Técnicas de diseño típicas

Puede desarrollar mejores programas en LabVIEW y en otros lenguajes de programación si sigue técnicas y arquitecturas de programación sistemáticas. Esta lección describe dos tipos de arquitecturas de programación: bucles simples y bucles múltiples. En conjunto, estas arquitecturas se conocen como patrones de diseño.

Las arquitecturas de bucle simple incluyen los patrones de diseño del VI simple, del VI general y de la máquina de estados.

Las arquitecturas de múltiples bucles son los patrones de diseño del VI de bucles paralelos, de maestro/esclavo y de productor/consumidor.

Comprender el uso apropiado de cada patrón de diseño ayuda a crear VIs de LabVIEW más eficientes.

Temas

A. Arquitecturas de bucle simple

B. Paralelismo

C. Arquitecturas de múltiples bucles

D. Eventos

E. Temporizar un patrón de diseño

A. Arquitecturas de bucle simple

Usted aprendió a diseñar tres tipos distintos de arquitecturas en el curso *LabVIEW Básico I: Introducción*: la arquitectura simple, la arquitectura general y la máquina de estados.

Patrones de diseño del VI simple

Para realizar cálculos o mediciones rápidas en el laboratorio, no necesita una arquitectura complicada. Su programa podría constar de un solo VI que realice una medición o un cálculo y muestre los resultados o los grabe en el disco. El patrón de diseño del VI simple normalmente no requiere una acción de inicio o parada específica por parte del usuario. El usuario sólo hace clic en el botón **Run**. Use esta arquitectura para aplicaciones sencillas o para componentes funcionales dentro de aplicaciones más grandes. Puede convertir estos VIs simples en subVIs que use como bloques de construcción para aplicaciones más grandes.

La figura 1-1 muestra el diagrama de bloques del VI Determine Warnings, que era el proyecto del curso *LabVIEW Básico I: Introducción.* Este VI ejecuta una sola tarea: determina qué alarma generar en función de un conjunto de entradas. Puede utilizar este VI como un subVI cuando tenga que determinar el nivel de alarma.

Observe que el VI de la figura 1-1 no contiene acciones de inicio o parada por parte del usuario. En este VI todos los objetos del diagrama de bloques se conectan mediante el flujo de datos. Puede determinar el orden general de las operaciones siguiendo el flujo de datos. Por ejemplo, la función Not Equal no se puede ejecutar hasta que se hayan ejecutado las funciones Greater Than or Equal, Less Than or Equal y ambas funciones Select.

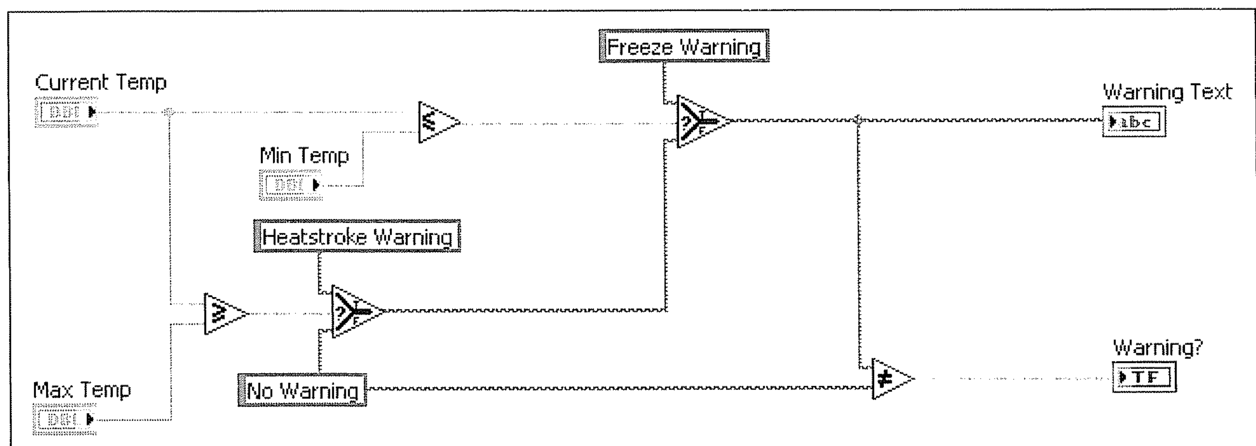

Figura 1-1. Arquitectura del VI simple

Patrones de diseño del VI general

Un patrón de diseño de VI general tiene tres fases principales: arranque, aplicación principal y cierre. Cada una de estas fases puede contener código que usa otro tipo de patrón de diseño.

- **Arranque**: inicializa el hardware, lee la información de configuración de los archivos o solicita al usuario ubicaciones de archivos de datos.
- **Aplicación principal**: consta de al menos un bucle que se repite hasta que el usuario decida salir del programa o éste termine por otras razones, como la finalización de E/S.
- **Cierre**: cierra archivos, escribe información de configuración en el disco o restablece E/S al estado predeterminado.

La figura 1-2 muestra el patrón de diseño del VI general.

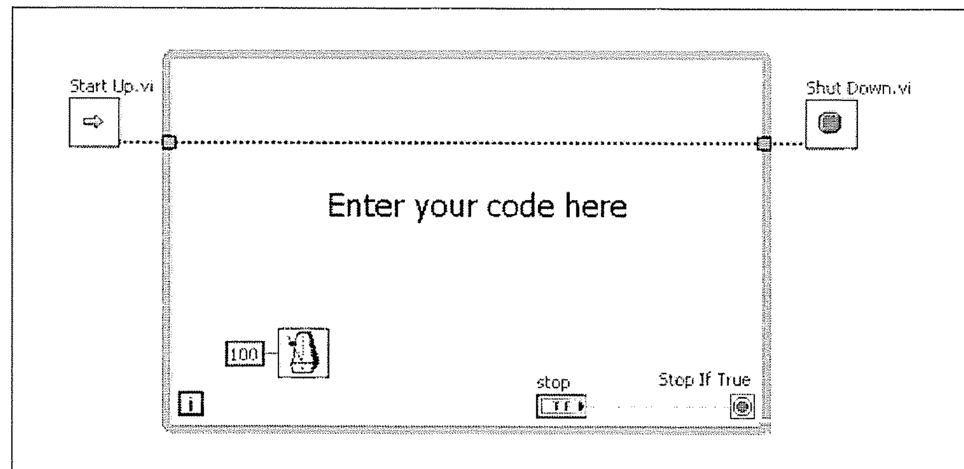

Figura 1-2. Patrón de diseño del VI general

En la figura 1-2, los cables del cluster de error controlan el orden de ejecución de las tres secciones. El bucle While no se ejecuta hasta que el VI Start Up termine de ejecutarse y devuelva los datos del cluster de error. Como consecuencia, el VI Shut Down no puede ejecutarse hasta que termine la aplicación principal del bucle While y los datos del cluster de error abandonen el bucle.

Consejo La mayoría de los bucles requieren una función Wait, especialmente si ese bucle monitoriza la interacción del usuario con el panel frontal. Sin la función Wait, el bucle podría ejecutarse continuamente y usar todos los recursos del sistema del ordenador. La función Wait obliga al bucle a ejecutarse asincrónicamente aunque especifique 0 milisegundos como periodo de espera. Si las operaciones del bucle principal reaccionan a las entradas del usuario, puede aumentar el periodo de espera a un nivel aceptable para los tiempos de reacción. Una espera de 100–200 ms suele ser buena porque la mayoría de los usuarios no puede detectar ese retardo entre hacer clic en un botón del panel frontal y la ejecución del evento siguiente.

Para aplicaciones simples, el bucle de la aplicación principal es obvio y contiene código que utiliza el patrón de diseño del VI simple. Cuando la aplicación incluye interfaces de usuario complicadas o varias tareas como acciones del usuario, triggers de E/S, etc., la fase de la aplicación principal se complica más.

Patrón de diseño de la máquina de estados

El patrón de diseño de la máquina de estados es una modificación del patrón de diseño general. Normalmente tiene una fase de arranque y de cierre. Sin embargo, la fase de la aplicación principal consta de una estructura Case embebida en el bucle. Esta arquitectura permite ejecutar diferente código cada vez que se ejecute el bucle, en función de alguna condición. Cada caso define un estado de la máquina, de ahí el nombre máquina de estados. Use este patrón de diseño para VIs que se dividan fácilmente en varias tareas más simples, como VIs que actúan como una interfaz de usuario.

Una máquina de estados en LabVIEW consta de un bucle While, una estructura Case y un registro de desplazamiento. Cada estado de la máquina de estados es un caso distinto en la estructura Case. Debe colocar VIs y otro código que tenga que ejecutar el estado dentro del caso apropiado. Un registro de desplazamiento almacena el estado que debe ejecutarse en la siguiente iteración del bucle. El diagrama de bloques de un VI de máquina de estados con cinco estados aparece en la figura 1-3. La figura 1-4 muestra los otros casos, o estados, de la máquina de estados.

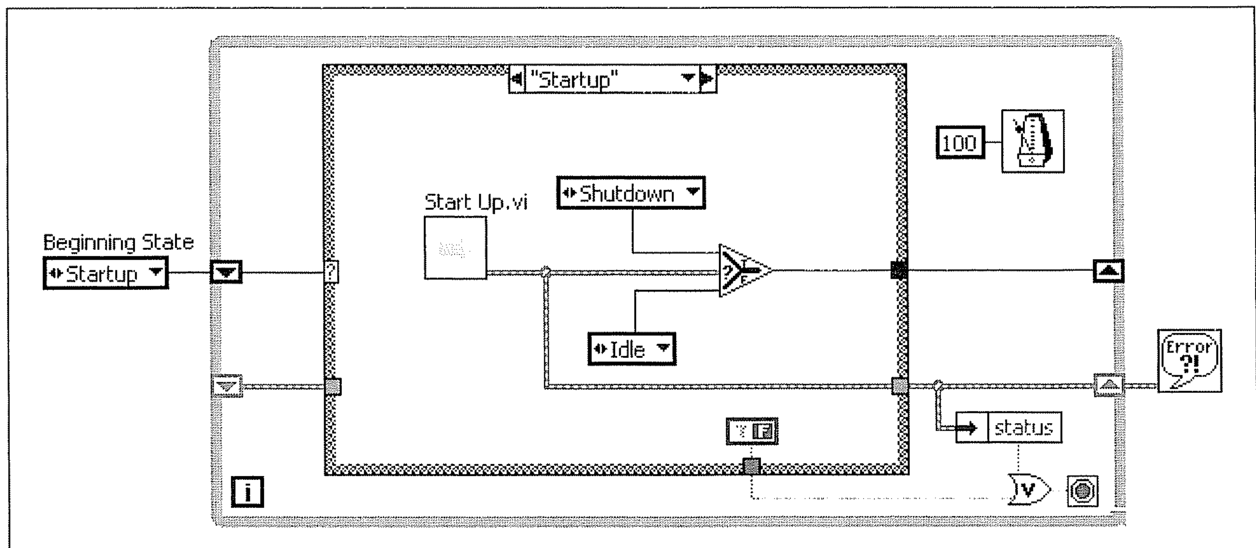

Figura 1-3. Máquina de estados con estado de arranque

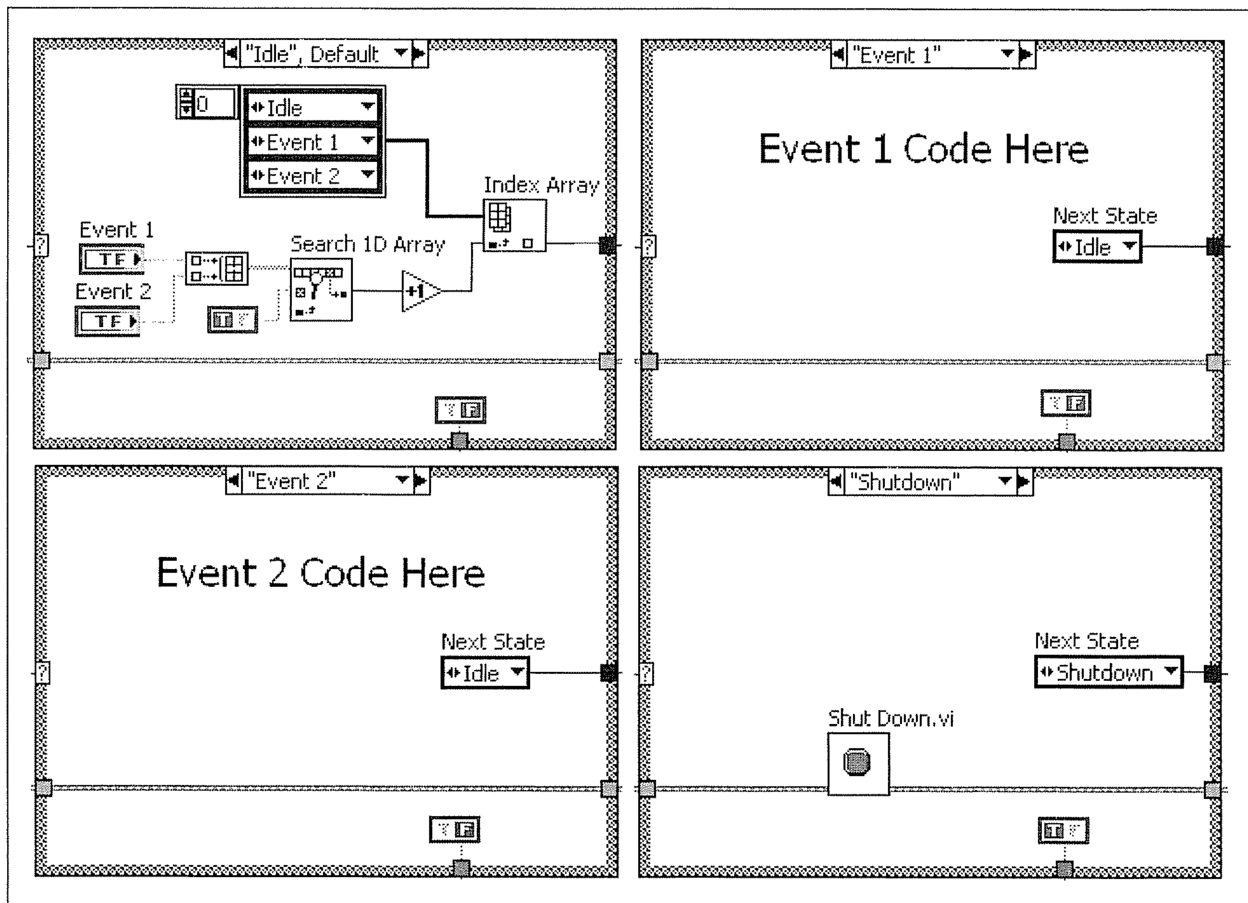

Figura 1-4. Estados inactivo (predeterminado), evento 1, evento 2 y cierre

En el patrón de diseño de la máquina de estados, usted diseña la lista de posibles tareas, o estados, y después los asigna a cada caso. Para el VI del ejemplo anterior, los posibles estados son Startup, Idle, Event 1, Event 2 y Shutdown. Una constante de tipo enumerada almacena los estados. Cada estado tiene su propio caso en la estructura Case. El resultado de un caso determina qué caso ejecutar después. El registro de desplazamiento almacena el valor que determina el próximo caso a ejecutar.

El patrón de diseño de la máquina de estados puede hacer el diagrama de bloques mucho más pequeño y, por lo tanto, más fácil de leer y de depurar. Otra ventaja de la arquitectura de la máquina de estados es que cada caso determina el siguiente estado, a diferencia de las estructuras Sequence, que deben ejecutar cada marco secuencialmente.

Una desventaja del patrón de diseño de la máquina de estados es que con el enfoque del ejemplo anterior, es posible saltar estados. Si se llama a dos estados de la estructura a la vez, este modelo controla sólo un estado, y el otro estado no se ejecuta. Saltar estados puede producir errores difíciles de depurar porque son complicados de reproducir. Versiones más complejas del patrón de diseño de la máquina de estados contienen código adicional

que crea una cola de eventos, o estados, para que no se pierda un estado. Consulte la lección 2, *Comunicación entre múltiples bucles*, para obtener información adicional acerca de las colas.

B. Paralelismo

En este curso el paralelismo hace referencia a ejecutar varias tareas a la vez. Piense en el ejemplo de crear y mostrar dos ondas sinusoidales a distintas frecuencias. Con el paralelismo, coloque una onda sinusoidal en un bucle y la segunda onda sinusoidal en otro bucle.

Un reto al programar tareas paralelas es pasar datos entre varios bucles sin crear una dependencia de datos. Por ejemplo, si pasa los datos usando un cable, los bucles ya no serán paralelos. En el ejemplo de la onda sinusoidal múltiple, quizá desee compartir un mecanismo de parada simple entre los bucles, como en la figura 1-5.

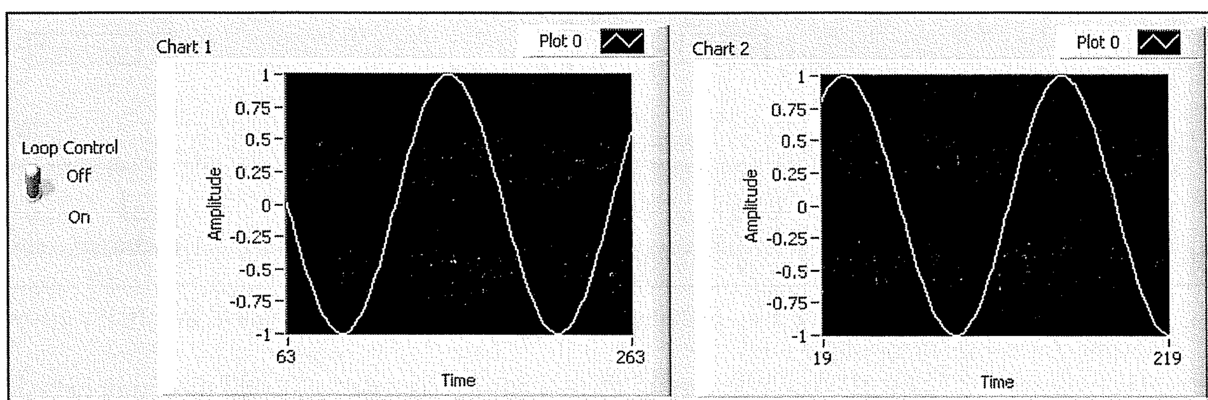

Figura 1-5. Panel frontal de bucles paralelos

Compruebe lo que sucede cuando intenta compartir datos entre bucles paralelos con un cable usando diferentes métodos.

Método 1 (incorrecto)

Coloque el terminal **Loop Control** fuera de ambos bucles y cabléelo a cada terminal condicional, como en la figura 1-6. El Loop Control es una entrada de datos de ambos bucles, por lo que el terminal **Loop Control** se lee sólo una vez, antes de que cualquier bucle While empiece a ejecutarse. Si False pasa a los bucles, los bucles While se ejecutan indefinidamente. Si apaga el interruptor, no detendrá el VI, porque el interruptor no se lee durante la iteración de cada bucle.

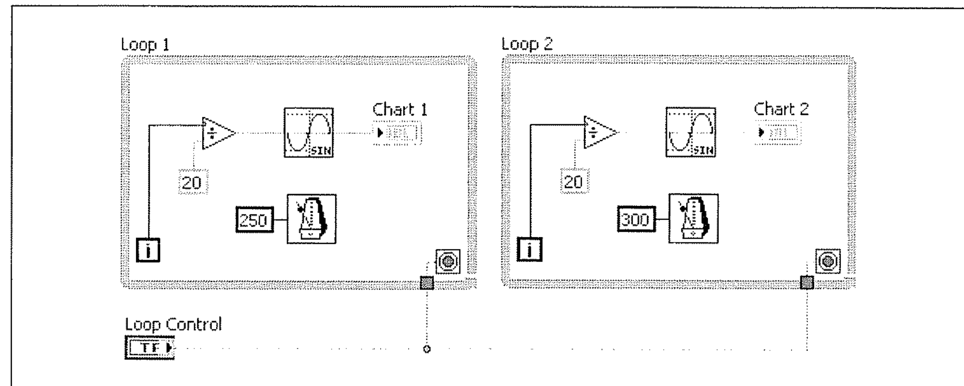

Figura 1-6. Ejemplo de método 1 de bucles paralelos

Método 2 (incorrecto)

Introduzca el terminal **Loop Control** dentro del Loop 1 para que se lea en cada iteración del Loop 1, como se muestra en el siguiente diagrama de bloques. Aunque el Loop 1 termina correctamente, el Loop 2 no se ejecuta hasta que recibe todas sus entradas de datos. El Loop 1 no pasa los datos fuera del bucle hasta que se detenga el bucle, por lo que el Loop 2 debe esperar al valor final del **Loop Control**, disponible cuando termine el Loop 1. Por lo tanto, los bucles no se ejecutan en paralelo. Asimismo, el Loop 2 se ejecuta sólo para una iteración porque su terminal condicional recibe un valor True del interruptor **Loop Control** del Loop 1.

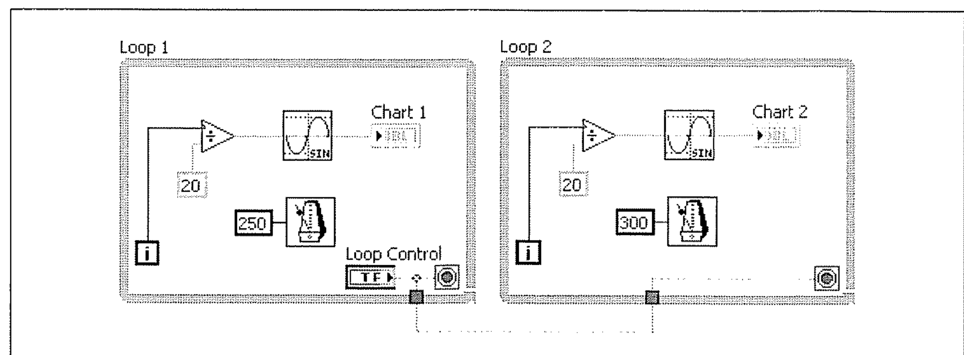

Figura 1-7. Ejemplo de método 2 de bucles paralelos

Método 3 (solución)

Si pudiera leer el valor del control de bucle desde un archivo, ya no dependería de un flujo de datos entre los bucles, ya que cada bucle puede acceder al archivo de forma independiente. Sin embargo, leer y escribir en archivos puede ser lento, al menos en cuanto a tiempo de procesador. Otra forma de realizar esta tarea es buscar la ubicación donde los datos de control del bucle estén almacenados en la memoria y leer esa ubicación de memoria directamente. Consulte la lección 2, *Comunicación entre múltiples bucles*, para obtener información sobre métodos para solucionar este problema.

C. Arquitecturas de múltiples bucles

Aprendió varias razones para usar el paralelismo en la sección anterior. Esta sección describe las siguientes arquitecturas de múltiples bucles: bucle paralelo, maestro/esclavo y productor/consumidor de datos.

Patrón de diseño del bucle paralelo

Algunas aplicaciones deben responder a varias tareas y ejecutarlas a la vez. Un modo de mejorar el paralelismo es asignar un bucle distinto a cada tarea. Por ejemplo, quizá tenga un bucle distinto para cada botón del panel frontal y para cualquier otro tipo de tarea, como una selección de menú, trigger de E/S, etc. La figura 1-8 muestra este patrón de diseño de bucles paralelos.

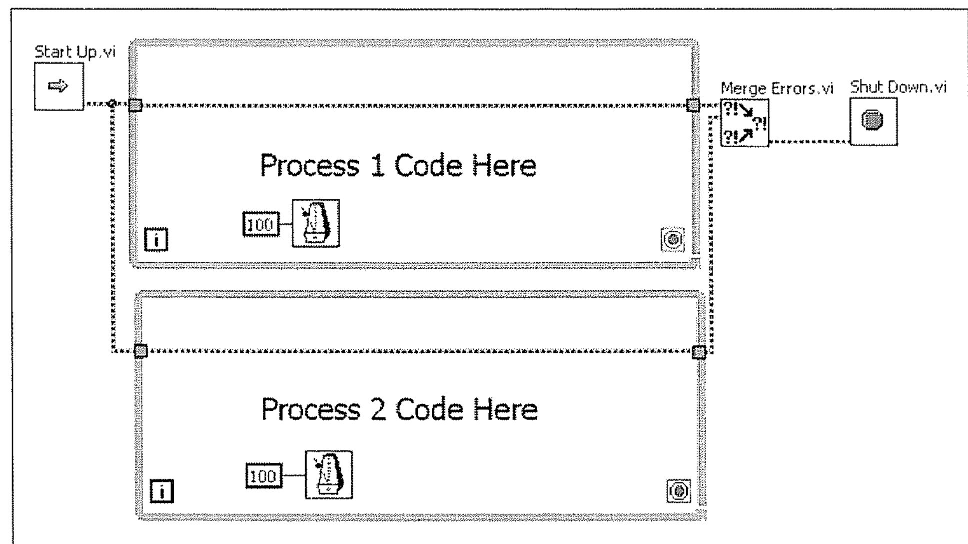

Figura 1-8. Patrón de diseño de bucles paralelos

Esta estructura es directa y apropiada para algunos VIs de menús simples, donde un usuario debe seleccionar uno de los botones que realizan distintas acciones. El patrón de diseño de bucles paralelos permite controlar varias tareas simultáneas e independientes. En este patrón de diseño, responder a una acción no impide que el VI responda a otra acción. Por ejemplo, si un usuario hace clic en un botón que muestra un cuadro de diálogo, los bucles paralelos pueden seguir respondiendo a tareas de E/S.

Sin embargo, en el patrón de diseño de bucles paralelos debe coordinarse y comunicarse entre varios bucles. El botón **Stop** del segundo bucle en la figura 1-8 es una variable local. No puede usar cables para pasar datos entre bucles porque impedirá que los bucles se ejecuten en paralelo. Debe utilizar una técnica de comunicación para pasar información entre procesos. Consulte la lección 2, *Comunicación entre múltiples bucles*, para obtener información adicional acerca del uso de variables locales, notificadores o colas para comunicación entre bucles paralelos.

Patrón de diseño de maestro/esclavo

El patrón de diseño de maestro/esclavo consta de varios bucles paralelos. Cada uno de los bucles puede ejecutar tareas a distintas velocidades. Un bucle actúa como el maestro y los otros como esclavos. El bucle maestro controla todos los bucles esclavos y se comunica con ellos mediante técnicas de comunicación, como en la figura 1-9.

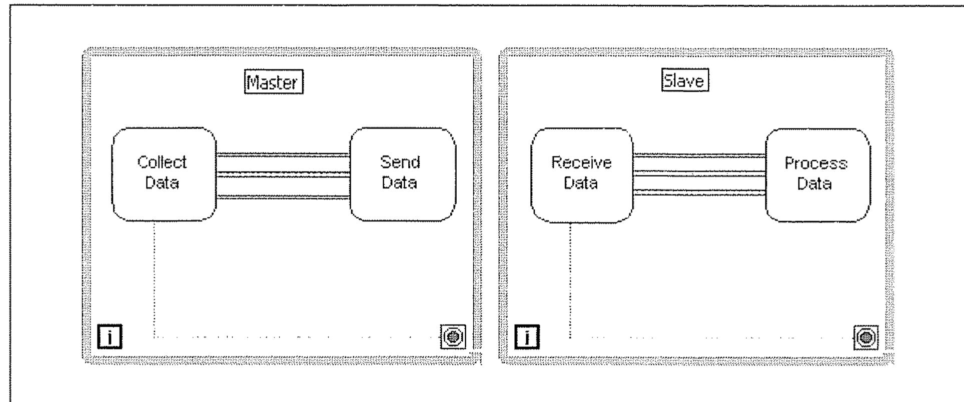

Figura 1-9. Patrón de diseño de maestro/esclavo

Use el patrón de diseño de maestro/esclavo cuando un VI tenga que responder a controles de interfaz de usuario mientras recopila datos simultáneamente. Por ejemplo, desea crear un VI que mida y registre una tensión que varía lentamente una vez cada cinco segundos. El VI adquiere una forma de onda desde una línea de transmisión y la muestra en un gráfico tipo "graph" cada 100 ms. El VI también incluye una interfaz de usuario para que éste cambie los parámetros para cada adquisición. El patrón de diseño de maestro/esclavo es idóneo para esta aplicación de adquisición. Para esta aplicación, el bucle maestro contiene la interfaz del usuario. La adquisición de tensión ocurre en un bucle esclavo, mientras que la presentación de gráficos se realiza en otro bucle esclavo.

Con el enfoque estándar del patrón de diseño de maestro/esclavo para este VI, situaría los procesos de adquisición en dos bucles While diferentes, ambos controlados por un bucle maestro que recibe entradas de los controles de la interfaz de usuario. Esto garantiza que los procesos de adquisición no interfieran entre sí y que cualquier retardo ocasionado por la interfaz de usuario, como mostrar un cuadro de diálogo, no retrase iteraciones de los procesos de adquisición.

Los VIs que implican control también se benefician del uso de patrones de diseño de maestro/esclavo. Piense en un VI en el que un usuario controle un brazo robótico de movimiento libre mediante botones de un panel frontal. Este tipo de VI requiere un control eficiente, preciso y sensible, por los daños físicos al brazo o alrededores que podrían ocurrir si el control se administra mal. Por ejemplo, si el usuario indica al brazo que detenga su

movimiento hacia abajo, pero el programa está ocupado con el control giratorio del brazo, el brazo robótico podría chocar con la plataforma de soporte. Aplique el patrón de diseño de maestro/esclavo a la aplicación para evitar estos problemas. En este caso, el bucle maestro controla la interfaz del usuario y cada sección controlable del brazo robótico tiene su propio bucle esclavo. Como cada sección controlable del brazo tiene su propio bucle y su propio tiempo de procesado, la interfaz del usuario tiene un control más sensible del brazo robótico.

Con un patrón de diseño de maestro/esclavo, es importante que dos bucles While no escriban en los mismos datos compartidos. No más de un bucle While debe escribir en datos compartidos. Consulte la lección 2, *Comunicación entre múltiples bucles*, para obtener información adicional acerca de datos compartidos.

El esclavo no debe tardar demasiado en responder al maestro. Si el esclavo está procesando una señal del maestro y éste envía más de un mensaje al esclavo, éste recibirá sólo el último mensaje. Este uso de la arquitectura de maestro/esclavo podría causar una pérdida de datos. Use una arquitectura de maestro/esclavo sólo si está seguro de que cada tarea de esclavo tarda menos tiempo en ejecutarse que el bucle maestro.

Patrón de diseño de productor/consumidor

El patrón de diseño de productor/consumidor se basa en el patrón de diseño de maestro/esclavo y mejora la compartición de datos entre varios bucles que se ejecutan a distintas velocidades. Al igual que el patrón de diseño de maestro/esclavo, el patrón de diseño de productor/consumidor separa tareas que producen y consumen datos a distintas velocidades. Los bucles paralelos en el patrón de diseño de productor/consumidor están separados en dos categorías: los que producen datos y los que consumen los datos producidos. Las colas de datos comunican datos entre los bucles. Las colas de datos también almacenan temporalmente datos en un buffer entre los bucles productor y consumidor.

Consejo Un búfer es un dispositivo de memoria que almacena datos temporales entre dos dispositivos o, en este caso, múltiples bucles.

Use el patrón de diseño de productor/consumidor cuando tenga que adquirir varios conjuntos de datos que deben procesarse en orden. Suponga que desea crear un VI que acepte datos mientras procesa los conjuntos de datos en el orden en que se recibieron. El patrón de productor/consumidor es ideal para este tipo de VI, porque poner en cola (producir) los datos se realiza mucho más rápidamente que el procesamiento de éstos (consumir). Podría colocar al productor y al consumidor en el mismo bucle para esta aplicación, pero la cola de procesamiento no podría recibir más datos hasta que el primer dato se procesara completamente. El enfoque de

productor/consumidor para este VI pone en cola los datos en el bucle productor y procesa los datos en el bucle consumidor, como en la figura 1-10.

Consejo Las funciones de manejo de colas permiten almacenar un conjunto de datos que puede pasarse entre bucles múltiples que se ejecutan simultáneamente o entre VIs. Consulte la lección 2, *Comunicación entre múltiples bucles*, para obtener información adicional acerca de colas.

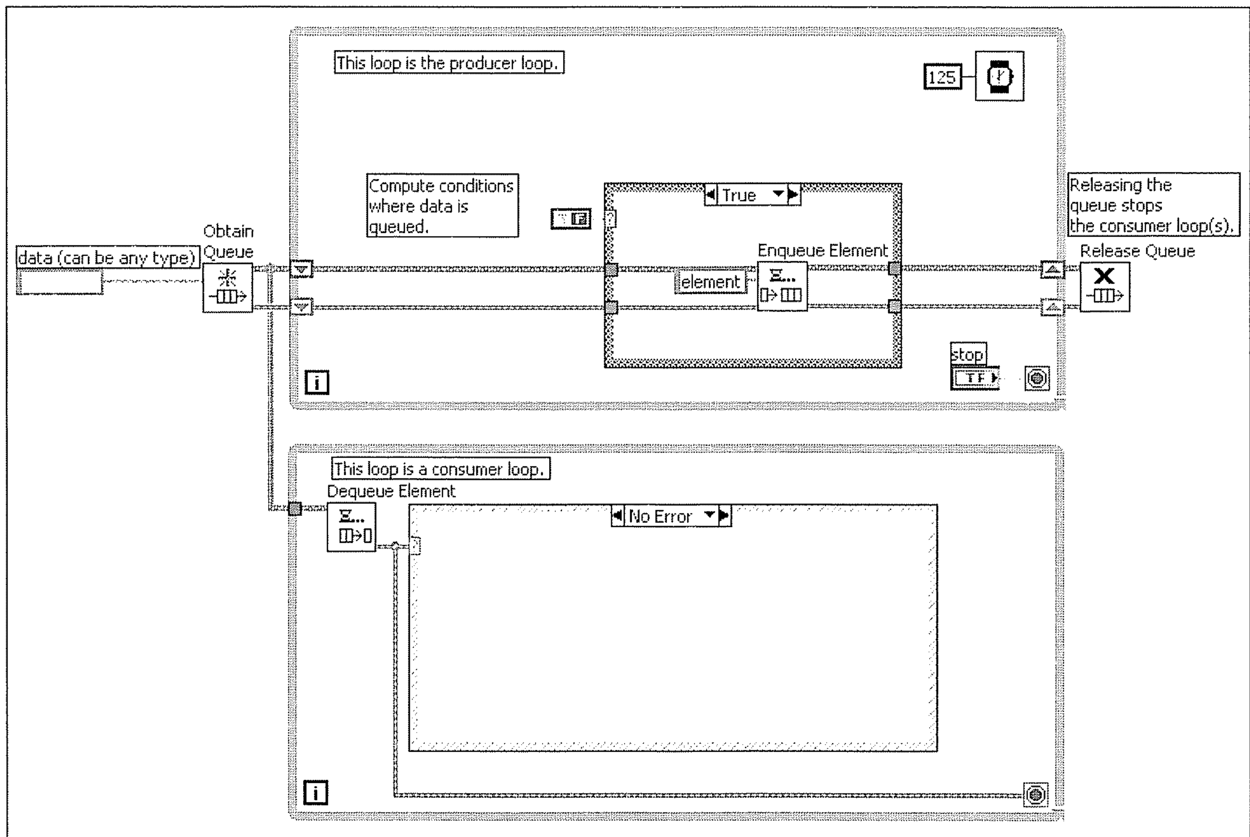

Figura 1-10. Patrón de diseño de productor/consumidor

Este patrón de diseño permite que el bucle consumidor procese los datos a su propio ritmo, mientras que el bucle productor sigue poniendo en cola datos adicionales.

También puede usar el patrón de diseño de productor/consumidor para crear un VI que analice la comunicación de red. Este tipo de VI requiere dos procesos simultáneos y a distintas velocidades. El primer proceso sondea constantemente la línea de red y captura paquetes. El segundo proceso analiza los paquetes que captura el primer proceso.

En este ejemplo, el primer proceso actúa como productor porque suministra datos al segundo proceso, que actúa de consumidor. El patrón de diseño de productor/consumidor es una arquitectura efectiva para este VI. Los bucles paralelos productor y consumidor controlan la captura y el análisis de datos fuera de la red. La comunicación en cola entre los dos bucles permite el almacenamiento temporal de los paquetes de red recuperados. El uso del buffer puede ser importante si la comunicación de red es intensa. Mediante el buffer, los paquetes pueden capturarse y comunicarse más rápidamente de lo que pueden analizarse.

D. Eventos

LabVIEW es un entorno de programación de flujo de datos donde éste determina el orden de ejecución de elementos del diagrama de bloques. Las características de programación orientada a eventos amplían el entorno de flujo de datos de LabVIEW para permitir que el usuario interactúe directamente con el panel frontal y permitir otra actividad asíncrona que influya más en la ejecución del diagrama de bloques.

Nota Las funciones de programación orientada a eventos sólo están disponibles en los Sistemas de desarrollo Full y Professional de LabVIEW. Puede ejecutar un VI creado con estas funciones de LabVIEW Base Package, pero no puede volver a configurar los componentes que controlan eventos.

¿Qué son los eventos?

Un evento es una notificación asíncrona de que algo ha ocurrido. Los eventos pueden provenir de la interfaz de usuario, E/S externa u otras partes del programa. Los eventos de la interfaz de usuario son clics del ratón, pulsaciones de teclas, etc. Los eventos de E/S externa son temporizadores o triggers de hardware que señalan cuándo se completa la adquisición de datos o cuándo ocurre una condición de error. Otros tipos de eventos pueden generarse programáticamente y utilizarse para comunicarse con distintas partes del programa. LabVIEW admite eventos generados por la interfaz de usuario o programáticamente. LabVIEW también admite eventos generados por ActiveX y .NET, que son eventos de E/S externa.

En un programa orientado a eventos, los eventos que ocurren en el sistema influyen directamente en el flujo de ejecución. Por el contrario, un programa de procedimiento se ejecuta en un orden predeterminado y secuencial. Los programas orientados a eventos suelen incluir un bucle que espera que ocurra un evento, ejecuta el código para responder al evento y se vuelve a iterar para esperar al siguiente evento. El modo en que el programa responde a cada evento depende del código escrito para ese evento. El orden en que se ejecuta un programa orientado a eventos depende de qué eventos ocurran y en qué orden ocurran. Algunas secciones del programa podrían ejecutarse

con frecuencia porque los eventos que controlan ocurren frecuentemente. Quizá no se ejecuten otras secciones del programa porque los eventos nunca ocurren.

¿Por qué usar eventos?

Use eventos de interfaz de usuario en LabVIEW para sincronizar acciones del usuario en el panel frontal con la ejecución del diagrama de bloques. Los eventos permiten ejecutar un caso de control de eventos específico cada vez que un usuario realice una acción concreta. Sin los eventos, el diagrama de bloques debe sondear el estado de objetos del panel frontal en un bucle, comprobando si ha ocurrido algún cambio. Sondear el panel frontal requiere una cantidad de tiempo significativa de la CPU y quizá no detecte cambios si ocurren con demasiada rapidez. Al usar eventos para responder a acciones específicas del usuario, no necesita sondear el panel frontal para determinar qué acciones realizó el usuario. LabVIEW notifica activamente al diagrama de bloques cada vez que ocurre una interacción que especificó. El uso de eventos reduce los requisitos de CPU del programa, simplifica el código del diagrama de bloques y garantiza que el diagrama de bloques pueda responder a todas las interacciones que realice el usuario.

Use eventos generados programáticamente para comunicarse entre varias partes del programa que no tengan dependencia del flujo de datos. Los eventos generados programáticamente comparten muchas ventajas de los eventos de interfaz de usuario y pueden compartir el mismo código de control de eventos, lo que facilita la implementación de arquitecturas avanzadas, como las máquinas de estados con colas que usan eventos.

Consulte la lección 3, *Programación de eventos*, para obtener información adicional acerca de los eventos.

E. Temporizar un patrón de diseño

Esta sección describe dos formas de temporización: de ejecución y de control de software. La temporización de ejecución usa funciones de temporización para facilitar al procesador el tiempo para completar otras tareas. La temporización de control de software temporiza una operación del mundo real para realizarla en un periodo de tiempo concreto.

Temporización de ejecución

La temporización de ejecución implica temporizar un patrón de diseño explícitamente o en función de eventos que ocurren en el VI. La temporización explícita utiliza una función que permite específicamente tiempo al procesador para completar otras tareas, como la función Wait Until Next ms Multiple. Cuando la temporización se basa en eventos, el patrón de diseño espera a que ocurra alguna acción antes de continuar y permite que el procesador complete otras tareas mientras espera.

Use la temporización explícita para patrones de diseño como maestro/esclavo, productor/consumidor y máquina de estados. Estos patrones de diseño realizan algún tipo de sondeo mientras se ejecutan.

Consejo El sondeo es el proceso de realizar solicitudes continuas de datos desde otro dispositivo. En LabVIEW, esto normalmente significa que el diagrama de bloques pregunta continuamente si hay datos disponibles, en general desde la interfaz de usuario.

Por ejemplo, el patrón de diseño de maestro/esclavo de la figura 1-11 usa un bucle While y una estructura Case para implementar el bucle maestro. El maestro ejecuta continuamente y sondea un evento de algún tipo, como cuando el usuario hace clic en un botón. Cuando ocurre el evento, el maestro envía un mensaje al esclavo. Debe temporizar el maestro para que no acapare la ejecución del procesador. En este caso, normalmente use la función Wait (ms) para regular la frecuencia de sondeo del maestro.

Consejo Use siempre una función de temporización como Wait (ms) o Wait Until Next ms Multiple en cualquier patrón de diseño que se ejecute continuamente y deba regularse. Si no usa una función de temporización en una estructura que se ejecuta continuamente, LabVIEW usará todo el tiempo del procesador y quizá no se ejecuten los procesos en segundo plano.

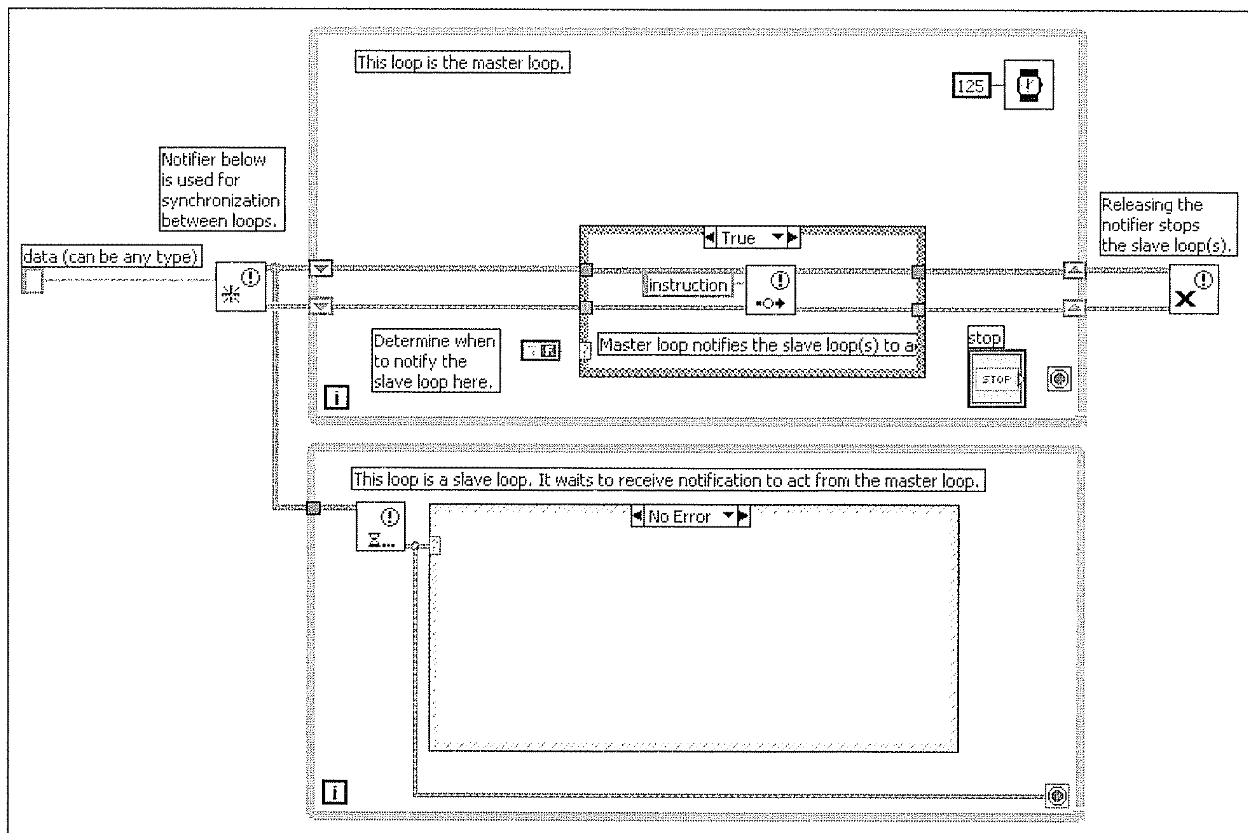

Figura 1-11. Patrón de diseño de maestro/esclavo

Tenga en cuenta que el bucle esclavo no contiene ninguna forma de temporización. El uso de las funciones de sincronización, como las colas y los notificadores, para pasar mensajes, proporciona una forma inherente de temporización en el bucle esclavo, porque éste espera a que la función Notifier reciba un mensaje. Una vez que la función Notifier reciba un mensaje, el esclavo lo procesará. Así se crea un diagrama de bloques eficiente que no malgasta ciclos del procesador al sondear mensajes. Éste es un ejemplo de temporización de ejecución esperando un evento.

Cuando implementa patrones de diseño en los que la temporización se basa en la existencia de eventos, no tiene que determinar la frecuencia de temporización correcta, ya que la ejecución del patrón de diseño ocurre sólo cuando tiene lugar un evento. En otras palabras, el patrón de diseño se ejecuta sólo cuando recibe un evento.

Temporización de control del software

Muchas aplicaciones que crea deben ejecutar una operación durante un tiempo específico. Piense en la implementación de un patrón de diseño de máquina de estados para un sistema de adquisición de datos de temperatura. Si las especificaciones requieren que el sistema adquiera los datos de temperatura durante 5 minutos, usted podría permanecer en el estado de adquisición durante 5 minutos. Sin embargo, durante ese tiempo no puede procesar ninguna acción de la interfaz de usuario como detener el VI. Para procesar acciones de la interfaz de usuario, debe implementar la temporización, de modo que el VI se ejecute continuamente durante el tiempo especificado. Implementar este tipo de temporización conlleva que la aplicación se esté ejecutando mientras monitoriza un reloj en tiempo real.

En el curso *LabVIEW Básico I: Introducción* implementó temporización de control de software para controlar el tiempo hasta que el VI adquiera el siguiente dato, como en la figura 1-12. Observe el uso del VI Express Elapsed Time para realizar el seguimiento de un reloj.

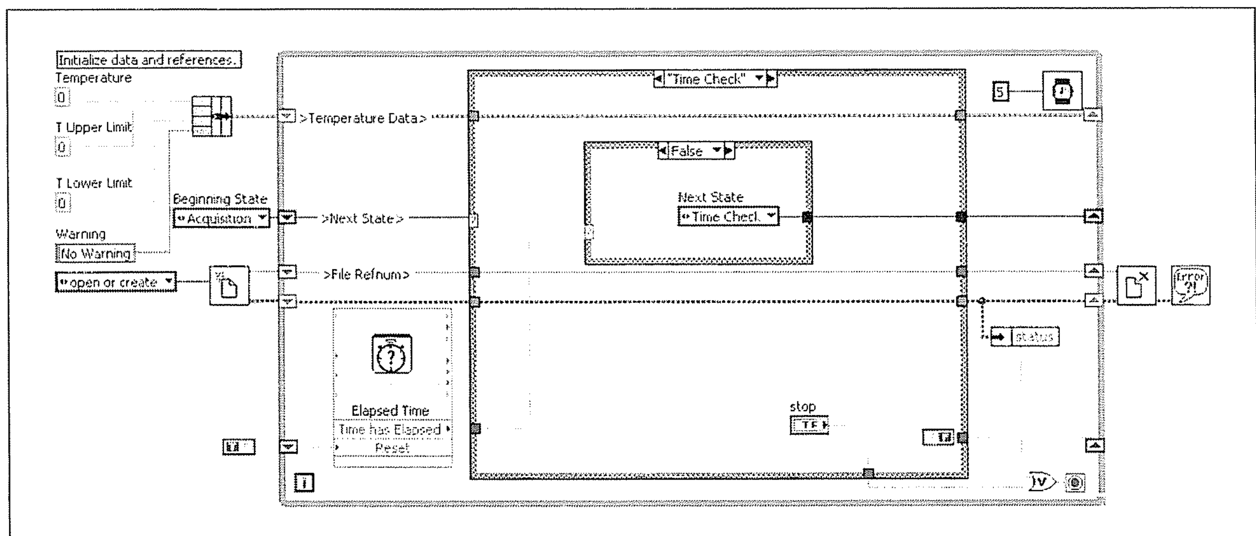

Figura 1-12. Uso del VI Express Elapsed Time

Si usa las funciones Wait (ms) o Wait Until Next ms Multiple para realizar temporización de software, la ejecución de la función que está temporizando no ocurrirá hasta que termine la función de espera. Estas funciones de temporización no son el método preferido para realizar la temporización de control de software, sobre todo para VIs en los que debe ejecutarse continuamente el sistema. Un buen patrón para usar en la temporización de control de software cicla el tiempo actual en todo el VI, como en la figura 1-13.

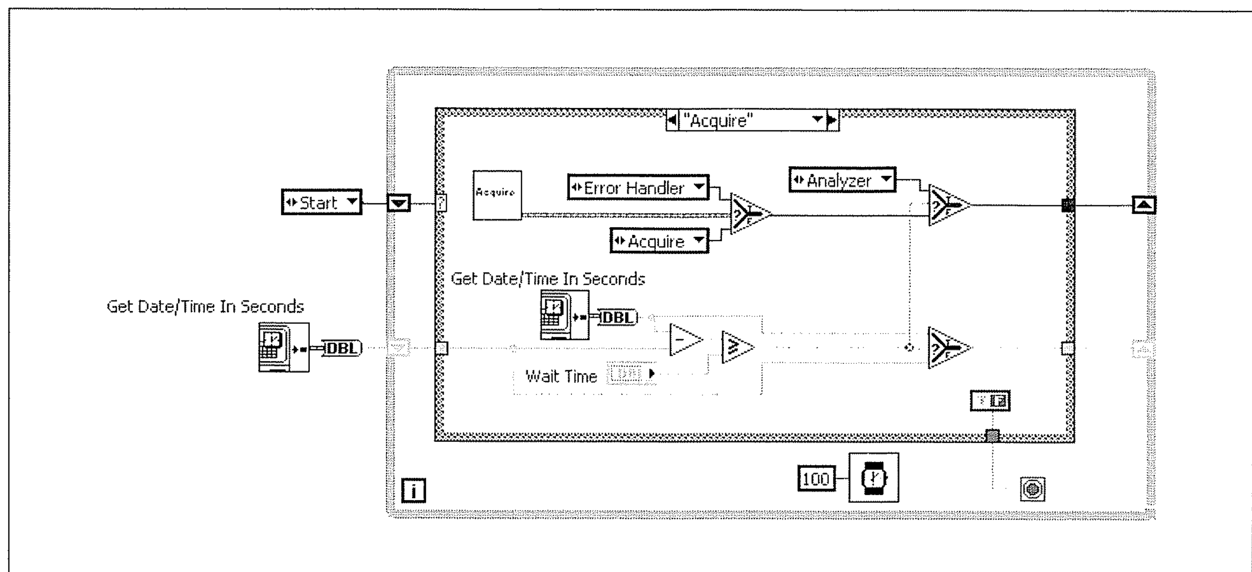

Figura 1-13. Temporización de software usando la función Get Date/Time In Seconds

La función Get Date/Time In Seconds, conectada al terminal izquierdo del registro de desplazamiento, inicializa éste con la hora del sistema actual. Cada estado usa otra función Get Date/Time In Seconds y compara la hora actual con la hora de inicio. Si la diferencia entre estas dos horas es mayor o igual que el tiempo de espera, el estado terminará de ejecutarse y se ejecutará el resto de la aplicación.

Consejo Use siempre la función Get Date/Time In Seconds en lugar de la función Tick Count para este tipo de comparación, porque el valor de la función Tick Count puede volver a 0 durante la ejecución.

Consulte la lección 2, *Comunicación entre múltiples bucles*, para obtener información adicional acerca de crear una variable global funcional de temporización para hacer la funcionalidad de temporización modular y reutilizable.

Autorrevisión: cuestionario

1. La temporización de control de software facilita al procesador el tiempo para completar otras tareas.
   a. Verdadero
   b. Falso
2. La temporización de ejecución es un método para facilitar al procesador el tiempo para completar otras tareas.
   a. Verdadero
   b. Falso
3. Puede usar un cable para pasar datos entre dos bucles paralelos.
   a. Verdadero
   b. Falso

Autorrevisión: respuestas al cuestionario

1. FALSO: la temporización de control de software es un método para controlar un reloj en tiempo real.
2. VERDADERO: la temporización de ejecución es un método para facilitar al procesador el tiempo para completar otras tareas.
3. FALSO: si pasa los datos entre dos bucles usando un cable, los bucles ya no se ejecutarán en paralelo.