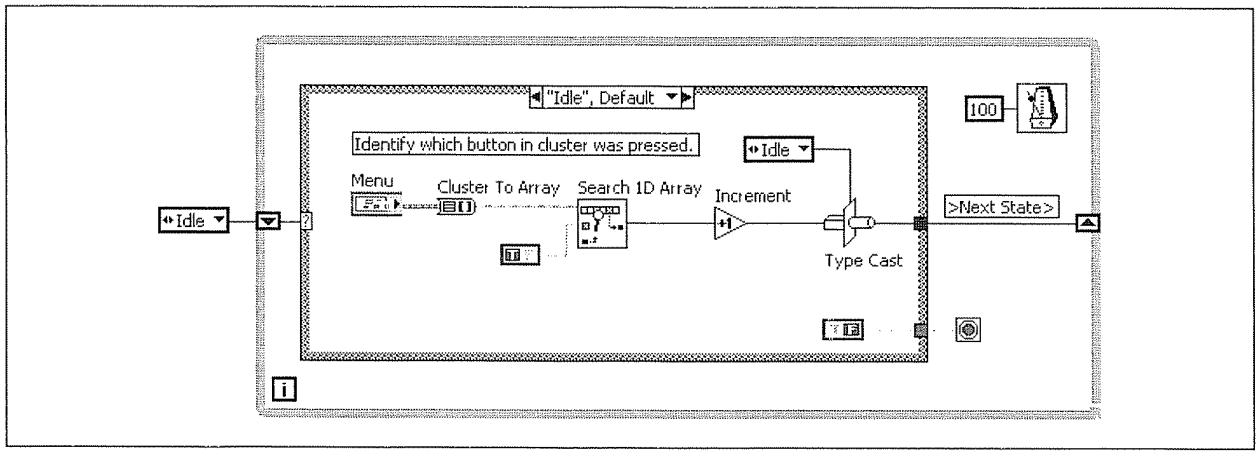


Figura 10-17. Estado Idle

4. Cree el diagrama de bloques de la figura 10-17. Use los siguientes consejos para ayudarle:
   - ❑ Cablee el enum a la estructura Case usando un registro de desplazamiento en el bucle While.
   - ❑ Tras cablear la constante enumerada al terminal de selección de caso de la estructura Case, haga clic con el botón derecho en la estructura Case y seleccione **Add Case for Every Value** para añadir automáticamente un caso para cada elemento del enum.
   - ❑ Copie el enum que se usará en la estructura Case. La copia también se vincula al enum con definición de tipo.

- ❑ Cablee una constante False al terminal condicional; la máquina de estados no debe detenerse al salir del estado Idle.

- ❑ En el estado Idle, convierta el cluster en un array para que en éste se pueda buscar el botón en el que se haga clic. La función Search 1D Array devuelve el índice del botón donde se ha hecho clic. Como el estado Idle no tiene un botón asociado, este índice debe aumentarse en uno. Use la función Type Cast para seleccionar el elemento apropiado desde la constante enum. Es muy importante que el orden del cluster coincida con el de los elementos de la constante enumerada.

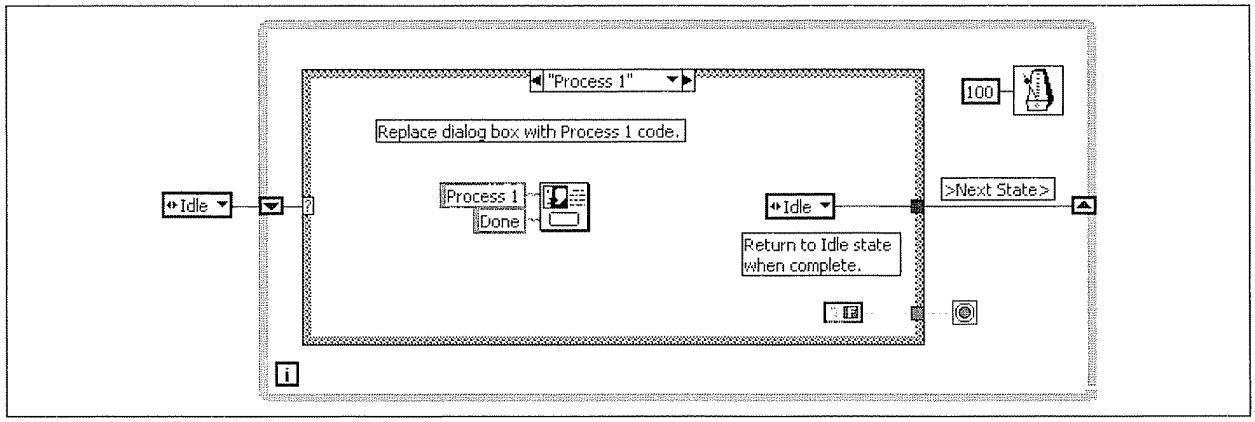


Figura 10-18. Estado Process 1

5. Complete el estado Process 1 como en la figura 10-18. Use los siguientes consejos para ayudarle:

   - ❑ Use la función One Button Dialog para simular el código del estado Process 1.

   - ❑ Cablee una constante False al terminal condicional; la máquina de estados no debe detenerse al salir del estado Process 1.

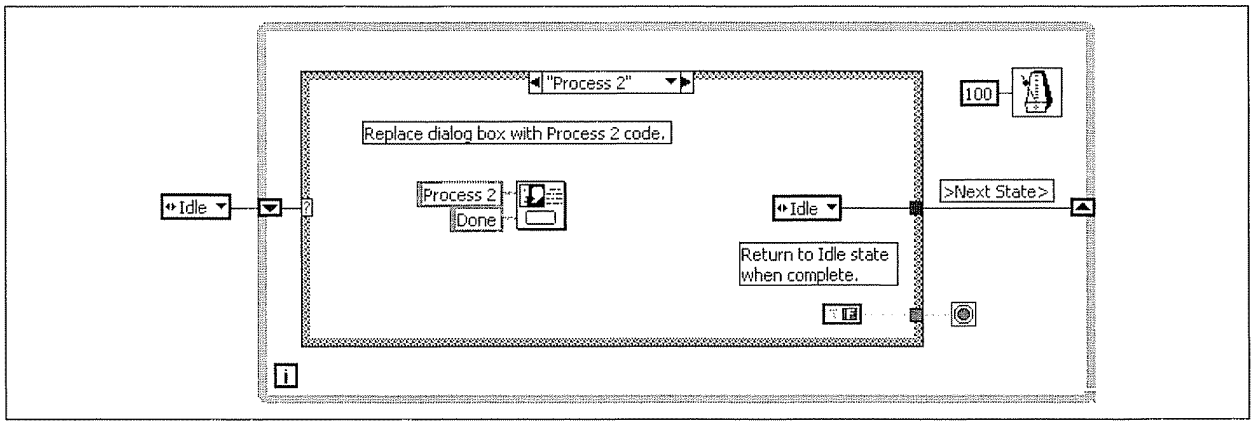


Figura 10-19. Estado Process 2

6. Complete el estado Process 2 como en la figura 10-19. Use los siguientes consejos para ayudarle:

   - ❑ Use la función One Button Dialog para simular el código del estado Process 2.
   - ❑ Cablee una constante False al terminal condicional; la máquina de estados no debe detenerse al salir del estado Process 2.

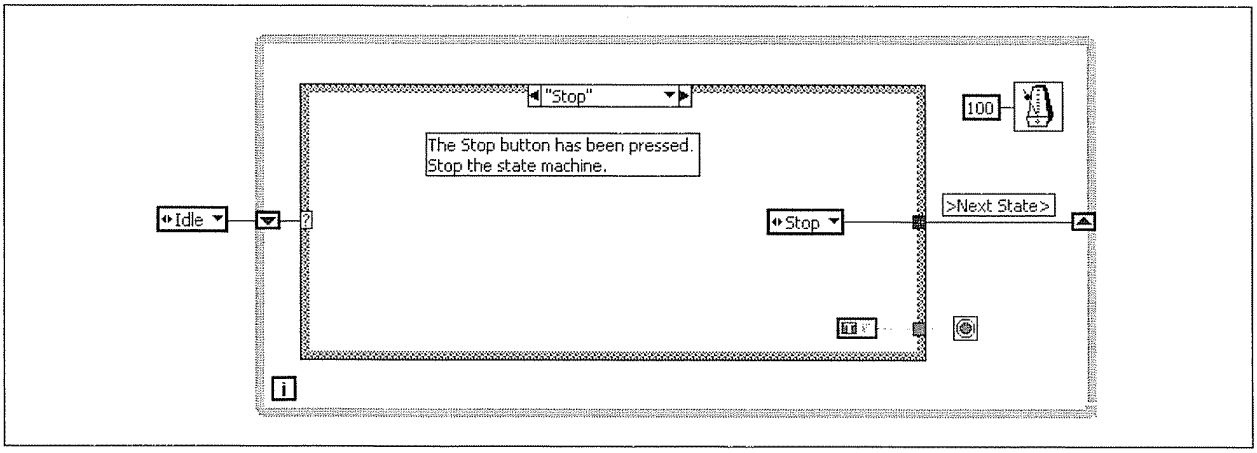


Figura 10-20. Estado Stop

7. Complete el estado Stop como en la figura 10-20. Use los siguientes consejos para ayudarle:

   - ❑ Pase una constante True al terminal condicional; la máquina de estados debe detenerse sólo desde este estado.

8. Guarde el VI tras terminar.

Prueba

1. Cambie a la ventana del panel frontal.
2. Ejecute el VI. Pruebe el VI para asegurarse de que funciona según lo esperado. En caso contrario, compare su diagrama de bloques con las figuras 10-17 a la 10-20.
3. Guarde y cierre el VI tras terminar.

Fin del ejercicio 10-1

D. Paralelismo

A menudo debe programar varias tareas para que se ejecuten a la vez. En LabVIEW las tareas pueden ejecutarse en paralelo si no tienen dependencia de datos entre sí y si no están usando el mismo recurso compartido. Un ejemplo de un recurso compartido es un archivo o un instrumento.

Aprenderá sobre los modelos de diseño de LabVIEW para ejecutar varias tareas a la vez en el curso *LabVIEW Básico II: Desarrollo.* Estos modelos de diseño incluyen bucles paralelos, maestro/esclavo y productor/consumidor.

Resumen

Usar una máquina de estados en lugar de una estructura de programación secuencial tiene las siguientes ventajas:

- Puede cambiar el orden de la secuencia.
- Puede repetir elementos en la secuencia.
- Puede configurar una condición para determinar cuándo debe ejecutarse un elemento de la secuencia.
- Puede detener el programa en cualquier punto de la secuencia.